TESIS DE MAGISTER

EN INGENIERÍA DEL SOFTWARE

SISTEMA PARA EL TRAZADO DEL APRENDIZAJE DE LAS REGLAS DE UN JUEGO DE INGENIO POR PARTE DE NIÑOS CON SÍNDROME DE DOWN

Autor : Ing. Walter Otto Krause

Directores : Dr. Ramón García Martinez

M. Ing. Paola Britos

M. Ing. Sandra Gomez

Buenos Aires, Marzo de 2006

Dedicatoria:

A mi esposa Alejandra, fuente de Amor y Comprensión de toda mi vida,
a mis hijos Ezequiel y Leonel, fuente de inspiración de todos mis proyectos.

A los que me acompañan siempre (Papá, Pilar);
a los que me acompañaron siempre
y ahora no están (Mamá, Abuela Frida, Abuela Adela).

Finalmente, a todos y cada uno de los chicos con necesidades educativas especiales,
con el deseo que cada día podamos enseñarles más a partir de sus habilidades
y no desde nuestras limitaciones.

Agradecimientos:

Al Ing. Pablo Ghidoli, compañero de aventuras tecnológicas.

Al Lic. Ariel Bertotto y en él a todo SoftLab, por su apoyo.

A la M. Ing. Paola Britos por haberme guiado en la organización de este proyecto
y corregido el texto con paciencia inagotable.

Al Dr. Ramón García Martínez por su apoyo en el tema de tesis elegido
y su permanente estímulo a la concreción del mismo.

A la M.Ing. Sandra Gomez quien con sus aportes y correcciones
ayudó a mejorar la calidad del presente documento de tesis.

Resumen

Sabidas las dificultades que poseen los niños con Síndrome de Down para aprender a partir del lenguaje expresivo oral, ¿Se puede a partir de la evaluación de su desempeño ante un juego interactivo inferir la mejor manera de aprender a través de estímulos visuales y a partir de ello, mejorar los métodos de enseñanza para que se adecúen a esta situación?. El objetivo del sistema será el de trazar la actuación de un niño con Síndrome de Down ante un problema planteado en modo de juego de ingenio donde fundamentalmente deberá hacer uso de habilidades Lógico-Matemáticas y Espaciales, que ayudando al niño en forma interactiva, lo irá guiando en el aprendizaje de sus reglas. El trazado de la actuación del niño podrá ser analizado posteriormente con otras herramientas de software y dicha información podrá ser de utilidad para los investigadores del área educativa a fin de poder adaptar los métodos de enseñanza centrándose en las fortalezas de los niños con necesidades educativas especiales.

Índice

1 Introducción

Cuando se descubre la atracción que los niños con SD tienen por las computadoras, surgen algunos programas educativos con la finalidad de facilitar su aprendizaje, sin embargo, la mayoría de ellos está destinado a mejorar su mayor déficit, es decir, el lenguaje [Newcomb, 1998]. También existen otros para el apoyo en operaciones matemáticas básicas [González C., 2000]. Asimismo, existe gran cantidad de software educativo y lúdico para toda la población infantil orientados a la resolución de problemas de ingenio (puzzles en general), pero que no guardan registro de las decisiones tomadas por el niño, ni tienen métodos cuidados para asistirlo en su aprendizaje.

Sería de utilidad entonces para los investigadores de procesos de aprendizaje de niños con Síndrome de Down (SD), disponer de un programa que mientras le enseña al pequeño a jugar, permita el registro de cada decisión tomada por el niño durante dicho proceso. Esto posibilitará una revisión posterior que incluye por ejemplo obtener información sobre los aciertos, las clases de errores cometidos, en qué circunstancias, si aprende con las ayudas recibidas. En particular, si el juego es de los tipificados como "juego de ingenio", rompecabezas o puzzles, el niño podrá mejorar sus habilidades lógico-matemáticas y espaciales [Gorriz, B., 2001]. También sería interesante minimizar el uso de recursos lingüísticos, debido a que en esta área se encuentran las mayores dificultades de desarrollo del niño, haciendo que dicha variable no intervenga en el proceso.

El objetivo de este proyecto, está orientado a dar apoyo a la investigación sobre procesos de aprendizaje en niños con SD. Se buscarán aportes de la teoría de Inteligencias múltiples [Gardner, 1996], con el fin de desarrollar un software multimedia que evite el uso de lo que Gardner denomina inteligencia lingüística. Para ello, los sonidos que emita el programa, deberán manifestar los estados de ánimo del instructor (la computadora en este caso) y serán acompañados por imágenes acordes, sin pronunciar palabra alguna. De igual manera, se evitará la inclusión de cualquier vocablo escrito. El niño interactuará con el programa a través del uso del ratón o del teclado, sin esperar la emisión de vocablos. También podrá utilizarse para que en posteriores análisis, el investigador encuentre similitudes o diferencias con otras poblaciones de niños (hipoacúsicos, con diferentes problemas neurológicos, niños sin NEE (Necesidades Educativas Especiales)).

Se considerará exitoso este proyecto, si el investigador que hará uso de la herramienta verifica que un niño testigo esté en condiciones de avanzar en el aprendizaje del juego y disponga del registro de interacciones del niño con el programa.

1.1 Lineamientos del Documento de Tesis

El presente capítulo introduce el problema en cuestión y esboza una solución que será desarrollada en el resto del documento. El mismo, reúne la información sugerida en la metodología Métrica V3 del Ministerio de Administraciones Públicas Español. Se ha buscado la forma adecuada de presentar el trabajo realizado a partir del registro cronológico de procesos, actividades y tareas que la metodología indica.

En el segundo capítulo se encuentra el Estudio de Viabilidad del Sistema, se describe el software y se establece el alcance del mismo, para cerrar el mismo analizándose si el producto puede encuadrarse dentro de lo que se denominan Sistemas Tutoriales Inteligentes.

El tercer capítulo, muestra la Especificación de Requerimientos del Software de acuerdo al estándar IEEE 830 [IEEE 830, 1993].

En el capítulo 4, se observa el Proceso de Gestión del Proyecto en el cual se estima el esfuerzo y se planifican las tareas.

En el siguiente capítulo, se vuelca el registro de los otros procesos de Interfaz que sumados al de Gestión, completan lo sugerido en la metodología y que son: Gestión de Configuración, Calidad y Seguridad.

En el capítulo 6, se encuentra el proceso de Análisis del Sistema y en el siguiente el Diseño del Sistema.

El octavo capítulo está dedicado a la construcción e implantación del sistema.

En el noveno, se encuentra la evaluación de los resultados y los lineamientos a seguir incluyendo posibles mejoras en futuras versiones.

En el Anexo I se encuentra la comparación entre lo planificado y lo realizado, en el Anexo II se presenta la Gestión de Configuración llevada a cabo y en el Anexo III se presentan las unidades y las clases de diseño y en el Anexo IV se observan las referencias bibliográficas y software evaluado o utilizado.

1.2 Estado de la tecnología

La mayoría del software desarrollado para esta población, se orienta a enseñar al niño a hablar, en general, haciendo uso de imágenes, inclusive, animaciones faciales que gesticulan los vocablos. Podemos incluir entre ellos el proyecto Icatiani [Toledo Ma., Kirschning I., 2001], cuyo principal objetivo fue el desarrollo de un software que apoya la adquisición del lenguaje ajustándose a algunas de las necesidades de la escuela de educación especial.

Suele incluirse simultáneamente el texto escrito, a modo de introducción a la lectura, como lo hace el programa SOALE [Cana López, D., 1996], que "es un método de lenguaje - escritura, es decir, no se propone, como único objetivo la enseñanza de las habilidades lectoras, sino que, mediante estrategias adaptadas a los déficits de los niños con SD, se pretenden desarrollar las habilidades lingüísticas de los niños y, más adelante y si es posible, abordar la enseñanza de la lectura.".

Para el apoyo de las matemáticas, podemos encontrar un Sistema Tutorial Inteligente, independiente del dominio particular de enseñanza, para la ayuda de alumnos con necesidades educativas especiales [González C., 2000]. En este caso, orientado a trabajar con dos tipos de NEE: Síndrome de Down y dislexia. "La adaptación del sistema se basa en la secuenciación de los conceptos a enseñar, la dinámica de la presentación, la

interacción del sujeto con el sistema y la presentación de los estilos de aprendizaje según la secuencia de eventos y tipo de tareas presentadas. A través de esto, el sistema se ajusta a las características individuales de los alumnos."

Se ha acotado el alcance al software desarrollado en lengua española. Sin embargo, lo aquí presentado es bastante descriptivo de una población de similares características en lengua inglesa, donde el material disponible es mucho más basto.

Cabe destacar que no se ha encontrado (en idioma alguno) software que registre los procesos de aprendizajes lógico-matemáticos y espaciales, evitando el uso del lenguaje expresivo oral o escrito.

1.3 Identificación del problema

Hace casi 200 años se pensaba que una persona ciega no sería capaz de leer. Sin embargo Lois Braille, quien tenía prácticamente perdida su visión por una infección, creó a partir de un sistema de comunicación militar el lenguaje hoy ampliamente difundido. Nadie piensa ya que una persona ciega no puede leer, aunque utilice otros métodos.

Para el caso de niños con SD, Chapman y Hesketh observan que el fenotipo conductual en los niños con SD se caracteriza por "retraso mental, acompañado de un déficit específico adicional en el desarrollo del lenguaje..." [Chapman, R., Hesketh. L, 2000].

Lo que se pretende a través de este trabajo de tesis es aportar una herramienta que ayude a contestar la siguiente pregunta: ¿existirá un proceso que ayude a una persona con SD a adquirir habilidades lógico-matemáticas y espaciales que no haga uso del lenguaje expresivo oral?

El software en general está orientado a personas sin NEE. Cuando vemos un video-juego de acción, que al finalizar un nivel (que puede ser exitoso o no), muestra al protagonista contento, no deja dudas que el nivel fue superado. ¿Qué significado puede tener para un niño que no sabe leer las palabras GANA o PIERDE al finalizar la etapa?

En la investigación de procesos, la observación de una o más sesiones sería suficiente para la observación de niños sin NEE. Pero si por el contrario, el niño no avanza en las pruebas planteadas, se pueden presentar, entre otros, los siguientes inconvenientes:

- El proceso se dilata en varias sesiones
- El proceso se descontextualiza.
- El estado de ánimo en el momento de la observación es variable.

Por el contrario, si el observador es una computadora, que registra cronológicamente las acciones que el programa lleva a cabo en conjunto con las decisiones que toma el usuario, en principio se puede decir que es una tarea viable. Para ello, bastaría con diseñar adecuadamente la información a guardar. La sesión comienza en el momento que el niño quiere jugar y no depende de la hora, del día. Cuando el niño lo decide, deja de jugar y cuando quiere, vuelve a empezar. Este proceso de aprendizaje puede durar días o meses, dependiendo no sólo de las dificultades en el aprendizaje, sino también con la familiaridad

que tenga con las computadoras. Asimismo, distintos investigadores podrán intercambiar información sobre estudios suministrados con una contextualización similar.

Si el investigador cuenta con una herramienta que mientras guía al niño en el aprendizaje de habilidades evitando el uso del lenguaje (oral o escrito), y mientras el niño aprende, va dejando sus huellas de forma tal que el investigador pueda reconstruir el proceso seguido, el objetivo se habrá alcanzado. A partir de la generación del reporte, los posteriores análisis que hace el investigador con él (comparaciones entre pares dentro de una población o con otras poblaciones en forma individual o en conjunto, estadísticas, etc.) están fuera del alcance de este proyecto.

1.4 Esbozo de la solución

Siguiendo la línea de Gardner, recogemos recomendaciones acerca de cómo mejorar las habilidades lógico-matemáticas, fortaleciendo las acciones de "Experimentar, preguntar, resolver rompecabezas lógicos, calcular, etc." [Gorriz, B., 2001]

Tomando como premisa que el juego es un facilitador del aprendizaje (según Piaget y Vigotzky entre tantos), el niño deberá ver al programa de esa manera, como un juego. Si se administra este programa a un niño, será porque un conocimiento básico de uso de computadora tiene, en general, y dicho conocimiento seguramente será a través de un juego. Es un objetivo que el niño no note diferencias entre este programa y un juego de computadora, quedando las tareas de gestión del reporte lejos del alcance del entorno que pueda manejar el niño.

El juego deberá ser multimedia. El niño irá aprendiendo habilidades que le permitan ir incorporando los conocimientos necesarios para demostrar que ha aprendido las reglas del juego. Para ello, el programa irá variando los tiempos de espera (tiemeout) para mostrar la ayuda al movimiento que debe realizar el niño. Ese tiempo dependerá de la situación en que se encuentre el desarrollo de la partida, de partidas anteriores y de lo adecuado o no que haya sido el movimiento similar previo.

Para gestionar el proyecto se usará como apoyo la metodología Métrica V3 [Métrica V3, 2000] del Ministerio de Administraciones Públicas Español.

Para la implementación se planea el desarrollo en entorno Windows (sin resignar futura portabilidad a Linux - fuera del alcance del proyecto).

El entorno de desarrollo deberá soportar librerías 3D, que dada la portabilidad esperada, serán orientadas al uso de OpenGL (www.opengl.org). Se estima a priori que el lenguaje de programación puede ser Delphi, sin descartar otros.

Se utilizará el paradigma de orientación a objetos para el desarrollo del software. Sin embargo, el desarrollo de software para animación 3D, tiene sus particularidades, en especial, en lo que respecta precisamente a la animación de las escenas que habrá que considerar tanto en el diseño como en el desarrollo.

Los objetos 3D, estáticamente se comportan como cualquier otro objeto en programación clásica, sin embargo, dinámicamente, no alcanza con indicarle por ejemplo a un cubo que

se desplace, sino, habrá un evento (temporal) que al dispararse le indicará que llegó el momento de reubicarse. Cada escena que se ve en pantalla (pueden ser del orden de 220 imágenes por segundo – fps de Frames per Second), mostrará a los objetos en su posición actual, para lo cual previamente, cada objeto debió tomar su posición correcta. En este sentido, OpenGL cuenta con "Listas de Despliegue, que son una manera de almacenar comandos de dibujo en una lista para un trazado posterior" [García Trujillo, C, 2001].

Un entorno gráfico resulta agradable si sus objetos tienen realismo. Para ello, los objetos 3D deberán diseñarse, no solo para cumplir con colores y dimensiones, sino, también habrá que considerar sus texturas y movimientos, como así también la iluminación de la escena y la posición de las cámaras.

1.5 Metodología Métrica V3

Los procesos de la estructura principal de MÉTRICA Versión 3 son los siguientes:

- Planificación de Sistemas de Información.
- Desarrollo de Sistemas de Información.
- Mantenimiento de Sistemas de Información.

En el presente trabajo el mayor hincapié se hará en las tareas que conforman al segundo de ellos, debido a las características del desarrollo independiente y de carácter académico.

El proceso de desarrollo se subdivide en cinco procesos, a saber:

- Estudio de Viabilidad del Sistema (EVS).
- Análisis del Sistema de Información (ASI).
- Diseño del Sistema de Información (DSI).
- Construcción del Sistema de Información (CSI).
- Implantación y Aceptación del Sistema (IAS).

La metodología cuenta a su vez con cuatro interfaces a saber:

- Gestión de Proyectos (GP)
- Seguridad (SEG)
- Gestión de la Configuración (GC)
- Aseguramiento de Calidad (GC)

Cada tarea, de cada actividad, de cada proceso, es a su vez un proceso al cual se lo identificará por sus entradas y sus salidas irán conformando el presente documento.

2 Estudio De Viabilidad Del Sistema

En este apartado, se presentarán las actividades correspondientes al Estudio de Viabilidad del sistema. También se hará referencia a las actividades realizadas correspondientes a las interfaces que define la metodología Métrica versión 3 que se sincronizaron con las plasmadas aquí. Dadas las características del sistema a desarrollar, que no será implementado para integrarse en una organización, sino, que será para uso de profesionales investigadores en forma individual, el estudio de viabilidad se orienta a establecer los alcances del sistema (se verá en este capítulo) y a formalizar los requerimientos (en el capítulo 3).

Dice la metodología: "Si la justificación económica es obvia, el riesgo técnico bajo, se esperan pocos problemas legales y no existe ninguna alternativa razonable, no es necesario profundizar en el estudio de viabilidad del sistema, analizando posibles alternativas y realizando una valoración y evaluación de las mismas, sino que éste se orientará a la especificación de requisitos, descripción del nuevo sistema y planificación".

En nuestro caso, se cumplen las condiciones que hacen innecesaria la evaluación de alternativas de solución.

2.1 Establecimiento del Alcance del Sistema

Durante la presente actividad, se acotará el alcance de la necesidad planteada, se determinarán los objetivos y se iniciará el estudio de los requisitos.

Se analizan asimismo las restricciones de carácter general o particular que pudieran condicionar el estudio y la planificación de las alternativas de solución que se propongan.

La metodología Métrica versión 3 nos indica la realización una serie de tareas dentro del Estudio de Viabilidad del Sistema:

- EVS 1.1 Estudio de la Solicitud
- EVS 1.2 Identificación del Alcance del Sistema
- EVS 1.3 Especificación del Alcance del Sistema

2.1.1 Estudio de la Solicitud

2.1.1.1 Descripción General del Sistema

En la Introducción (Capítulo 1) se ha descrito el problema planteado. Ahora se presenta una descripción más general explorando los distintos aspectos del sistema a desarrollar para dar solución al problema planteado.

Desde un punto de vista general, el sistema se puede describir a partir de dos visiones distintas:

- Visión del investigador
- Visión del usuario

La primera de ellas, es la del investigador que desea obtener el trazado de las acciones llevadas a cabo por el usuario y las respuestas llevadas a cabo por el programa tendientes a asistir al usuario en el aprendizaje de las reglas del juego.

La segunda de ellas, es la de un niño al que se le quiere enseñar a jugar un juego.

Dadas las características de los niños a quienes estará dirigida la investigación, la interacción del sistema con el niño deberá ser esencialmente visual. Los sonidos de apoyo estarán orientados a expresar sensaciones y no vocablos.

El niño a medida que va jugando, podrá ir acrecentando sus conocimientos sobre el tema. La idea es que el niño pueda "descubrir" cada paso que tiene que dar y el sistema esté atento a ayudarlo para que pueda avanzar en casos de error o de inacción prolongada.

El sistema deberá ser atractivo para el niño, dentro de un entorno tridimensional, tendiendo a incentivar al niño a querer aprender a jugar y que el aprender a jugar sea entretenido.

El juego a enseñar, deberá cumplir algunos requisitos que se detallan a continuación:

- Tendrá un reglamento.
- Hará uso de asociación de colores y formas.
- Se deberá poder jugar tanto en la computadora como fuera de ella, de forma tal que el investigador disponga de un método de comprobación de aprendizaje a su disposición.
- Deberá ser competitivo (al menos entre dos jugadores) de forma tal de enriquecer la participación del niño con la presencia de un oponente. De esta forma, ampliando el punto anterior, el investigador podrá evaluar el aprendizaje del niño "jugando" una partida.
- Tendrá un resultado final que puede ser el triunfo de cualquiera de los 2 participantes o empate, sin embargo, el resultado no es un parámetro a evaluar, dado que el objetivo es el aprendizaje de las reglas (táctica) y no el ganar las partidas (más ligado a las estrategias).

Hasta este punto se ha limitado el alcance del desarrollo, presentándose a continuación aquello que expresamente se estipula fuera de los límites concretos del desarrollo:

- El sistema no se utilizará para el análisis de los resultados. El trazado podrá exportarse a archivos, que el investigador podrá analizar haciendo uso de planillas de Cálculos, bases de datos u otros sistemas de utilidad. Para ello podrá aplicar técnicas estadísticas o data mining.

- No se busca la generalización para su aplicación a otros juegos con reglamentos distintos.

La comprobación de la utilidad del sistema dependerá de la aprobación del investigador quien basará su decisión en experiencias que demuestren que un niño puede aprender a jugar (respetando las reglas) simplemente usando el sistema, sin ninguna indicación o ayuda externa.

2.1.1.2 Catálogo de Objetivos del Estudio de Viabilidad del Sistema

El presente proyecto no tiene antecedentes que permitan analizar la situación actual, por lo tanto, objetivos del proceso EVS serán:

- Definir los requisitos del sistema.
- Analizar las alternativas de solución, valorarlas y seleccionar la más adecuada.

En relación con las interfaces que se llevan a cabo en forma simultánea a este proceso:

- Definir los requisitos de Gestión de Configuración y establecer un plan.
- Identificar las propiedades de Calidad para el sistema.
- Estimar el esfuerzo y planificar el sistema.

Cabe aclarar que, como análisis de la situación actual, se incluirá un apartado a modo de introducción a los Sistemas Tutoriales Inteligentes (STI), con el objetivo de identificar si el presente proyecto se encuadra en un desarrollo de este tipo de sistemas.

2.1.1.3 Catálogo de Requisitos del sistema

Se presentan en el capítulo 3: Especificación de Requerimientos del Software.

2.1.2 Identificación del Alcance del sistema

En esta tarea, se busca restringir el alcance del sistema a la empresa donde será implementado y adecuar en tiempo su desarrollo respecto de los demás proyectos en marcha. En este caso, al ser un proyecto aislado, no modificará ninguna planificación prevista. El resultado podrá observarse en:

- Descripción General del Sistema (Ítem 2.1.1.1)
- Catálogo de Requisitos (Capítulo 3)
- Catálogo de Usuarios (Ítem 3.2.3)

2.1.3 Especificación del Alcance del EVS

En este apartado se definen los objetivos del estudio de la situación actual que darán origen a la siguiente información:

- Catálogo de Objetivos del EVS (Ítem 2.1.1.)
- Catálogo de Usuarios (Ítem 3.2.3)
- Plan de Trabajo (Ítem 4.2)

2.1.3.1 Estudio de la Situación Actual

A los objetivos planteados previamente en el Catálogo de Objetivos del EVS (Ítem 2.1.1.2), se agrega el objetivo del estudio de la situación actual, a saber:

- Analizar si el proyecto puede "enmarcarse" dentro de lo que se denominan Sistemas Tutoriales Inteligentes.

Este análisis permitirá definir el tipo de proyecto entre un desarrollo de software tradicional yo un Sistema Tutorial Inteligente(STI), condicionando así la metodología a emplear.

El sistema a desarrollar está orientado principalmente a estimular y reconocer inteligencias espaciales y lógico-matemáticas. Para ello, intentará que el usuario aprenda las reglas que impone el juego. Los STI tienen también como objetivo la enseñanza y la recolección de datos sobre el camino que recorre el alumno en el aprendizaje y es por estas similitudes que se incorpora el presente análisis.

Los Sistemas Tutoriales Inteligentes están dotados de:

- Un módulo del dominio: que define el dominio del conocimiento (conocimiento sobre qué enseñar). Suele denominarse también módulo experto.
- Un módulo del estudiante: capaz de definir el conocimiento del estudiante en cada punto durante la sesión de trabajo (conocimiento sobre a quién enseñar).
- Un módulo del tutor: que genera las interacciones de aprendizaje (estrategias) basadas en las diferencias de lo que espera como respuesta adecuada el módulo experto y las interacciones producidas por el estudiante (conocimiento sobre cómo enseñar).
- Un módulo de la interfaz con el usuario: que permite la interacción del estudiante con un STI de una manera eficiente (conocimiento sobre cómo presentar el material).

A continuación podemos observar las distintas categorías de Sistemas Tutoriales Inteligentes, incluyendo principales características descriptivas de cada una de ellas:

- Orientado por el instructor:
  - El instructor controla el paso / ritmo.
  - Se presenta la información directamente.
  - Se presentan problemas para estimular la fijación de conocimientos
  - Puede incluir restricciones de tiempo
  - Proporciona feedback inmediatamente.
- Descubrimiento Guiado:
  - El instructor controla el paso / ritmo.
  - Se presenta la información directamente.
  - No existe límite de tiempo

        - Las respuestas del sistema van desapareciendo con el tiempo.

- Usuarios avanzados
    - Los usuarios pueden crear sus propios problemas y ejercicios
    - Los estudiantes determinan el paso / ritmo
    - Los estudiantes pueden irse en cualquier momento

- Exploratorio
    - Provee caminos alternativos de información
    - El estudiante controla el paso / ritmo
    - Ningún feedback
    - Los alumnos pueden hacer sus propios problemas
    - Los estudiantes son responsables del aprendizaje del material

De las cuatro categorías, la que mejor encuardra al sistema a desarrollar es la de **descubrimiento guiado**. Esta aproximación da énfasis a la "producción" de nuevo conocimiento y no a la "reproducción" del mismo. Invita al aprendiz a pensar, ir más allá de la información dada y entonces descubrir la información correcta [EPITOME project, 1996]. Lo esencial de este estilo es una particular relación maestro-estudiante, en la cual, las secuencias de información del maestro y las preguntas, causan un juego de respuestas por parte del aprendiz. La combinación de información y pregunta hecha por el maestro "despierta" una contestación correcta que es descubierta por el aprendiz. El efecto acumulativo de este proceso converge, llevando al estudiante a descubrir el concepto o principio buscado.

El descubrimiento guiado produce mejores resultados en una situación uno-a-uno. En un grupo (ej., aula) muchos aprendices pueden beneficiarse de este proceso, pero los descubrimientos se producen a velocidades diferentes. Si cuando uno de ellos ha descubierto la respuesta, la transmite en público, los demás perderán parte importante del proceso. Por consiguiente, este estilo no es bastante apropiado llevar a cabo en el aula.

Los resultados de aprendizaje que pueden alcanzarse en este estilo son la resolución de problemas definidos y con conceptos concretos. Aprender en el estilo del descubrimiento guiado, induce a un aprendiz a desarrollar habilidades más cognoscitivas que el estilo orientado por el instructor [Mosston y Ashworth, 1990]. El desarrollo cognoscitivo es beneficioso para el desarrollo de habilidades en la resolución de problemas. Los conceptos definidos y concretos también pueden lograrse presentando la información indirectamente, por ej. en una pregunta al aprendiz sobre el punto en cuestión.

Los tipos de aprendizajes más apropiados son los que intentan enseñar el principio y el concepto del tema en cuestión. El objetivo a descubrir no debe ser un hecho, una palabra específica o término técnico. Estas categorías de conocimiento son sabias diciéndose, directamente. La estructura del conocimiento en este estilo es "de abajo hacia arriba (bottom-up)". El aprendiz descubrirá el tema más específico inicialmente y luego, gradualmente va subiendo a niveles más altos de generalización.

Dada esta introducción, se tomará como base para hacer una adaptación al sistema en cuestión:

- Módulo del dominio: Se asocia al módulo que conoce y hace cumplir las reglas del juego de ingenio.
- Módulo del estudiante: Se asocia al módulo que va controlando la partida, que sabe cual fue el desempeño del usuario hasta el momento.
- Módulo del tutor: Se asocia al módulo que genera en tiempo y forma la ayuda o corrección a partir de los movimientos efectuados por el usuario.
- Un módulo de la interfaz con el usuario: Se asocia al módulo que controla el entorno visual mencionado (conocimiento sobre cómo presentar el material).

Como vemos, el proyecto podría encuadrarse como un ITS sin embargo, carece de lo que se ha encontrado en común a todas las categorías de ITS que son las preguntas y respuestas. Por ello, el desarrollo del sistema, no se realizará siguiendo los lineamientos de los ITS. A pesar de ello, la huella dejada por esta investigación podrá observarse a lo largo de los sucesivos procesos encarados, en particular, en los de análisis y diseño.

3 Especificación de Requerimientos del Sistema

A continuación se presentarán la especificación de requerimientos del software de acuerdo al estándar IEEE 830 [IEEE 830, 1993].

3.1 Introducción

3.1.1 Objetivos

El objetivo del sistema consiste en *trazar la actuación* de un niño con Síndrome de Down *ante un problema planteado* en modo de juego de ingenio, de forma tal que *ayudando* al niño en forma *interactiva*, lo irá *guiando* en el aprendizaje de sus reglas. El detalle de la definición de estos objetivos se observa en los Ítems 3.2 y 3.3 en forma de requisitos.

3.1.2 Alcance

El desarrollo del sistema está limitado a lo indicado en sus objetivos. En particular, el sistema no se utilizará para el análisis de los datos registrados.

Tampoco se considerará dentro de los alcances del sistema, la enseñanza del uso de la computadora a los usuarios.

Por último cabe indicar que el sistema busca enseñar estrategias para el aprendizaje de las reglas del juego y no estrategias para ganar partidas.

3.1.3 Definiciones, sinónimos y abreviaturas

3.1.3.1 Definiciones

Color de Piezas: cuando se haga referencias al color de las piezas, se estará refiriendo al color blanco o negro correspondiente al asignado al usuario o al oponente virtual. No es una referencia a los colores de los lados de cada pieza.

Juego de Ingenio: se utiliza para referenciar a aquellos juegos que ejercitan preponderantemente la inteligencia espacial, haciendo uso de figuras. Los "puzzles" o "rompecabezas", representan adecuadamente a esta categoría de juegos.

3.1.3.2 Sinónimos

Investigador: Es la persona que analizará los datos obtenidos en el trazado de las sesiones de juego por parte del niño con Síndrome de Down (SD). Desde el punto de vista de sistemas, el investigador también será un usuario, pero en este documento se evitará referenciarlo así para evitar confusiones.

Rotación: Se entiende por rotación al giro de 90º que puede realizar una pieza. Se entiende por giro o rotación hacia a la izquierda, cuando sea en sentido contrario a las agujas del reloj, mirando la pieza desde arriba. Si es hacia la derecha, será en el mismo sentido de las agujas del reloj.

Usuario: Cuando se haga referencia al estudiante, aprendiz o niño, se estará haciendo referencia al usuario del juego de ingenio.

3.1.3.3 Abreviaturas

3D: Tridimensional.

SD: Síndrome de Down

STI: Sistema tutorial Inteligente

3.2 Descripción General

3.2.1 Perspectiva del producto

El producto está orientado a dar una solución a investigadores de la educación a niños con SD.

3.2.2 Funciones del producto

El producto tiene dos funciones principales a satisfacer:

- **Trazar Partida**: los movimientos del usuario y las ayudas que brinda el sistema deberán almacenarse cronológicamente.
- **Jugar Partida**: el usuario deberá poder iniciar una partida, realizar los movimientos de las piezas y recibir ayuda de ser necesario en el momento previsto. El sistema realizará las jugadas correspondientes al oponente del usuario.

3.2.3 Características de los usuarios

Como ya se ha mencionado, existen dos usuarios: el investigador y el niño.

El **investigador** se caracteriza por tener conocimientos sobre los procesos de aprendizaje de personas con necesidades educativas especiales, en particular, de la población (niños con SD) a la que va destinado el sistema. Existen investigadores que consideran necesario estudiar y tener en cuenta las diferentes patologías específicas (o en este caso, condición genética), para mejorar luego las intervenciones psicoeducativas [Troncoso, 1998].

El **niño** con SD se caracteriza por tener entre otras, dificultades en el desarrollo del lenguaje. Por ello, es que se intenta evitar su uso a fin de analizar su desempeño ante estímulos esencialmente visuales. Los investigadores consideran que los niños tienen un mejor manejo de su memoria visual que la auditiva [García Escamilla, 1983].

3.2.4 Restricciones

El sistema debe adaptarse a diferentes capacidades de aprendizaje, sin esto significar que el investigador deba realizar una tarea que implique "horas" readaptando la configuración del software.

3.2.5 Suposiciones y dependencias

Suponemos que los usuarios están familiarizados con el uso de computadoras, siendo necesario que manejen adecuadamente el ratón (mouse).

3.3 Requisitos específicos

3.3.1 Interfaces externas

No existen interfaces con otros sistemas. Sin embargo, el formato del registro de la partida deberá ser adecuado para utilizar desde planillas de cálculo o sencillo para importar desde bases de datos.

3.3.2 Funciones

A partir de la segunda entrevista, se convino en determinar las Reglas del Juego que se transcriben a continuación. Luego se describirán las funciones.

3.3.2.1 Reglas del Juego

Jugadores:

Es un juego para dos jugadores (Blanco y Negro).

Tablero:

El juego consta de un tablero dividido en 3 zonas:

- Zona de juego
- Zona del jugador Blanco
- Zona del jugador Negro

La zona de juego consta de 24 casilleros donde pueden ser colocadas las piezas.

Los casilleros perimetrales externos tienen sus bordes exteriores asignados con uno de los tres colores de lados de pieza posibles. De igual forma, los casilleros del perímetro interno también tienen asignados colores a sus bordes interiores.

La siguiente figura (Ilustración 3-1) muestra esquemáticamente el tablero de juego.

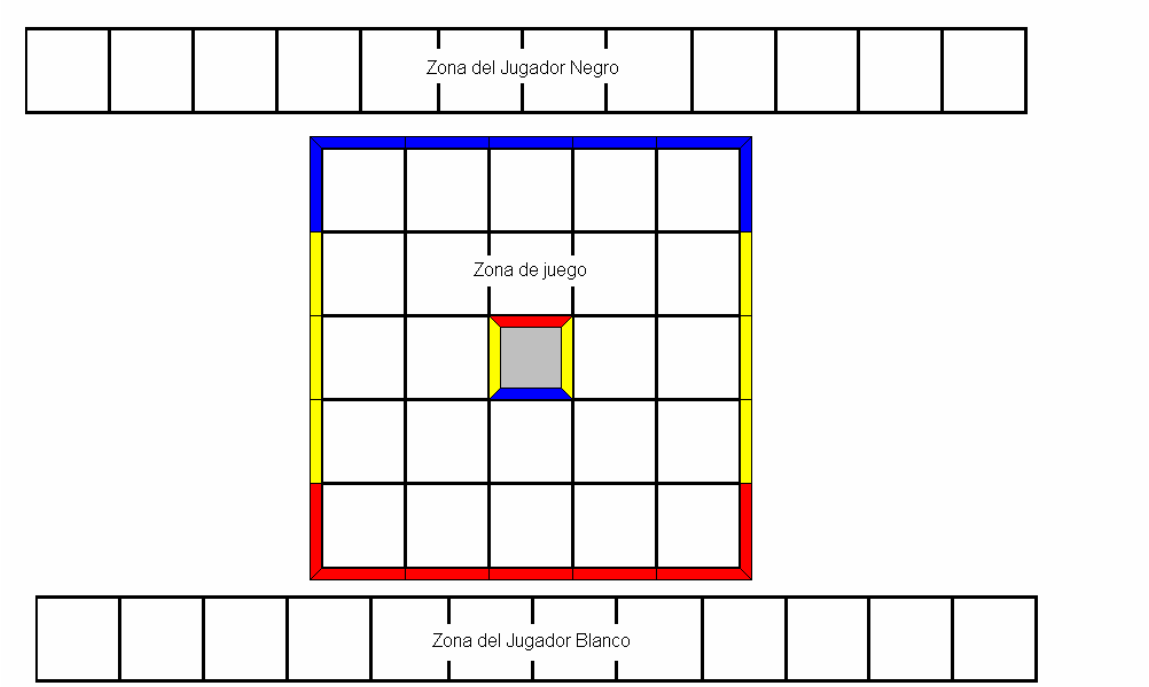


Ilustración 3-1 Distribución del Tablero

Piezas (Ilustración 2):

- Cada jugador tiene asignadas 12 piezas que al iniciar la partida, las cuales deberán colocarse en la posición inicial en su respectiva zona.
- Cada pieza de forma cuadrada es única independientemente del Jugador.
- Cada lado de la pieza tiene un color asignado como se observa a continuación.

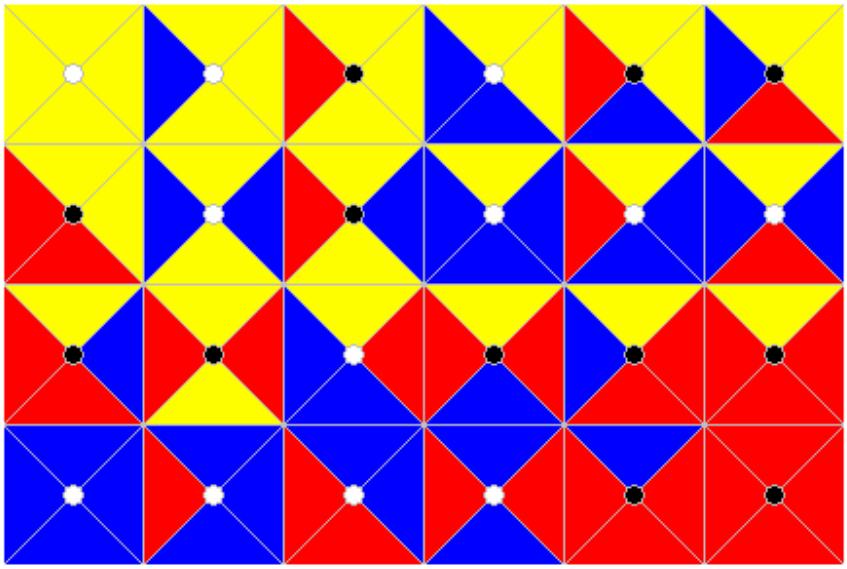


Ilustración 3-2 Piezas que componen el juego

En el círculo central de cada pieza, está indicado según su color, el jugador al cual pertenece.

Regla básica elemental:

Las piezas pueden ocupar un casillero únicamente en el caso que los colores de sus lados coincidan con los colores que rodean al casillero. Dichos colores están fijados por los bordes perimetrales o por otras piezas previamente colocadas en casilleros vecinos. Si alguno de los bordes del casillero elegido no es perimetral o no tiene una pieza contigua colocada, cualquiera de los tres colores es apropiado para ser colocado en dicha posición.

Jugar una pieza:

Jugar una pieza implica: elegirla, elegir el casillero, rotarla de ser necesario y luego aceptar la pieza jugada en el casillero elegido, siempre respetando la regla básica elemental. Una vez jugada la pieza, no hay posibilidad alguna de moverla.

Objetivo del juego:

El objetivo del juego para cada jugador será el de colocar más piezas en el tablero que su oponente.

Desarrollo del juego:

Comienza la partida el jugador Blanco, jugando una pieza.

Luego lo hará el jugador Negro y así siguiendo alternadamente hasta que alguno de ellos (o ambos) no puedan jugar más.

Final de la Partida y Resultado:

Si a su turno el jugador Negro no puede jugar más piezas y el Blanco aun sí, el jugador Blanco juega una pieza y es el ganador.

Si a su turno el jugador Blanco no puede jugar más piezas y el Negro aun sí, el jugador Negro juega una pieza y es el ganador.

Si a su turno el jugador Blanco no puede jugar más piezas y el Negro tampoco, el resultado es empate.

3.3.2.2 Requisitos funcionales de "Trazar Partida"

El sistema deberá almacenar la información que permita reconstruir la interacción con el niño. En otras palabras, sería la información necesaria para que un simulador pueda recrear la partida sin perder información (el simulador puede ser una mejora a futuro que no está incluida en el presente desarrollo). Para ello deberá registrar los horarios de ocurrencia y la situación del juego en los siguientes eventos:

- Inicio de la partida.
- Selección de Pieza.
- Selección de Casillero.
- Indicación de Rotación (Incluyendo el sentido de giro).
- Aceptación de Pieza en Casillero.
- Invocaciones erróneas de objetos (piezas, casilleros, indicadores de rotación), causados por infracción a las reglas del juego o por carecer de sentido en el contexto del juego (por ejemplo, elegir una pieza del oponente o una pieza ya jugada).
- Ayudas del sistema en cada oportunidad que ocurra haciendo referencia al tipo de ayuda efectuada.
- Resultado.

3.3.2.3 Requisitos funcionales de "Jugar Partida"

- Iniciar Partida: Se disponen las piezas en el tablero en su posición inicial.

- Asignación del Jugador: Se asigna al usuario un jugador y se orientará el tablero de forma tal que la zona del usuario quede en la parte inferior de la pantalla.

- Seleccionar Pieza (1): El jugador indicará la pieza elegida para jugar y el sistema responderá mostrando que la pieza ha sido elegida en una posición superior a la que tenía originalmente.

- Seleccionar Casillero (2): El jugador indicará el casillero elegido en el cual desea jugar la pieza elegida y el sistema, si fuera posible jugar la pieza en dicho casillero, desplazará la pieza desde su posición de Pieza Seleccionada hasta colocarse encima del casillero elegido, manteniendo la elevación respecto del resto de las piezas. En el caso que no sea posible, indicará esto con un mensaje de error.

- Rotar Pieza (3): El jugador indicará el sentido de rotación de la pieza y el sistema responderá rotando la pieza 90 grados en el sentido indicado, manteniendo la elevación respecto del resto de las piezas.

- Aceptar Pieza (4): El jugador indicará la pieza ya colocada sobre el casillero elegido y en una orientación tal que cumple con la regla básica elemental. El sistema responderá bajando la pieza hasta la altura del resto de las piezas, ocupando el casillero indicado.

- Respetar secuencia: (1), (2), (3) y (4) son movimientos que deben realizarse secuencialmente. El paso (3) no será necesario en el caso en que la pieza pueda aceptarse sin necesidad de rotarla.

- Cancelar selección de Pieza elegida: El usuario podrá seleccionar la pieza elegida indicando algún objeto de la zona de piezas aun no jugadas.

- Cambiar selección de casillero: El usuario podrá cambiar la selección de casillero por otro válido.

- Jugar en lugar del oponente: Una vez que el usuario aceptó una pieza en un casillero respetando la regla básica elemental, el sistema responderá jugando una pieza del oponente, en caso de poder hacerlo.

- Detectar condiciones de final de juego: El sistema detectará dicha situación de acuerdo a las reglas y actuará en consecuencia según el caso.

- Ayudas: Cuando el sistema se encuentre en espera de alguno de los movimientos a realizar por el usuario, y se cumpla el tiempo de espera asignado a dicha situación, presentará una ayuda visual indicando cual podría ser una acción adecuada. Se acompañará con una señal sonora. El usuario podrá tomar la ayuda o desecharla, sin que esto implique error alguno. Por ejemplo, el sistema puede sugerir elegir una pieza, si el usuario elige otra que puede jugar, está respetando las reglas, por lo tanto, ello no es un error. El investigador, podrá tomar en cuenta esta situación en el análisis de la partida e interpretarla adecuadamente.

- Tiempos de espera para mostrar ayudas: Este tiempo deberá ser tal de ir alargándose en el caso que el usuario haya demostrado haber aprendido las reglas y acortándose en caso contrario.

3.3.3 Requerimientos de "performance"

Dado el requerimiento de uso de tecnología 3D, el sistema deberá comportarse adecuadamente y de igual manera, en distintas computadoras independientemente de las capacidades de las mismas. Deberá establecerse el equipo mínimo para su uso.

3.3.4 Requerimientos de bases de datos lógicas

No existe requerimiento alguno, sin embargo, la información almacenada, deberá tener una estructura adecuada para que su importación a bases de datos o planillas de cálculo sea sencilla.

3.3.5 Restricciones de diseño

La interfaz del usuario no debe hacer uso de lenguaje expresivo (ni escrito, ni oral).

3.3.6 Atributos del sistema software

- El entorno debe ser tridimensional.
- Las ayudas deben ser claras y específicas (esto significa que, por ejemplo, si el usuario debe seleccionar un casillero, la ayuda será tal que indique un casillero que esté libre y pueda jugarse la pieza previamente seleccionada).

4 Gestión del Proyecto

Dadas las características de este desarrollo unipersonal, los objetivos de la gestión del proyecto estarán limitados a la planificación, seguimiento y control de las actividades.

4.1 Estimación de Esfuerzo

Dentro de las actividades de inicio del proyecto, se encuentra la estimación de esfuerzo, cuyo objetivo es el de conocer el tamaño que tendrá el software, establecer el costo (no aplicable a este caso), la duración y los recursos necesarios.

4.1.1 Identificación de elementos a desarrollar

4.1.1.1 Catálogo de Clases Principales (Clases Claves)

A continuación se identifican las clases principales mencionadas en el capítulo anterior a fin de llevar a cabo la estimación de esfuerzo. En procesos posteriores se realizará una identificación más exhaustiva y detallada.

Clases de los elementos del juego:

- Jugador
- Tablero de juego
- Pieza
- Casillero de juego

Vale la pena aclarar que las clases referidas deberán implementarse a nivel conceptual y a nivel gráfico.

Clases para controlar el juego:

- Controlador de Estados

Clases del trazado:

- Registro del sistema

4.1.2 Cálculo del Esfuerzo

De acuerdo a lo sugerido por la metodología Métrica V3, se va a utilizar la técnica de Staffing Size. Staffing Size es un conjunto de métricas para estimar el número de personas necesarias en un desarrollo Orientado a Objetos, y para determinar el tiempo de su participación en el mismo.

1) Número de clases clave

Las clases clave representan el dominio del negocio a desarrollar y son las que se definen en las etapas iniciales del análisis. Estas fueron definidas en el ítem 5.1.1.

De acuerdo a lo definido obtenemos que:

- Clases de los elementos del juego (conceptuales y visuales): 4
- Clases para controlar el juego: 1
- Clases del trazado: 1
- **Total: 6 clases clave**

2) Número de clases secundarias

Las técnicas de Métrica 3 expresan: "El número de clases secundarias suele variar de una a tres veces el número de clases clave. El intervalo depende principalmente del tipo de clases de usuario. Las interfaces gráficas de usuario incrementan en dos veces el número de clases en la aplicación final. Las aplicaciones sin interfaces de usuario se incrementan en una vez el número de clases, es decir, en una aplicación con unas 100 clases clave y con interfaces gráficas de usuarios, una estimación previa podría apuntar a unas 300 clases para la aplicación final.". De acuerdo a esto y a lo indicado en el punto 1), consideraremos a las Clases de los elementos del juego como gráficas de usuarios y al resto como que no lo son, por lo tanto, la estimación de clases secundarias es:

- Clases de los elementos del juego (conceptuales y visuales): 4 * 2 = 8
- Clases para controlar el juego: 1 * 1 = 1
- Clases del trazado: 1 * 1 = 1
- **Total: 10 clases secundarias**

3) Número medio de personas por día y por clase

La metodología sugiere utilizar de diez a quince días para una clase en producción, es decir, incluyendo la documentación y pruebas de las clases. Esto da como resultado que para un prototipo, la duración será de seis a ocho días, incluyendo código para las pruebas unitarias, pero sin tener en cuenta las pruebas de integración y las pruebas formales de casos.

Dado el tamaño reducido del software a desarrollar, y al ser una tesis de magíster de desarrollo unipersonal, tomaremos como referencias los valores que se sugieren para prototipos.

El rango de seis a ocho días se debe a las variaciones que tienen los proyectos y que analizamos a continuación, indicando sí en este proyecto estamos en una situación favorable, indiferente o desfavorable.

- Las clases de interfaz versus resto de clases del modelo: Las clases de interfaz de usuario suelen tener muchos más métodos y son menos estables en memoria que las propias del modelo de clases.

  Análisis: Desfavorable, ya que gran parte de las clases son de interfaz.

- Clases abstractas versus clases concretas: El sobre esfuerzo necesario para desarrollar una clase abstracta, se puede compensar con el que precisa el desarrollo de una clase concreta.

  Análisis: Normal, ya que no existen elementos de juicio para indicar otra cosa al momento

- Clases clave versus clases de soporte: Las clases clave generalmente conllevan un tiempo superior de desarrollo, porque son las clases que representan las características principales del dominio del negocio.

  Análisis: Desfavorable, ya que las clases involucradas son las del negocio.

- Clases avanzadas versus a clases sencillas: La utilización de clases más complejas como los patrones y los marcos hace que el modelo sea mucho más efectivo, aunque el desarrollo de este tipo de clases requiere un mayor esfuerzo.

  Análisis: Normal, ya que podemos estimar que se está en una posición intermedia.

- Clases "maduras" versus "inmaduras": Las clases maduras, aquellas que su funcionamiento y utilidad ha sido ampliamente comprobado porque se han utilizado durante un periodo de tiempo suficiente, suelen tener más métodos pero requieren menos tiempo de desarrollo, porque únicamente habrá que realizar algún desarrollo adicional sobre las ya existentes.

  Análisis: Desfavorable, ya que ninguna clase ha sido desarrollada al momento.

- Profundidad de herencia en la jerarquía de clases: Las clases más anidadas, es decir con una profundidad mayor en la jerarquía, suponen menos esfuerzo de desarrollo ya que suelen ser una especialización de superclases, y generalmente tienen menos métodos.

  Análisis: Favorable, ya que no se estima una profundidad de clases elevada.

- Ámbito de programación: Depuradores de código integrados, visores de jerarquía de clases, compiladores incrementales y otro tipo de herramientas pueden facilitar y acelerar el desarrollo.

  Análisis: Favorable, ya que se dispone de herramientas adecuadas.

- Librerías de clase: El número, el tipo y la madurez de las clases disponibles para reutilizar, pueden afectar a los niveles de productividad.

  Análisis: Favorable, ya que s e dispone de librerías adecuadas.

De acuerdo al análisis realizado se puede aplicar una simple asociación para cada caso para luego promediarlas, para ello, se utilizará la siguiente nomenclatura:

- F: Favorable = Mínimo de días por clase sugeridos = 6 días/clase
- N: Normal o indiferente = Promedio de días por clase sugeridos = 7 días/clase
- D: Desfavorable = Máximo de días por clase sugeridos = 8 días/clase

En nuestro caso tenemos: D – N – D – N – D – F – F – F (total 8 ítems).

Entonces, los días/clase serán: Días/clase = (8 + 7 + 8 + 7 + 8 + 6 + 6 + 6) / 8 = **7 días/clase**

Esto significa que el esfuerzo necesario para concluir este proyecto será de

Esfuerzo = (Clases Clave + Clases Secundarias) * días/clase

= (6 + 10) * 7 = **112 días**

Dado que el proyecto tendrá asignada una sola persona, y la jornada la consideramos de 8 Horas Hombre por día, tendremos:

Esfuerzo = 112 días * 8 HH/día = 896 HH

Esfuerzo = 896 HH

Duración del proyecto:

La duración del proyecto estará asociada a la disponibilidad de carga horaria del tesista que se estima en unas 17 hs. semanales.

Dado que la planificación se volcará a un documento de Microsoft Project, tendremos que convertir las horas en días para adaptarlo así al caso particular de dedicación parcial que nos atañe. Para ello, dividiremos las horas semanales asignadas por 5 (días hábiles) a considerar: Horas por día laborable = Horas Semanales / 5 = 17 / 5 = 3,4 HH/día.

Pasamos ahora el esfuerzo a días de trabajo calculado: Días de Trabajo calculados = Esfuerzo / Horas por Día laborable = 896 / 3,4 = 263,5 días

Por último, se obtiene la duración total del proyecto: Días Calendario = 263,5 * 7 / 5 = 368,9 días.

Por lo visto, el proyecto tiene una duración estimada en un año aproximadamente, dato que a priori parece razonable.

A continuación se adjunta la planificación del proyecto de acuerdo a la estimación realizada:

4.2 Planificación

En este apartado se realizará la planificación del proyecto. Para ello, se tendrán en cuenta los procesos, actividades y tareas que plantea la metodología y son de aplicación al presente proyecto. A continuación se presenta la exportación del MS-Project:

Fecha de comienzo del proyecto: mar 01/03/05
Fecha de fin del proyecto: mié 01/03/06

Procesos, Actividades y Tareas	Duración	Comienzo	Fin
EVS Estudio de Viabilidad del Sistema	13 días	01/03/05	17/03/05
EVS-1 Establecimiento del Alcance del Sistema	9 días	01/03/05	11/03/05
EVS-1.1 Estudio de la Solicitud	5 días	01/03/05	07/03/05
EVS-1.2 Identificación del Alcance del sistema	3 días	08/03/05	10/03/05
EVS-1.3 Especificación del Alcance del EVS	1 día	11/03/05	11/03/05
EVS 3 Definición de requisitos del sistema	4 días	14/03/05	17/03/05
GPI Gestión del Proyecto	6 días	18/03/05	25/03/05
GPI-1 Estimación del Esfuerzo	2 días	18/03/05	21/03/05
GPI-2 Planificación del Sistema	4 días	22/03/05	25/03/05
GC Gestión de la Configuración del Sistema	5 días	28/03/05	01/04/05
GC-1 Identificación y Registro de Productos	5 días	28/03/05	01/04/05
CAL Gestión de la Calidad del Sistema	5 días	04/04/05	08/04/05
EVS-CAL-1 Identificación de las Propiedades de Calidad para el Sistema	2 días	04/04/05	05/04/05
ASI-CAL-1 Especificación de la política de gestión de calidad para el sistema	3 días	06/04/05	08/04/05

Procesos, Actividades y Tareas	Duración	Comienzo	Fin
ASI Análisis del sistema de información	76 días	11/04/05	25/07/05
ASI-1 Definición del sistema	2 días	11/04/05	12/04/05
ASI-1.1 Determinación del Alcance del Sistema	1 día	11/04/05	11/04/05
ASI-1.4 Identificación de los Usuarios Participantes y Finales	1 día	12/04/05	12/04/05
ASI-2 Establecimiento de Requisitos	20 días	13/04/05	10/05/05
ASI-2.1 Obtención de Requisitos	3 días	13/04/05	15/04/05
ASI-2.2 Especificación de Casos de Uso	12 días	18/04/05	03/05/05
ASI-2.3 Análisis de Requisitos	4 días	04/05/05	09/05/05
ASI-2.4 Validación de Requisitos	1 día	10/05/05	10/05/05
ASI-3 Identificación de los subsistemas de análisis	3 días	11/05/05	13/05/05
ASI-3.1 Determinación de Subsistemas de Análisis	3 días	11/05/05	13/05/05
ASI-4 Análisis de los Casos de Uso	7 días	16/05/05	24/05/05
ASI-4.1 Identificación de las Clases Asociadas a un Caso de Uso	4 días	16/05/05	19/05/05
ASI-4.2 Descripción de la interacción de objetos.	3 días	20/05/05	24/05/05
ASI-5 Análisis de Clases	9 días	25/05/05	06/06/05
ASI-5.1: Identificación de Responsabilidades y Atributos	3 días	25/05/05	27/05/05
ASI-5.2: Identificación de Asociaciones y Agregaciones	3 días	30/05/05	01/06/05

Procesos, Actividades y Tareas	Duración	Comienzo	Fin
ASI-5.3: Identificación de Generalizaciones	3 días	02/06/05	06/06/05
ASI-8: Definición de Interfaces de Usuario	11 días	07/06/05	21/06/05
ASI-8.1: Especificación de Principios Generales de la Interfaz	3 días	07/06/05	09/06/05
ASI-8.2: Identificación de Perfiles y Diálogos	2 días	10/06/05	13/06/05
ASI-8.3: Especificación de Formatos Individuales de la Interfaz de Pantalla	2 días	14/06/05	15/06/05
ASI-8.4: Especificación del Comportamiento Dinámico de la Interfaz	4 días	16/06/05	21/06/05
ASI-9: Análisis de Consistencia y Especificación de Requisitos	12 días	22/06/05	07/07/05
ASI-9.1: Verificación de los Modelos	3 días	22/06/05	24/06/05
ASI-9.2: Análisis de Consistencia entre Modelos	3 días	27/06/05	29/06/05
ASI-9.3: Validación de los Modelos	3 días	30/06/05	04/07/05
ASI-9.4: Elaboración de la Especificación de Requisitos Software (ERS)	3 días	05/07/05	07/07/05
ASI-10: Especificación del Plan de Pruebas	11 días	08/07/05	22/07/05
ASI-10.1: Definición del Alcance de las Pruebas	3 días	08/07/05	12/07/05
ASI-10.2: Definición de Requisitos del Entorno de Pruebas	2 días	13/07/05	14/07/05
ASI-10.3: Definición de las Pruebas de Aceptación del Sistema	6 días	15/07/05	22/07/05
ASI-11: Aprobación del Análisis del Sistema de Información	1 día	25/07/05	25/07/05
ASI-11.1: Presentación y Aprobación del Análisis del Sistema de Inf.	1 día	25/07/05	25/07/05

Procesos, Actividades y Tareas	Duración	Comienzo	Fin
DSI Diseño del sistema de información	62 días	26/07/05	19/10/05
DSI-1: definición de La arquitectura del Sistema	10 días	26/07/05	08/08/05
DSI-1.1: definición de Niveles de arquitectura	2 días	26/07/05	27/07/05
DSI-1.2: Identificación de Requisitos de Diseño y Construcción	3 días	28/07/05	01/08/05
DSI-1.5: Identificación de Subsistemas de Diseño	2 días	02/08/05	03/08/05
DSI-1.6: Especificación del Entorno Tecnológico	2 días	04/08/05	05/08/05
DSI-1.7: Especificación de Requisitos de Operación y Seguridad	1 día	08/08/05	08/08/05
DSI-2: Diseño de La arquitectura de Soporte	2 días	09/08/05	10/08/05
DSI-2.1: Diseño de Subsistemas de Soporte	1 día	09/08/05	09/08/05
DSI-2.2: Identificación de Mecanismos Genéricos de Diseño	1 día	10/08/05	10/08/05
DSI-3: Diseño de Casos de Uso Reales	12 días	11/08/05	26/08/05
DSI-3.1: Identificación de Clases asociadas a un Caso de Uso	4 días	11/08/05	16/08/05
DSI-3.2: Diseño de La Realización de Los Casos de Uso	4 días	17/08/05	22/08/05
DSI-3.3: Revisión de La Interfaz de Usuario	2 días	23/08/05	24/08/05
DSI-3.4: Revisión de Subsistemas de Diseño e Interfaces	2 días	25/08/05	26/08/05

Procesos, Actividades y Tareas	Duración	Comienzo	Fin
DSI-4: Diseño de Clases	16 días	29/08/05	19/09/05
DSI-4.1: Identificación de Clases adicionales	4 días	29/08/05	01/09/05
DSI-4.2: Diseño de asociaciones y agregaciones	2 días	02/09/05	05/09/05
DSI-4.3: Identificación de atributos de Las Clases	4 días	06/09/05	09/09/05
DSI-4.4: Identificación de Operaciones de Las Clases	3 días	12/09/05	14/09/05
DSI-4.5: Diseño de La Jerarquía	3 días	15/09/05	19/09/05
DSI-7: Verificación y aceptación de La arquitectura del Sistema	5 días	20/09/05	26/09/05
DSI-7.1: Verificación de Las Especificaciones de Diseño	2 días	20/09/05	21/09/05
DSI-7.2: análisis de Consistencia de Las Especificaciones de Diseño	2 días	22/09/05	23/09/05
DSI-7.3: aceptación de La arquitectura del Sistema	1 día	26/09/05	26/09/05
DSI-8: Generación de Especificaciones de Construcción	9 días	27/09/05	07/10/05
DSI-8.1: Especificación del Entorno de Construcción	3 días	27/09/05	29/09/05
DSI-8.2: Definición de Componentes y Subsistemas de Construcción	3 días	30/09/05	04/10/05
DSI-8.3: Elaboración de Especificaciones de Construcción	3 días	05/10/05	07/10/05

Procesos, Actividades y Tareas	Duración	Comienzo	Fin
DSI-10: Especificación Técnica del Plan de Pruebas	5 días	10/10/05	14/10/05
DSI-10.1: Especificación del Entorno de Pruebas	2 días	10/10/05	11/10/05
DSI-10.2: Especificación Técnica de Niveles de Prueba	2 días	12/10/05	13/10/05
DSI-10.3: Revisión de La Planificación de Pruebas	1 día	14/10/05	14/10/05
DSI-11: Establecimiento de Requisitos de Implantación	2 días	17/10/05	18/10/05
DSI-11.1: Especificación de Requisitos de Documentación de Usuario	1 día	17/10/05	17/10/05
DSI-11.2: Especificación de Requisitos de Implantación	1 día	18/10/05	18/10/05
DSI-12: aprobación del Diseño del Sistema de Información	1 día	19/10/05	19/10/05
DSI-12.1: Presentación y aprobación del Diseño del Sistema de Información	1 día	19/10/05	19/10/05
CSI Construcción del Sistema de Información	67 días	20/10/05	20/01/06
CSI-1: Preparación del Entorno de Generación y Construcción	6 días	20/10/05	27/10/05
CSI-1.1: Implantación de la Base de Datos Física o Ficheros	2 días	20/10/05	21/10/05
CSI-1.2: Preparación del Entorno de Construcción	4 días	24/10/05	27/10/05

Procesos, Actividades y Tareas	Duración	Comienzo	Fin
CSI-2: Generación Del Código de los Componentes y Procedimientos	35 días	28/10/05	15/12/05
CSI-2.1: Generación del Código de Componentes	17 días	28/10/05	21/11/05
CSI-2.2: Generación del Código de los Procedimientos de Operación y Seg.	18 días	22/11/05	15/12/05
CSI-3: Ejecución de las Pruebas Unitarias	8 días	16/12/05	27/12/05
CSI-3.1: Preparación del Entorno de las Pruebas Unitarias	6 días	16/12/05	23/12/05
CSI-3.2: Realización y Evaluación de las Pruebas Unitarias	2 días	26/12/05	27/12/05
CSI-4: Ejecución de las Pruebas de Integración	9 días	28/12/05	09/01/06
CSI-4.1: Preparación del Entorno de las Pruebas de Integración	6 días	28/12/05	04/01/06
CSI-4.2: Realización de las Pruebas de Integración	2 días	05/01/06	06/01/06
CSI-4.3: Evaluación del Resultado de las Pruebas de Integración	1 día	09/01/06	09/01/06
CSI-5: Ejecución de las Pruebas del Sistema	3 días	10/01/06	12/01/06
CSI-5.1: Preparación del Entorno de las Pruebas del Sistema	1 día	10/01/06	10/01/06
CSI-5.2: Realización de las Pruebas del Sistema	1 día	11/01/06	11/01/06
CSI-5.3: Evaluación del Resultado de las Pruebas del Sistema	1 día	12/01/06	12/01/06

Procesos, Actividades y Tareas	Duración	Comienzo	Fin
CSI-6: Elaboración de los Manuales De Usuario	1 día	13/01/06	13/01/06
CSI-6.1: Elaboración de los Manuales de Usuario	1 día	13/01/06	13/01/06
CSI-7: Definición de la formación de Usuarios Finales	3 días	16/01/06	18/01/06
CSI-7.1: Definición del Esquema de Formación	2 días	16/01/06	17/01/06
CSI-7.2: Especificación de los Recursos y Entornos de Formación	1 día	18/01/06	18/01/06
CSI-9: Aprobación del Sistema de Información	2 días	19/01/06	20/01/06
CSI-9.1: Presentación y Aprobación del Sistema de Información	2 días	19/01/06	20/01/06
IAS Implantación y Aceptación del Sistema	10 días	23/01/06	03/02/06
IAS-3: Incorporación del Sistema al Entorno de Operación	2 días	23/01/06	24/01/06
IAS-3.1: Preparación de la Instalación	1 día	23/01/06	23/01/06
IAS-3.2: Realización de la Instalación	1 día	24/01/06	24/01/06
IAS-5: Pruebas de Implantación del Sistema	3 días	25/01/06	27/01/06
IAS-5.1: Preparación de las Pruebas de Implantación	1 día	25/01/06	25/01/06
IAS-5.2: Realización de las Pruebas de implantación	1 día	26/01/06	26/01/06
IAS-5.3: Evaluación del Resultado de las Pruebas de Implantación	1 día	27/01/06	27/01/06

Procesos, Actividades y Tareas	Duración	Comienzo	Fin
IAS-6: Pruebas de Aceptación del Sistema	5 días	30/01/06	03/02/06
IAS-6.1: Preparación de las Pruebas de Aceptación	2 días	30/01/06	31/01/06
IAS-6.2: Realización de las Pruebas de Aceptación	2 días	01/02/06	02/02/06
IAS-6.3: Evaluación del Resultado de las Pruebas de Aceptación	1 día	03/02/06	03/02/06
TES Tesis	**18 días**	**06/02/06**	**01/03/06**
TES-1 Preparación de la documentación de la Tesis	10 días	06/02/06	17/02/06
TES-2 Revisión y Aprobación del Documento Final	3 días	20/02/06	22/02/06
TES-3 Presentación de la Tesis	5 días	23/02/06	01/03/06

Tabla 4-1 - Procesos, Actividades y Tareas Planificadas

El proceso indicado como TES Tesis, no pertenece a la metodología, sin embargo se lo definió de manera similar al resto. El mismo tiene por objetivo la preparación, revisión y aprobación del documento final y el armado de la presentación para la defensa.

A continuación se presenta un resumen del tiempo asignado a cada tarea:

Proceso	Duración (días)
EVS Estudio de Viabilidad del Sistema	13
GPI Gestión del Proyecto	6
GC Gestión de la Configuración del Sistema	5
CAL Gestión de la Calidad del Sistema	5
ASI Análisis del sistema de información	76
DSI Diseño del sistema de información	62

Proceso	Duración (días)
CSI Construcción del Sistema de Información	67
IAS Implantación y Aceptación del Sistema	10
TES Tesis	18
Total	**262**

Tabla 4-2 - Resumen de la Planificación

5 Calidad, Seguridad y Gestión de Configuración

En este capítulo se completará la información correspondiente a las interfaces que recomienda la metodología, agregadas a la ya descrita en el capítulo 5 (Gestión de Proyectos).

5.1 Aseguramiento de la calidad

En este apartado se busca dar un marco común de referencia para la planificación de la calidad del producto a obtener.

5.1.1 Identificación de las Propiedades de Calidad para el Sistema

Se definen aquéllas propiedades que se consideran importantes para el sistema y que además permiten evaluar la calidad (EVS-CAL 1.2):

Fiabilidad:

Debe responder adecuadamente a los eventos originados por el usuario. Debe guardar la información requerida por el investigador.

Portabilidad:

Debe ser sencillamente instalable en cualquier PC que reúna los requisitos mínimos que se definan. Su funcionamiento debe ser similar aunque difiera el hardware entre dos equipos cualesquiera donde se instale el software. Este proyecto se desarrollará bajo sistema operativo Windows, sin abandonar las condiciones que lo hagan fácilmente portable a otros sistemas operativos (Linux) en el futuro.

Facilidad de Uso:

Debe ser tan sencillo de usar que ni siquiera existirá un manual de usuario para el niño que realiza la evaluación. Para el investigador, alcanzará con la descripción del archivo que contiene el trazado de la sesión.

Facilidad de Mantenimiento:

Deberá estructurarse de forma tal que la interfaz visual pueda modificarse (por ejemplo, cambiar la librería 3D) y el resto de los componentes del sistema permanecer invariables.

5.1.2 Establecimiento del Plan de Aseguramiento de Calidad

Una vez acordados los requisitos de calidad, se pasará a la planificación que nos facilitará su aseguramiento.

Actividad	Tareas
ASI-2 Establecimiento de Requisitos	ASI-2.4 Validación de Requisitos
ASI-9: Análisis de Consistencia y Especificación de Requisitos	ASI-9.1: Verificación de los Modelos ASI-9.2: Análisis de Consistencia entre Modelos ASI-9.3: Validación de los Modelos
ASI-10: Especificación del Plan de Pruebas	ASI-10.1: Definición del Alcance de las Pruebas ASI-10.2: Definición de Requisitos del Entorno de Pruebas ASI-10.3: Definición de las Pruebas de Aceptación del Sistema
DSI-10: Especificación Técnica del Plan de Pruebas	DSI-10.1: Especificación del Entorno de Pruebas DSI-10.2: Especificación Técnica de Niveles de Prueba DSI-10.3: Revisión de La Planificación de Pruebas
CSI-3: Ejecución de las Pruebas Unitarias	CSI-3.1: Preparación del Entorno de las Pruebas Unitarias CSI-3.2: Realización y Evaluación de las Pruebas Unitarias
CSI-4: Ejecución de las Pruebas de Integración	CSI-4.1: Preparación del Entorno de las Pruebas de Integración CSI-4.2: Realización de las Pruebas de Integración CSI-4.3: Evaluación del Resultado de las Pruebas de Integración
CSI-5: Ejecución de las Pruebas del Sistema	CSI-5.1: Preparación del Entorno de las Pruebas del Sistema CSI-5.2: Realización de las Pruebas del Sistema CSI-5.3: Evaluación del Resultado de las Pruebas del Sistema

Tabla 5-1 - Actividades y tareas para Asegurar la calidad

5.2 Seguridad

Entendemos por seguridad desde el punto de vista del sistema de información, aquella que se relaciona directamente al producto y también, la relacionada al proceso de desarrollo.

5.2.1 Seguridad del producto

Dado que el producto es monousuario, no existen consideraciones particulares que deban tenerse en cuenta en el aspecto seguridad. En el manual de uso para el investigador, se hará incapié en que se siga una política de copias de resguardo adecuada.

Desde el punto de vista del usuario, en caso de infortunio, la reinstalación del producto, dejará al mismo en condiciones de uso nuevamente.

5.2.2 Seguridad del proceso de desarrollo

El proceso de desarrollo conlleva una propia política de seguridad, simplificada en el presente por sus características a:

- Copia de Resguardo de Documentos.
- Copia de Resguardo de Código Fuente y Librerías.
- Copia de Resguardo de Archivos de Herramientas de desarrollo.

Dichas Copia de Resguardo se realizarán semanalmente, grabándose en CD.

5.3 Gestión de Configuración

Esta actividad está dirigida a la gestión del mantenimiento tanto del proyecto como del producto final, y su objetivo es identificar, definir, proporcionar información y controlar los cambios en la configuración del sistema, así como las modificaciones y versiones de los mismos.

Las particularidades del presente proyecto, hacen que la gestión de configuración se realice con base en dos aspectos:

- El documento del proyecto
- El software

Se adopta una nomenclatura simple que permite realizar la gestión a partir de la denominación de archivos y directorios.

No se incluye en el presente, la gestión de las solicitudes de modificación del producto en la etapa de mantenimiento, dado que ella no forma parte del proyecto en sí. Sin embargo, será el primer paso a tener en cuenta antes que el producto entre en producción.

5.3.1 Gestión de Configuración del Documento del Proyecto

El presente documento constituye el documento de tesis. El mismo ha sido dividido en capítulos y anexos, cada uno de los cuales es un producto a gestionar. Al finalizar el proyecto, se dará lugar a la formación de un único documento conformado por la última versión de cada producto, es decir, de cada capítulo. Para poder realizar esta tarea, cada capítulo ha sido adecuado para incluir la información de procesos concretos como sugiere la metodología. Al ser un desarrollo unipersonal, permite justificar el presente tratamiento a la gestión de configuración, que en caso de proyectos desarrollados en grupo, sería inaceptable.

Cada producto se denominará con la sigla de proyecto (**wok-**) y se le agregará una indicación de Capítulo (**Cap**) o Anexo (**Anx**). A continuación irá la letra **v**, seguida del número de versión. Luego, vendrá el número de la revisión y finalmente, una letra a modo de indicación interna. Estos tres últimos separados por puntos, seguidos de la extensión del archivo (**.doc**).

Por ejemplo, el primer documento de trabajo del presente capítulo, se denominará:

wok-Cap5v1.0.a.doc

5.3.2 Gestión de Configuración del Software

En este caso, cada archivo que compone el código fuente, constituirá en sí un producto. Asimismo, se agrega como producto, el conjunto de archivos que integran las texturas y los objetos 3D que utilizará el programa como así también la instalación final. La denominación adoptada de iguales características será en cada caso incluida:

- Archivos de Código Fuente: Se indicará en la parte superior a modo de comentario, seguida de las modificaciones realizadas a la versión previa. Serán acumulativas, ordenadas de forma tal de ver la última revisión en el primer lugar. El archivo de código fuente de proyecto (.prj en Delphi), indicará las revisiones del resto del código fuente compilado.
- Archivos 3D: Se archivarán en un directorio dado. El directorio será el que incluya la nomenclatura.
- Instalación: Será el archivo de instalación del producto que incluirá la última revisión del software.

6 Análisis del Sistema de Información

6.1 Definición del Sistema

En este apartado completaremos la información obtenida en el proceso de EVS, básicamente haciendo uso de diagramas UML que nos permitirán describir el modelo del negocio, los casos de uso, las secuencias y clases fundamentales del sistema.

6.1.1 Determinación del Alcance del Sistema

A continuación se presenta el diagrama UML de Modelo del Dominio (Ilustración 6-1) del sistema.

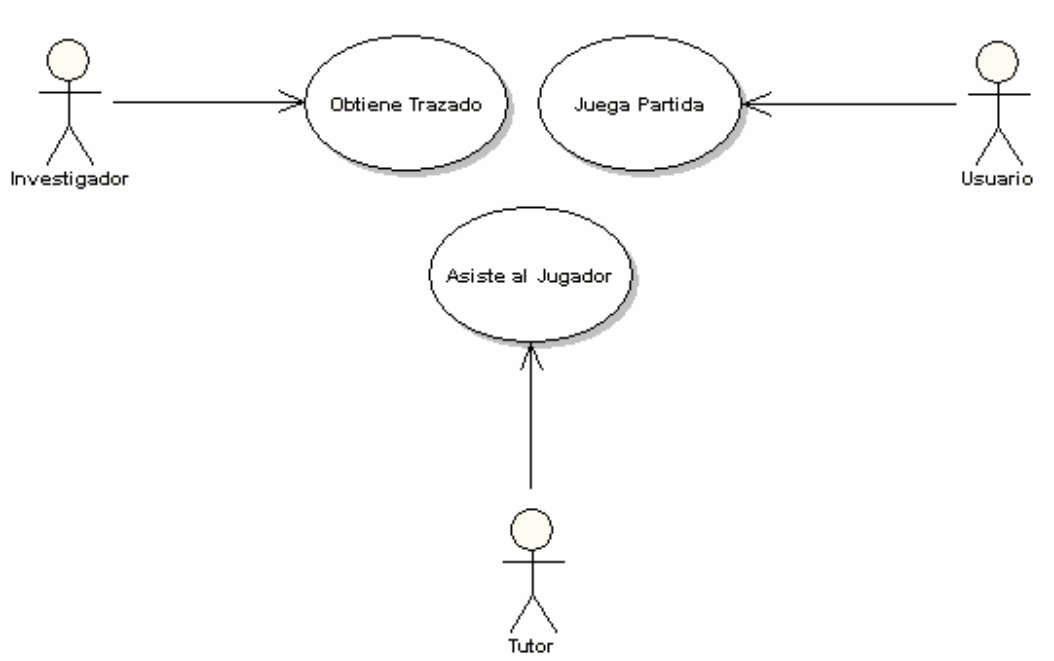


Ilustración 6-1 – Modelo del Dominio

En el gráfico se observa que el modelo del dominio presenta tres casos de uso. Dos de ellos coinciden con la descripción del sistema obtenida en el EVS en la que se habló de dos visiones, la del investigador y la del usuario. El investigador desea obtener el trazado de las decisiones tomadas por el usuario y las acciones llevadas a cabo por el programa tendientes a guiar al usuario en la resolución del problema o caso planteado. El usuario es un niño al que se le quiere enseñar a jugar un juego. Sin embargo, vemos que aparece un caso de uso más que se ha agregado a esta altura y que resulta fundamental tener en cuenta y es la aparición de un tutor que deberá ir guiando al usuario durante el desarrollo de la partida. Este tutor virtual será quien se encargue de las tareas de verificar el momento adecuado en el cual mostrar ayudas para luego realizar la indicación apropiada.

En el EVS se obvió esta definición, dado que es una tarea a realizar por el sistema. Los conceptos presentados, serán ampliados en la especificación de los Casos de Uso como se verá en 7.2. Establecimiento de los casos de Uso.

6.1.2 Identificación del los Usuarios Participantes y Finales

En el diagrama del Modelo del Dominio presentado en el punto anterior vemos que aparecen tres actores. A continuación los clasificaremos en actores principales y secundarios:

a) Actores Principales: En cuya categoría se incluyen aquellos actores que actúan directamente en el sistema y que su interacción es decisiva para el sistema en cuestión. Dentro de ellos, entonces, vamos a incluir a:

- Usuario
- Tutor

b) Actores Secundarios: Son aquellos actores que no desempeñan un papel que altera el transcurso del uso del sistema. El único actor que será incluido es:

- Investigador

Como puede observarse, el investigador es el destinatario principal del sistema y no es un actor principal desde el punto de vista del desarrollo del proyecto. Existen muchos ejemplos similares en desarrollos empresariales como pueden ser las bases de datos multidimensionales. En ellas hay un único destinatario final, que es un gerente de alto rango cuya interacción se limita a la obtención de un único reporte. Los demás actores principales realizan la interacción necesaria con el sistema (diseño de cubos, carga de datos, diseño y generación de la consulta multidimensional) para que él pueda obtener dicho reporte presionando un solo botón.

Por otra parte, se incluye al Tutor como actor "virtual", a fin de asociar aquellas funcionalidades del software que tienen que ver exclusivamente con el proceso de aprendizaje, como es la de asistir al usuario.

6.2 Establecimiento de Requisitos

6.2.1 Obtención de Requisitos

De los requisitos obtenidos en el proceso de EVS, pasamos a presentar el diagrama UML de Casos de Uso para el sistema (Ilustración 6-2), que serán descritos y analizados en el próximo ítem (Especificación de Casos de Uso).

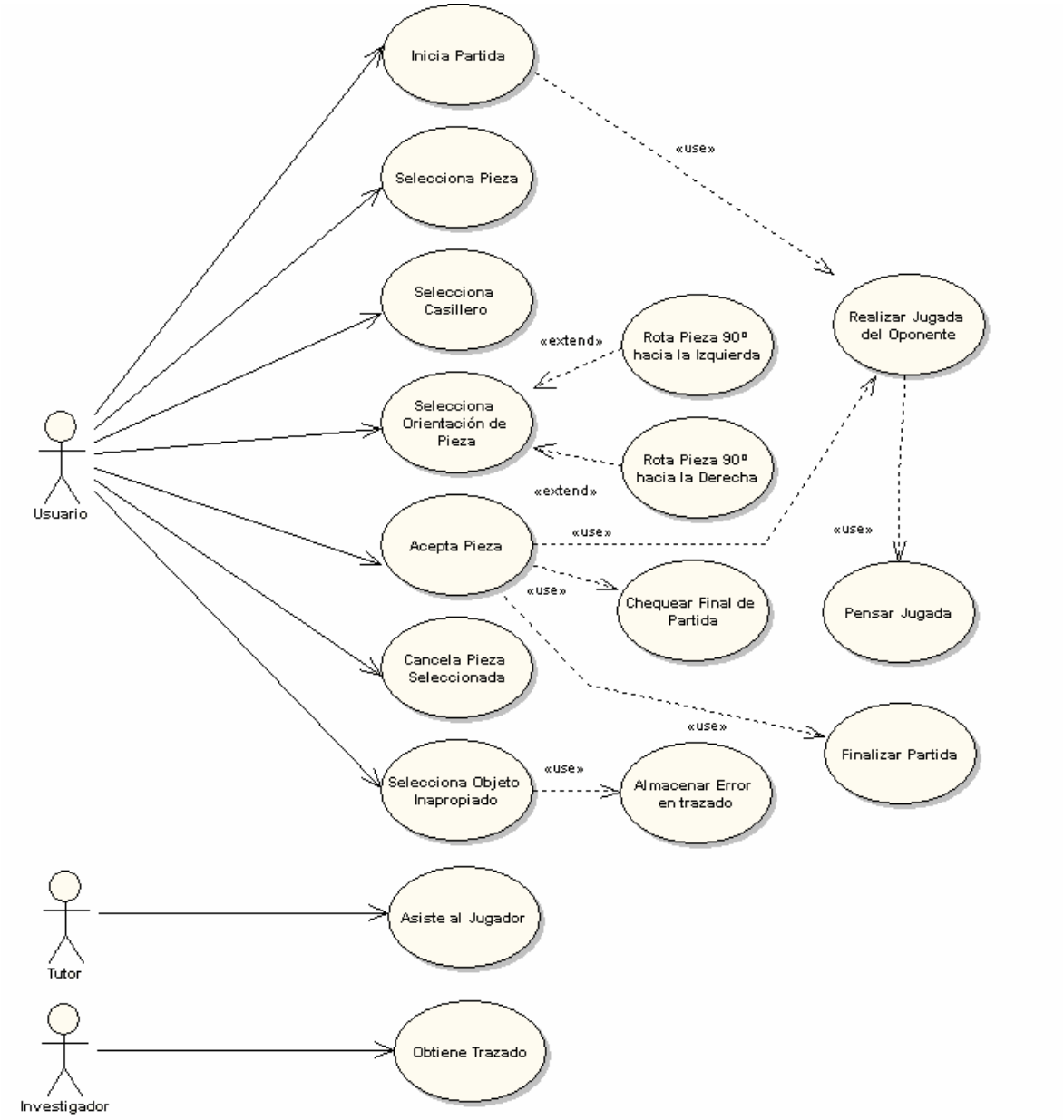


Ilustración 6-2 - Casos de Uso

Los casos de uso representan a los requisitos funcionales planteados en el EVS. Los requerimientos de performance, requerimientos de bases de datos lógicas, restricciones de diseño, atributos del sistema software, revisados durante la presente tarea, mantienen lo especificado en el EVS.

6.2.2 Especificación de Casos de Uso

A fin de describir los casos de usos presentados en el punto anterior, a continuación se analiza cada uno de ellos, siendo el objetivo de esta tarea especificar cada caso de uso

identificado, desarrollando el escenario. Para lo anterior se utilizan Diagramas de Secuencia. Sin embargo, antes de entrar en el detalle de cada uno de los casos de uso, se va a presentar un diagrama de estados (como lo sugiere la metodología). En particular, este diagrama de estados va a simplificar luego la definición del escenario en cada caso de uso, dado que simplemente se hará referencia al estado donde se encuentra al momento de darse las precondiciones (eventos).

6.2.2.1 Diagrama de Estados

En la especificación observamos la siguiente secuencia:

- Seleccionar Pieza(1)
- Seleccionar Casillero(2)
- Rotar Pieza(3)
- Aceptar Pieza(4)

Podemos considerar entonces un diagrama de estados que represente el modelo de la partida desde el punto de vista dinámico. Para ello, se hará uso de un estado inicial, un estado final y tres estados intermedios a saber:

- **Estado Inicial**: Piezas colocadas en posición inicial, sorteado el color de piezas de usuario y jugada de oponente realizada en caso de habérsele asignado piezas negras.
- **Antes de Seleccionar la Pieza**: Comenzó la partida o el oponente finalizó su jugada y el programa espera que el usuario seleccione una pieza a mover.
- **Pieza Seleccionada**: Normalmente estará en este estado cuando se encuentre a la espera de la selección del casillero (aunque se permite la selección de una pieza distinta a la ya elegida previamente cancelando la pieza seleccionada).
- **Casillero Seleccionado**: A este estado se llegará si estando en el estado *Pieza Seleccionada*, se elige un casillero donde puede o podría jugarse la pieza. "Puede jugarse" significa que la pieza puede aceptarse sin necesidad de rotarla previamente. "Podría jugarse", significa que si se rota adecuadamente la pieza, la misma "Puede jugarse". Al ingresar a este estado, estarán visibles los indicadores de rotación de la pieza.
- **Estado Final**: Uno de los jugadores o los dos no pueden colocar más piezas.

A continuación se muestra el diagrama de estados conceptual definido (Ilustración 6-3) que corresponde al controlador de la partida (abstracción que se hace para indicar que existirá un objeto que estará a cargo de mantener un desarrollo del juego):

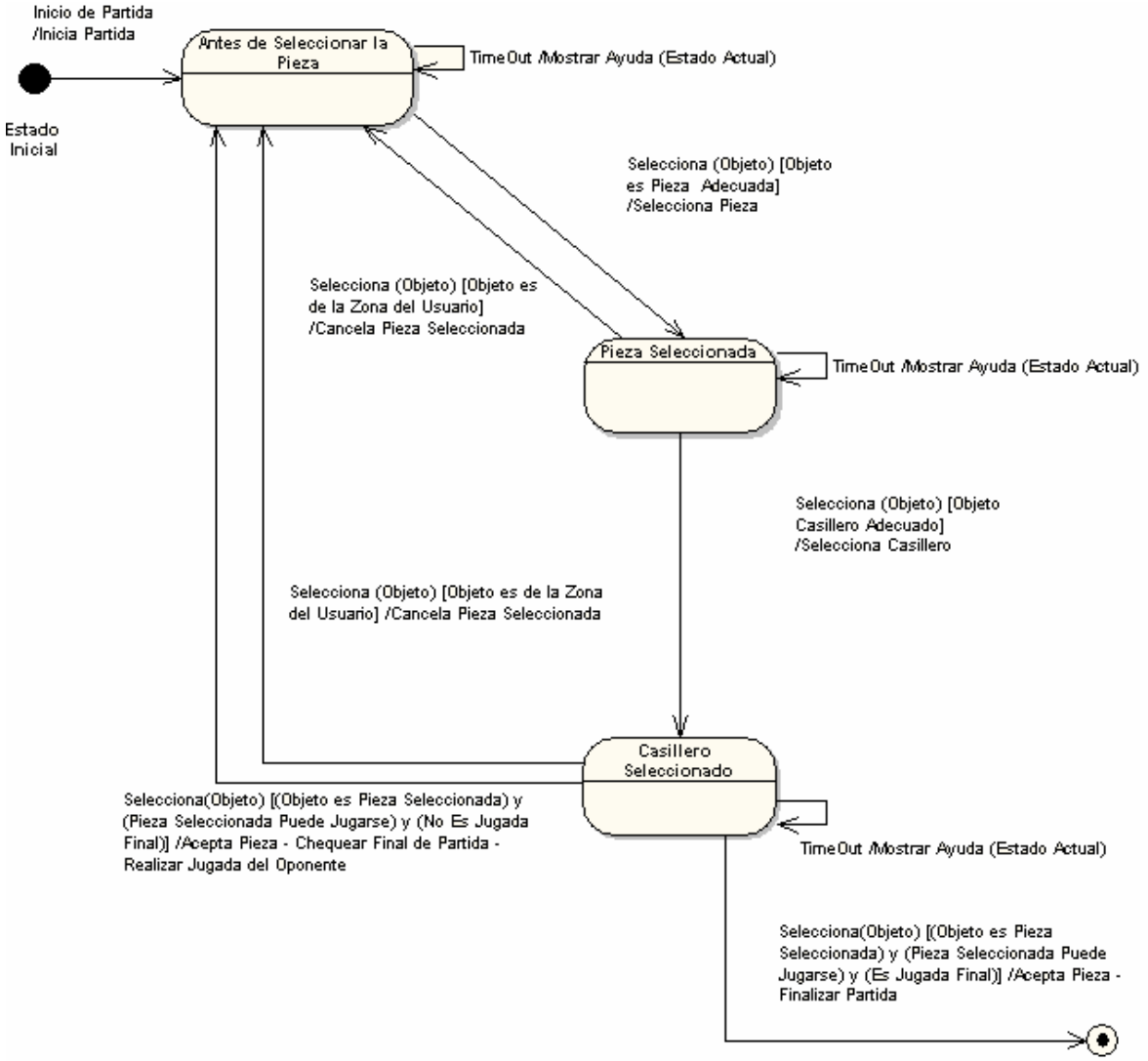


Ilustración 6-3 - Diagrama de Estados

En la Ilustración 6-3 pueden observarse los estados referidos previamente y los conectores dirigidos que indican las transiciones y que muestran la siguiente información en ese orden:

- El evento o suceso que da origen al cambio de estado.
- La condición de seguridad (guard) para que se cambie de estado. Se representa entre corchetes. Si existen varias condiciones, se anidan con "y" u "o". Se coloca una barra "/" al final para separarla de la próxima indicación.
- La acción a tomar al salir de un estado y antes de ingresar al próximo.

En este caso, vemos que solamente se han puesto de manifiesto dos eventos:

- Selecciona(Objeto): Indica que un Objeto ha sido seleccionado
- TimeOut: Indica que ha transcurrido el tiempo de espera asignado para que el programa ayude el usuario en su jugada.

El evento **Selecciona** (Objeto), disparará una acción dependiente del objeto y del estado en que se encuentra la partida. Al suceder un evento **TimeOut**, será de aplicación el caso de uso "Asiste al Jugador" que tomará el actor virtual denominado "Tutor" y cuya acción dependerá de ciertas condiciones de entorno y del estado actual.

A continuación se detallan las transiciones representadas en el diagrama de estados:

Estado Anterior	Estado Siguiente	Evento	Condición	Acciones
Estado Inicial	Antes de ...	Inicio de partida	-	Inicia Partida
Antes de ...	Antes de ...	TimeOut	-	Mostrar Ayuda (Antes de...)
Antes de ...	Pieza Selec.	Selecciona(Objeto)	Objeto es Pieza Adecuada	Selecciona Pieza
Pieza Selec.	Pieza Selec.	TimeOut	-	Mostar Ayuda (Pieza Selec.)
Pieza Selec.	Pieza Selec.	Selecciona(Objeto)	Objeto es Casillero Adecuado	Selecciona Casillero
Pieza Selec.	Antes de ...	Selecciona(Objeto)	Objeto es de la Zona del jugador	Cancela Pieza Seleccionada
Casillero Selec.	Casillero Selec.	TimeOut	-	Mostar Ayuda (Casillero Selec.)
Casillero Selec.	Antes de ...	Selecciona(Objeto)	Objeto es de la Zona del jugador	Cancela Pieza Seleccionada
Casillero Selec.	Antes de ...	Selecciona(Objeto)	(Objeto es Pieza Seleccionada) y (Pieza Seleccionada Puede Jugarse) y (No Es Jugada Final)	Acepta Pieza Chequear Final de Partida Realizar Jugada del Oponente
Casillero Selec.	Estado Final	Seleccionar(Objeto)	(Objeto es Pieza Seleccionada) y (Pieza Seleccionada Puede Jugarse) y (Es Jugada Final)	Si Puede Jugar, Acepta Pieza Si el oponente Puede Jugar Realizar Jugada del Oponente Finalizar Partida

Tabla 6-1 - Cambios de Estado

Con ello, hemos representado los estados a la espera de tres de las cuatro acciones que integran la secuencia de una movida de la siguiente forma:

Acción	Estado en espera de la Acción
Seleccionar Pieza(1)	Antes de Seleccionar la Pieza
Seleccionar Casillero(2)	Pieza Seleccionada
Aceptar Pieza(4)	Casillero Seleccionado

Tabla 6-2 - Acciones Esperadas según el Estado

Como puede observarse, no se hace mención a la acción Rotar Pieza (3), ya que en este nivel de conceptualización, la rotación de la pieza no provoca un cambio de estado.

6.2.2.2 Especificaciones

Para completar los casos de uso, es preciso especificar información relativa a:

- Descripción del escenario, es decir, cómo un actor interactúa con el sistema, y cuál es la respuesta obtenida.
- Precondiciones y poscondiciones.
- Identificación de interfaces de usuario.
- Condiciones de fallo que afectan al escenario, así como la respuesta del sistema (escenarios secundarios).

A continuación se especifica cada caso de uso, agregándose en cada uno de ellos, las clases que a priori se observan y que luego se integrarán en la actividad Análisis de casos de uso:

Inicia Partida

- **Escenario**:
  Este caso de uso corresponde al momento en el que arranca el programa o una vez que finaliza una partida y el usuario desea jugar nuevamente o pasa un tiempo configurado para que reinicie el juego desde el principio. Cualquiera de estos tres eventos son los que se consideran inicio de partida a fin de simplificar el esquema. Todo quedará dispuesto para que el usuario genere su primera interacción (seleccionar pieza), que dependerá del color de pieza que se le haya asignado.

- **Precondiciones:**
  Ocurre cualquiera de los tres posibles eventos de Inicio de Partida.

- **Postcondiciones**:
  Se deja al escenario en condiciones de juego con cada pieza en su posición inicial. Esto implica también, que se asignará el color de piezas al usuario (de piezas blancas o negras), se realizarán los movimientos del oponente en caso de haberse asignado al usuario piezas negras y dejará el tablero en condiciones de realizar la primera selección de pieza. Se registra la hora de inicio de la partida y las condiciones particulares configuradas que puedan diferenciar esta partida de otras.

- **Interfaz**:
  Se visualizará el tablero en las condiciones indicadas y los bordes, cada uno con su color. Teniendo en cuenta el color de piezas asignadas al jugador, se presentará el tablero de forma tal que sus piezas queden en la parte inferior de la pantalla y las del oponente en la superior, quedando en posición central la zona de juego.

- **Condiciones de fallo**:
  No aplicable.

Selecciona Pieza

- **Escenario**:
  El estado actual es Antes de Seleccionar Pieza.

- **Precondiciones**:
  El Objeto seleccionado es una pieza adecuada, es decir, es propia y no ha sido jugada previamente.

- **Postcondiciones**:
  La pieza queda seleccionada. Se pasa al estado Pieza Seleccionada. Se registra el suceso.

- **Interfaz**:
  La pieza se desplaza verticalmente hacia arriba, destacándose del resto, indicándose así su nueva condición. Cuando se indica "se desplaza", significa que se quiere mostrar la trayectoria que realiza la pieza y no el cambio de posición.

- **Condiciones de fallo**:
  Al tener que esperar que la pieza quede en su nueva posición para cambiar de estado, el sistema no aceptará las interacciones del usuario que ocurran durante dicho desplazamiento. Con ello, se registrarán esas interacciones anticipadas.

Selecciona Casillero

- **Escenario**:
  El estado actual es Pieza Seleccionada o Casillero Seleccionado (en este caso se desea cambiar el casillero elegido previamente).

- **Precondiciones**:
  El Objeto seleccionado es un casillero adecuado, es decir, es un casillero que está libre y la pieza seleccionada podría jugarse en dicha posición, con o sin rotación previa.

- **Postcondiciones**:
  El casillero queda seleccionado. Si el estado era Pieza Seleccionada, se pasa al nuevo estado Casillero Seleccionado. Se registra el suceso.

- **Interfaz**:
  La pieza seleccionada, se desplazará horizontalmente hasta posicionarse sobre el casillero elegido.

- **Condiciones de fallo**:
  Al tener que esperar que la pieza quede en su nueva posición para cambiar de estado o aceptar otra nueva intervención de parte del usuario, el sistema no aceptará las interacciones del usuario que ocurran durante dicho desplazamiento. Se registrarán esas interacciones anticipadas.

Selecciona Orientación de Pieza

- **Escenario**:
  El estado actual es Casillero Seleccionado.

- **Precondiciones**:
  El objeto seleccionado es un indicador de rotación.

- **Postcondiciones**:
  Cambia la orientación de la pieza 90º en el sentido que indica el objeto seleccionado. Se registra la rotación.

- **Interfaz**:
  La pieza seleccionada, rotará en forma animada 90º en el sentido indicado.

- **Condiciones de fallo**:
  Al tener que esperar que la pieza quede en su nueva posición para aceptar otra nueva intervención de parte del usuario, el sistema no aceptará las interacciones del usuario que ocurran durante dicha rotación. Se registrarán esas interacciones anticipadas.

Rota Pieza 90º hacia la Derecha

- **Escenario**:
  El estado actual es Casillero Seleccionado.

- **Precondiciones**:
  El objeto seleccionado es un indicador de rotación hacia la derecha.

- **Postcondiciones**:
  Cambia la orientación de la pieza 90º hacia la derecha. Se registra la rotación.

- **Interfaz**:
  La pieza seleccionada, rotará en forma animada 90º hacia la derecha.

- **Condiciones de fallo**:
  Al tener que esperar que la pieza quede en su nueva posición para aceptar otra nueva intervención de parte del usuario, el sistema no aceptará las interacciones del usuario que ocurran durante dicha rotación. Se registrarán esas interacciones anticipadas.

Rota Pieza 90º hacia la Izquierda

- **Escenario**:
  El estado actual es Casillero Seleccionado.

- **Precondiciones**:
  El objeto seleccionado es un indicador de rotación hacia la izquierda.

- **Postcondiciones**:
  Cambia la orientación de la pieza 90º hacia la izquierda. Se registra la rotación.

- **Interfaz**:
  La pieza seleccionada, rotará en forma animada 90º hacia la izquierda.

- **Condiciones de fallo**:
  Al tener que esperar que la pieza quede en su nueva posición para aceptar otra nueva intervención de parte del usuario, el sistema no aceptará las interacciones del usuario que ocurran durante dicha rotación. Se registrarán esas interacciones anticipadas.

Acepta Pieza

- **Escenario**:
  El estado actual es Casillero Seleccionado.

- **Precondiciones**:
  El objeto seleccionado es la pieza seleccionada y puede jugarse en el casillero seleccionado.

- **Postcondiciones**:
  La pieza es aceptada. La movida del usuario se ha completado y no hay vuelta atrás. Se registra que la movida se ha completado.

- **Interfaz**:
  La pieza descenderá verticalmente hasta apoyarse sobre el casillero seleccionado.

- **Condiciones de fallo**:
  Al tener que esperar que la pieza quede en su nueva posición para cambiar de estado, el sistema no aceptará las interacciones del usuario que ocurran durante dicho desplazamiento. Con ello, se registrarán esas interacciones anticipadas.

Cancela Pieza Seleccionada

- **Escenario**:
  El estado actual es Pieza Seleccionada o Casillero Seleccionado.

- **Precondiciones**:
  El Objeto seleccionado es un objeto del sector del usuario (casillero donde se "apoyan" las piezas en su posición inicial o es una pieza aun no jugada).

- **Postcondiciones**:
  El casillero queda seleccionado. Si el estado era Pieza Seleccionada, se pasa al nuevo estado Casillero Seleccionado. Se registra el suceso.

- **Interfaz**:
  La pieza seleccionada, se elevará verticalmente hasta destacarse del resto. Luego se desplazará horizontalmente hasta posicionarse sobre el casillero de posición de origen. Por último descenderá verticalmente hasta posarse sobre dicho casillero.

- **Condiciones de fallo**:
  Al tener que esperar que la pieza quede en su posición original para cambiar de estado o aceptar otra nueva intervención por parte del usuario, el sistema no aceptará las interacciones del usuario que ocurran durante dicho desplazamiento. Se registrarán esas interacciones anticipadas.

Selecciona Objeto Inapropiado

- **Escenario**:
  En cualquiera de los tres estados.

- **Precondiciones**:
  El objeto seleccionado, carece de sentido pudiendo ser tanto una pieza del oponente, una pieza propia ya jugada o el casillero de posición de origen de las piezas del oponente.

- **Postcondiciones**:
  Se registra el error.

- **Interfaz**:
  El programa deberá emitir una señal sonora que intentará que se interprete como un error.

- **Condiciones de fallo**:
  No aplicable.

Asiste al Jugador

Como se anticipó al explicarse el diagrama de estados, Asiste al jugador será un caso de uso que dependerá del estado actual, por ello, se describirá cada uno de los casos posibles.

Asiste al Jugador (Antes de Seleccionar la Pieza)

- **Escenario**:
  Se encuentra en el estado Antes de Seleccionar la Pieza.

- **Precondiciones**:
  Se cumplió el tiempo de espera asignado para este estado en esta condición de la partida.

- **Postcondiciones**:
  Se "Piensa" una movida Posible. Se registra la ocurrencia del suceso.

- **Interfaz**:
  Se muestra la ayuda, que en este caso será la indicación de una pieza del usuario que pueda ser jugada.

- **Condiciones de fallo**:
  No aplicable.

Asiste al Jugador (Pieza Seleccionada)

- **Escenario**:
  Se encuentra en el estado Pieza Seleccionada.

- **Precondiciones**:
  Se cumplió el tiempo de espera asignado para este estado en esta condición de la partida.

- **Postcondiciones**:
  Se "Piensa" una movida Posible que use la Pieza Seleccionada. Se registra la ocurrencia del suceso.

- **Interfaz**:
  Se muestra la ayuda, que en este caso será la indicación de un casillero de juego donde es posible colocar la pieza. Es posible, denota que se puede aceptar directamente en la orientación actual de la pieza o rotaciones de la pieza mediante.

- **Condiciones de fallo**:
  No aplicable.

Asiste al Jugador (Casillero Seleccionado)

- **Escenario**:
  Se encuentra en el estado Casillero Seleccionado.

- **Precondiciones**:
  Se cumplió el tiempo de espera asignado para este estado en esta condición de la partida.

- **Postcondiciones**:
  Se "Piensa" una movida Posible que use la Pieza Seleccionada en el Casillero seleccionado. Si la orientación de la jugada pensada coincide con la orientación actual de la pieza, la ayuda será para que se acepte la pieza. Caso contrario, se ayudará para indicar la rotación adecuada. Se registra la ocurrencia del suceso.

- **Interfaz**:
  Se muestra la ayuda, que en este caso será la indicación de la pieza seleccionada en caso que "Puede Jugarse" la pieza seleccionada en el casillero seleccionado. Sino, se destacará el indicador de rotación que corresponda dado que se cumple la condición de "Podría Jugarse", pero no se cumple que "Puede Jugarse".

- **Condiciones de fallo**:
  No aplicable.

Obtiene Trazado

- **Escenario**:
  En cualquier momento.

- **Precondiciones**:
  Se invoca una combinación de teclas (CTRL+T).

- **Postcondiciones**:
  Cancela la partida en curso, lo registra y abre un explorador de archivos en la carpeta donde se encuentran lo archivos conteniendo el trazado de las partidas.

- **Interfaz**:
  La que brinda el sistema operativo.

- **Condiciones de fallo**:
  No aplicable.

Chequear Final de Partida

- **Escenario**:
  El usuario aceptó la pieza seleccionada y el sistema necesita saber si el juego va a continuar. Existen varios casos que se van a presentar, por ello es que se incluye una tabla de decisión.

- **Precondiciones y Postcondiciones:**

Recordamos las condiciones de final de partida establecidas como requisitos:

Si a su turno el jugador Negro no puede jugar más piezas y el Blanco aun sí, el jugador Blanco juega una pieza y es el ganador.

Si a su turno el jugador Blanco no puede jugar más piezas y el Negro aun sí, el jugador Negro juega una pieza y es el ganador.

Si a su turno el jugador Blanco no puede jugar más piezas y el Negro tampoco, el resultado es empate.

Dado que el usuario puede tener piezas blancas o negras y que nuestro diagrama de estados no distingue esa condición, este es el momento donde sí se distingue. Esta tabla incluirá tres entradas y una salida:

- **Entradas:**
  - **Color de Pieza del Usuario**: **B** por Blancas y **N** por Negras)
  - **Usuario Puede Seguir Jugando**: **Sí**, indica que el usuario pudo jugar en este turno. **No**, lo contrario.
  - **Oponente Jugó**: **Sí**, indica que el oponente puede realizar al una jugada más. Al estar controlado el oponente, entonces realizará la movida. **No**, lo contrario.
- **Salida:**
  - **Situación: Indicará la situación en que queda la partida, a saber:**
    - **Puede Continuar (PC)**
    - **Finalizó (F)**
    - **Ultima Jugada del Usuario (UJU)**, que indicará que el oponente no puede seguir jugando y que el usuario tiene la oportunidad de ganar si coloca adecuadamente su pieza.
    - **Ultima Jugada del Oponente (UJO)**, que indicará que el oponente jugará nuevamente y finaliza la partida.

A continuación se observa la tabla:

Entradas \ Casos	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Color de Pieza del Usuario	B	B	B	B	N	N	N	N
Oponente Jugó	Sí	No	Sí	No	Sí	No	Sí	No
Usuario Puede Seguir Jugando	Sí	Sí	No	No	Sí	Sí	No	No
Salida: Situación	PC	F	UJO	F	PC	UJU	F	F

Tabla 6-3 - Decisiones en Final de Partida

- **Interfaz:**
  No aplicable.

- **Condiciones de fallo**:
  No debe alcanzarse nunca la situación indicada con X.

Finalizar Partida

- **Escenario**:
  En el estado Casillero Seleccionado.

- **Precondiciones**:
  Se invocó al caso de uso Chequear Final de Partida y el resultado ha sido "Terminó".

- **Postcondiciones**:
  Finaliza la partida. Se registra la situación y el resultado de la partida. Se pasa al Estado Final.

- **Interfaz**:
  Se muestra al usuario que finalizó la partida dando tres diferentes escenas en caso de haber ganado, empatado o perdido. Se muestra el botón de nueva partida.

- **Condiciones de fallo**:
  No aplicable.

Realizar Jugada del Oponente

- **Escenario**:
  Se accede desde el estado Casillero elegido.

- **Precondiciones**:
  El Oponente Puede Realizar una movida (Puede Jugar).

- **Postcondiciones**:
  Realiza la movida que el modelo le indica (pieza, casillero y orientación de pieza). Se registra.

- **Interfaz**:
  Se muestran sucesivamente los movimiento de seleccionar pieza, seleccionar casillero, rotar pieza (de ser necesario) y aceptar pieza.

- **Condiciones de fallo**:
  Al tener que esperar que la pieza quede en su posición de aceptada para cambiar de estado a Antes de Seleccionar Pieza, el sistema no aceptará las interacciones del usuario que ocurran durante dichos desplazamientos. Se registrarán esas interacciones anticipadas.

Pensar Jugada

- **Escenario**:
  Cuando se desee que el modelo sugiera una jugada a realizar.

- **Precondiciones**:
  Se indicará para qué jugador hay que pensar la jugada. Si existen restricciones, también se indicarán, a saber: Determinada Pieza (Devuelve el casillero y la orientación de la pieza solamente), Determinada Pieza en Determinado Casillero (devuelve la orientación de la pieza exclusivamente).

- **Postcondiciones**:
  Devuelve la pieza, el casillero y la orientación de la pieza, según se necesite, además de si es posible o no realizar la movida.

- **Interfaz**:
  No aplicable.

- **Condiciones de fallo**:
  No aplicable.

6.2.3 Análisis de Requisitos

Se procedió a la revisión completa de los casos de uso, se adecuaron las inconsistencias. El resultado final es el que se observa en 7.2.2

6.2.4 Validación de Requisitos

Se procedió a verificar con el investigador que los casos de usos fueran válidos, consistentes y completos.

6.3 Identificación de Subsistemas de Análisis

Se procede a analizar los beneficios que en este desarrollo se obtendrían de descomponer al sistema en diferentes subsistemas.

6.3.1 Determinación de Subsistemas de Análisis

Al analizarse los procesos del negocio, observamos que el sistema en cuestión, tiene una unicidad importante como para que el análisis se realice en conjunto.

El actor principal que es el **usuario**, será ayudado por su **tutor virtual** a tomar decisiones apropiadas. Las actividades que realiza y las que omite (pero debería realizar), son las que se registrarán para que luego, con el trazado obtenido el **investigador** pueda trabajar con estos datos.

El alcance del proyecto se limita hasta lo descripto. Si el sistema hubiera incluido la exploración de los datos obtenidos, las herramientas a desarrollar hubiesen constituido otro subsistema de análisis.

Por todo lo indicado, el sistema no se ha dividido en distintos subsistemas de análisis.

6.4 Análisis de los Casos de Uso

En esta actividad, se presentan las clases que permiten realizar cada uno de los casos de uso, de esa forma, luego se describen la interacción entre los objetos definidos.

6.4.1 Interacción entre Objetos e Identificación de Clases Asociadas

Esta tarea tiene como finalidad identificar las interacciones entre los objetos, de forma tal de describir cómo es el comportamiento en conjunto de los objetos de clases identificadas hasta el momento. Para ello se han usado diagramas de secuencia que clarifican la interacción entre los distintos objetos de las clases mencionadas.

Presentado el diagrama de Secuencia, se procede a listar las clases que dieron origen a los objetos del diagrama. Posteriormente se observan a algunas clases generales desde dos puntos de vista:

- Desde el punto de vista del modelo una pieza será la que podrá estar jugada, rotada, disponible.
- Desde el punto de vista de la interacción con el usuario, puede darse que ha sido seleccionada o se está desplazando o rotando.

Para ello, al incluirse ambas visiones, se incorporan nuevas clases que ayudarán a diferenciar lo expresado hasta aquí.

Previa a la presentación de las dos visiones se catalogan y clasifican las clases de acuerdo a:

- **Cases de Entidad**: representan la información manipulada en el caso de uso.
- **Clases de Interfaz de Usuario**: se utilizan para describir la interacción entre el sistema sus actores.
- **Clases de Control**: son responsables de la coordinación, secuencia de transacciones y control de los objetos relacionados con un caso de uso.

El detalle de la clasificación de todas las clases es el siguiente:

- **Controlador**: Clase de Control.
- **Tablero**: Clase de Entidad.
- **Jugador**: Clase de Entidad.

- **Pieza**: Clase de Entidad.
- **Pieza Visual**: Clase de Interfaz de Usuario.
- **Casillero**: Clase de Entidad.
- **Casillero Visual**: Clase de Interfaz de Usuario.
- **Lados**: Clase de Entidad.
- **Lado de Pieza Visual**: Clase de Interfaz de Usuario.
- **Lados de Pieza Visual**: Clase de Entidad.
- **Borde de Tablero Visual**: Clase de Interfaz de Usuario.
- **Bordes de Tablero Visual**: Clase de Entidad.
- **Casillero Visual de Posición Inicial**: Clase de Interfaz de Usuario.
- **Indicador de Rotación**: Clase de Interfaz de Usuario.
- **Indicador de Rotación hacia la derecha**: Clase de Interfaz de Usuario.
- **Indicador de Rotación hacia la izquierda**: Clase de Interfaz de Usuario.
- **Registrador de Eventos**: Clase de Entidad.
- **Manifestación de Error**: Clase de Interfaz de Usuario.
- **Botón de Reinicio**: Clase de Interfaz de Usuario.
- **Manifestación de Resultado**: Clase de Interfaz de Usuario
- **Casilleros**: Clase de Entidad.
- **Piezas**: Clase de Entidad.
- **Tablero Visual**: Clase de Interfaz de Usuario.

A continuación, se presentan los diagramas de secuencia para cada Caso de Uso, de acuerdo a la especificación dada previamente, en los cuales se incluyen notas aclaratorias cuando es requerido.

En los diagramas se utilizarán los nombres descriptivos de clases utilizados hasta el momento. Cuando se definan las clases del sistema en el proceso de diseño, se adoptarán nombres adecuados al lenguaje de programación y se hará en cada caso la correspondencia con los nombres de clases utilizados en el presente proceso de análisis.

Inicia Partida (Ilustración 6-4)

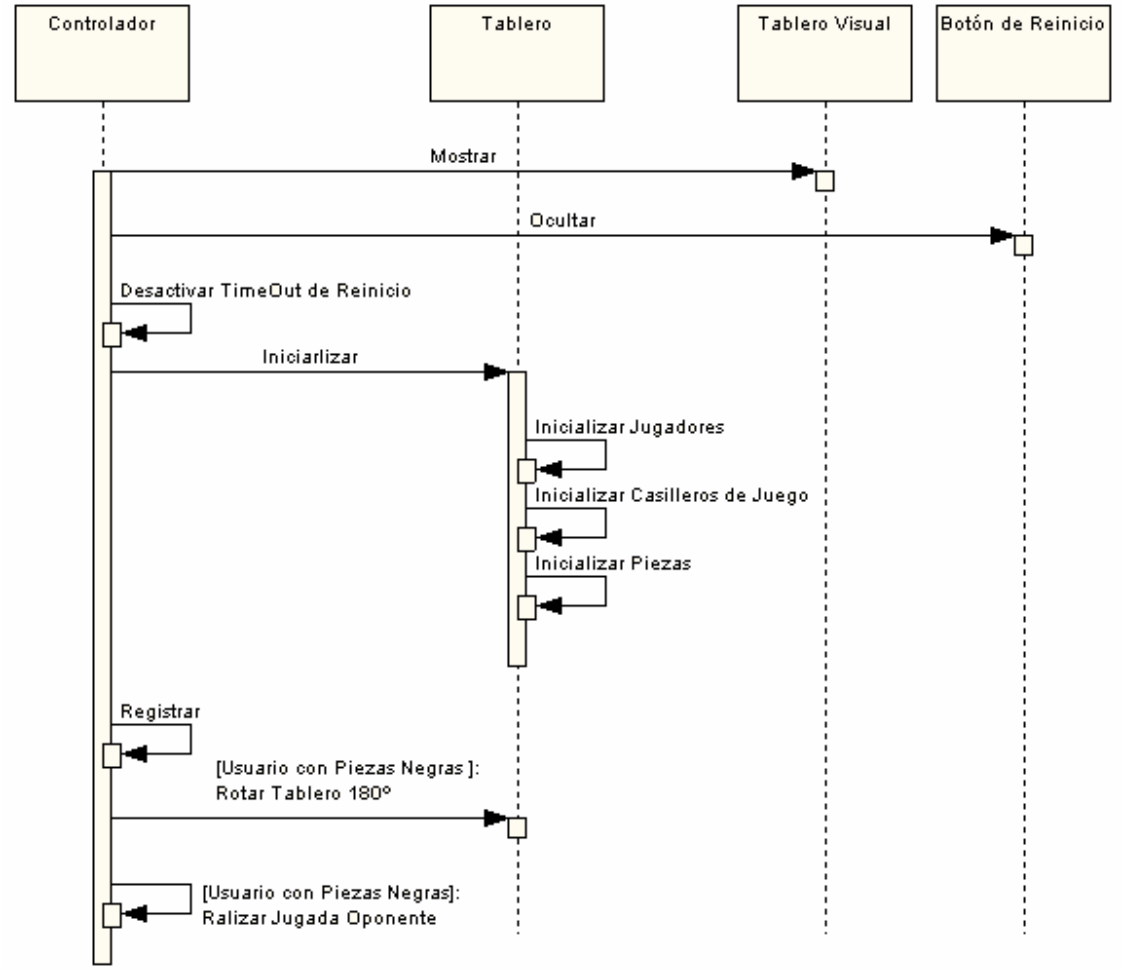


Ilustración 6-4 - Inicia Partida

Clases Involucradas: Controlador – Tablero – Tablero Visual - Botón de Reinicio – Registrador de Eventos

A continuación pueden observarse los diagramas correspondientes a Inicializar Jugadores (Ilustración 6-5), Inicializar Casilleros de Juego (Ilustración 6-6) y finalmente, Inicializar Piezas (Ilustración 6-7) que extienden el presente caso de uso.

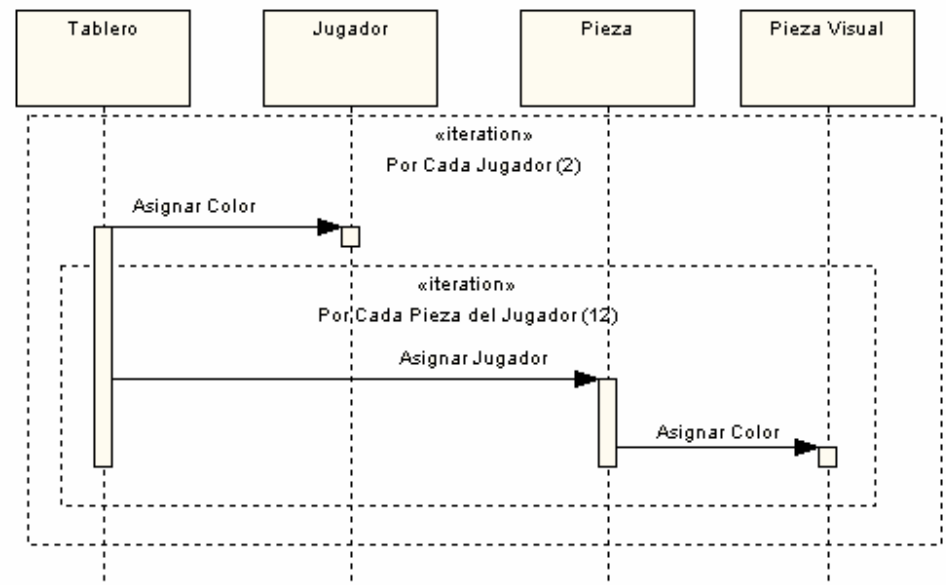


Ilustración 6-5 - Inicializar Jugadores

Clases Involucradas: Tablero – Jugador - Pieza – Pieza Visual

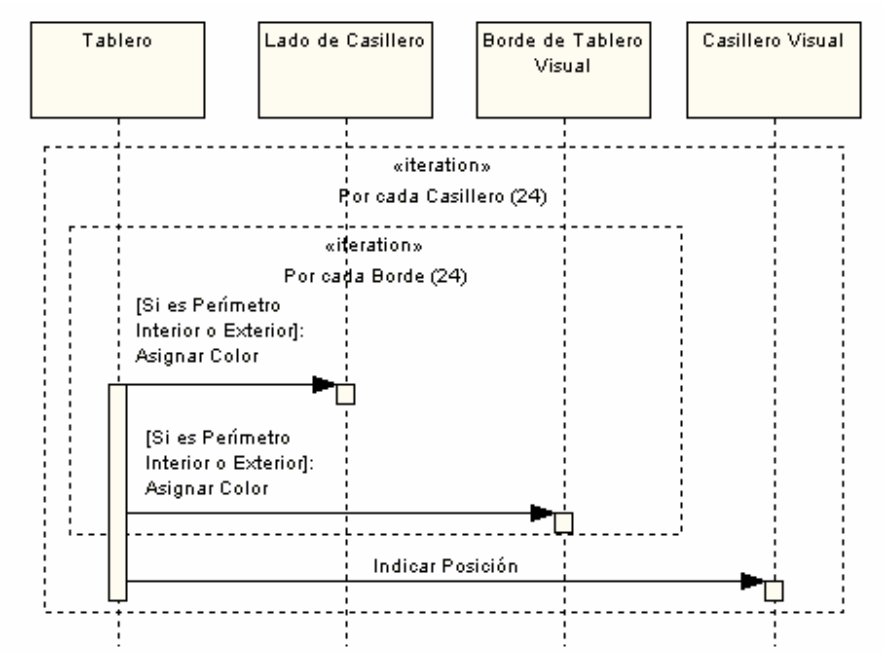


Ilustración 6-6 - Inicializar Casilleros de Juego

Clases Involucradas: Tablero – Lado de Casillero – Borde de Tablero Visual – Casillero Visual

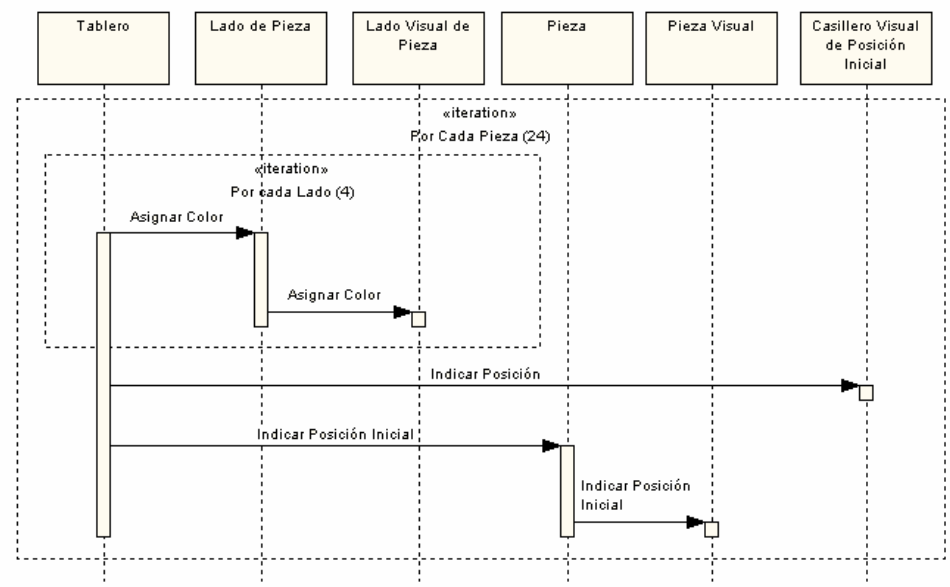


Ilustración 6-7 - Inicializar Piezas

Clases Involucradas: Tablero – Lado de Pieza – Lado de Pieza Visual - Pieza – Pieza Visual – Casillero Visual de Posición Inicial

Selecciona Pieza (Ilustración 6-8)

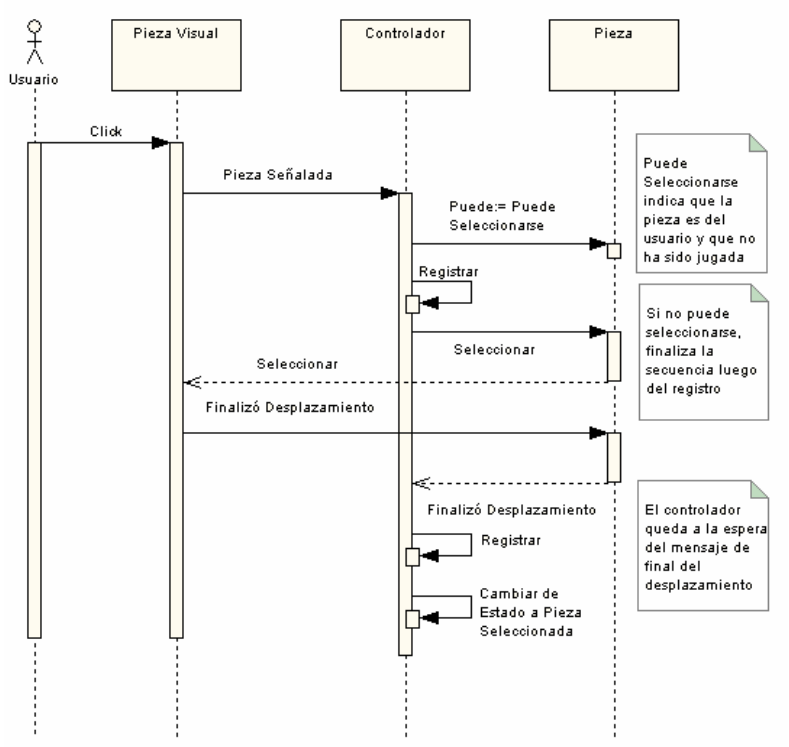


Ilustración 6-8 - Selecciona Pieza

Clases Involucradas: Pieza Visual - Controlador – Pieza – Registrador de Eventos

Selecciona Casillero (Ilustración 6-9)

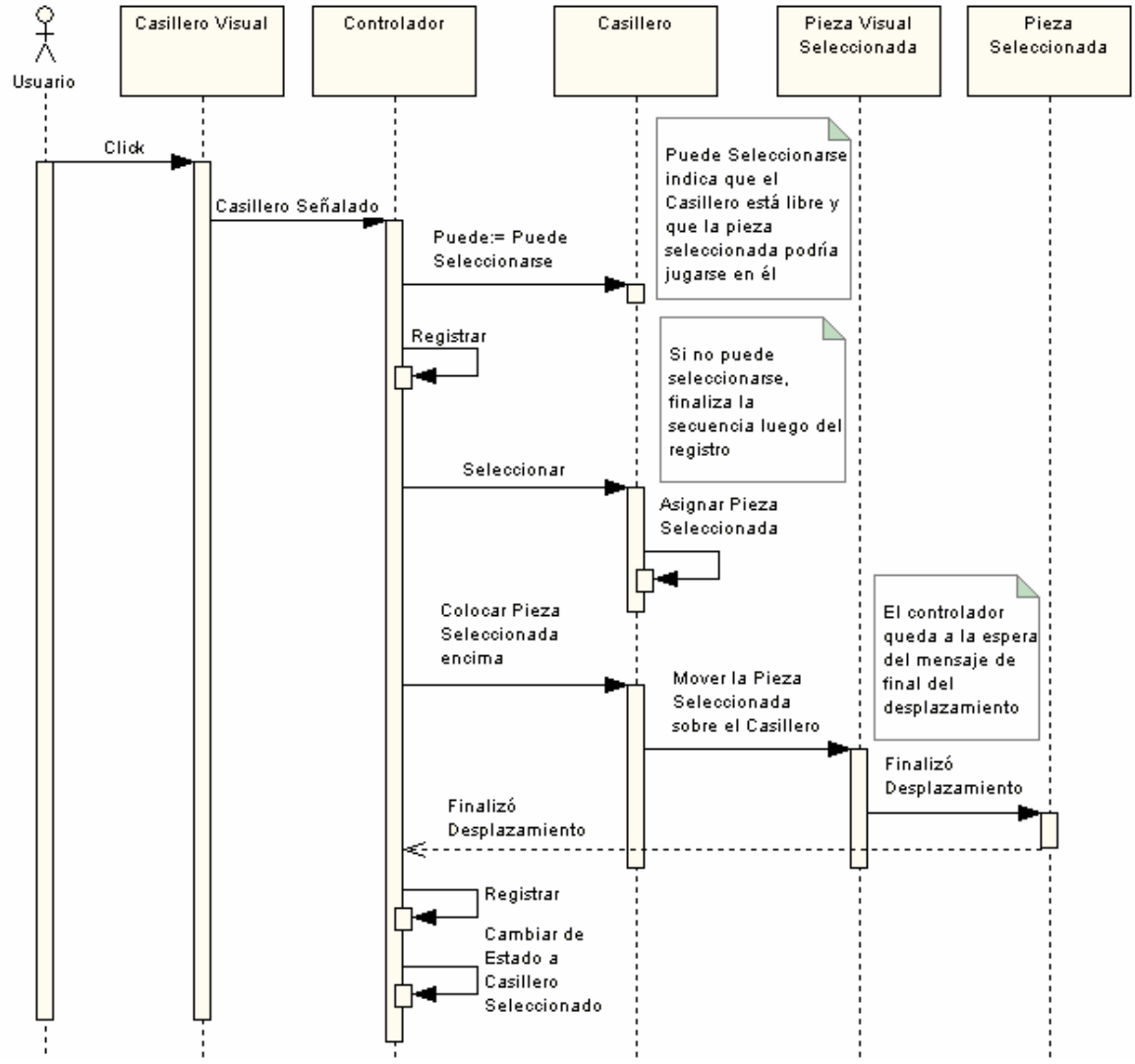


Ilustración 6-9 - Selecciona Casillero

Clases Involucradas: Casillero Visual - Controlador – Casillero – Pieza – Pieza Visual – Registrador de Eventos

Selecciona Orientación de Pieza

Se representan cada Caso de Uso dentro de la jerarquía en Rota Pieza 90º hacia la Derecha y Rota Pieza 90º hacia la Izquierda.

Rota Pieza 90º hacia la Derecha (Ilustración 6-10)

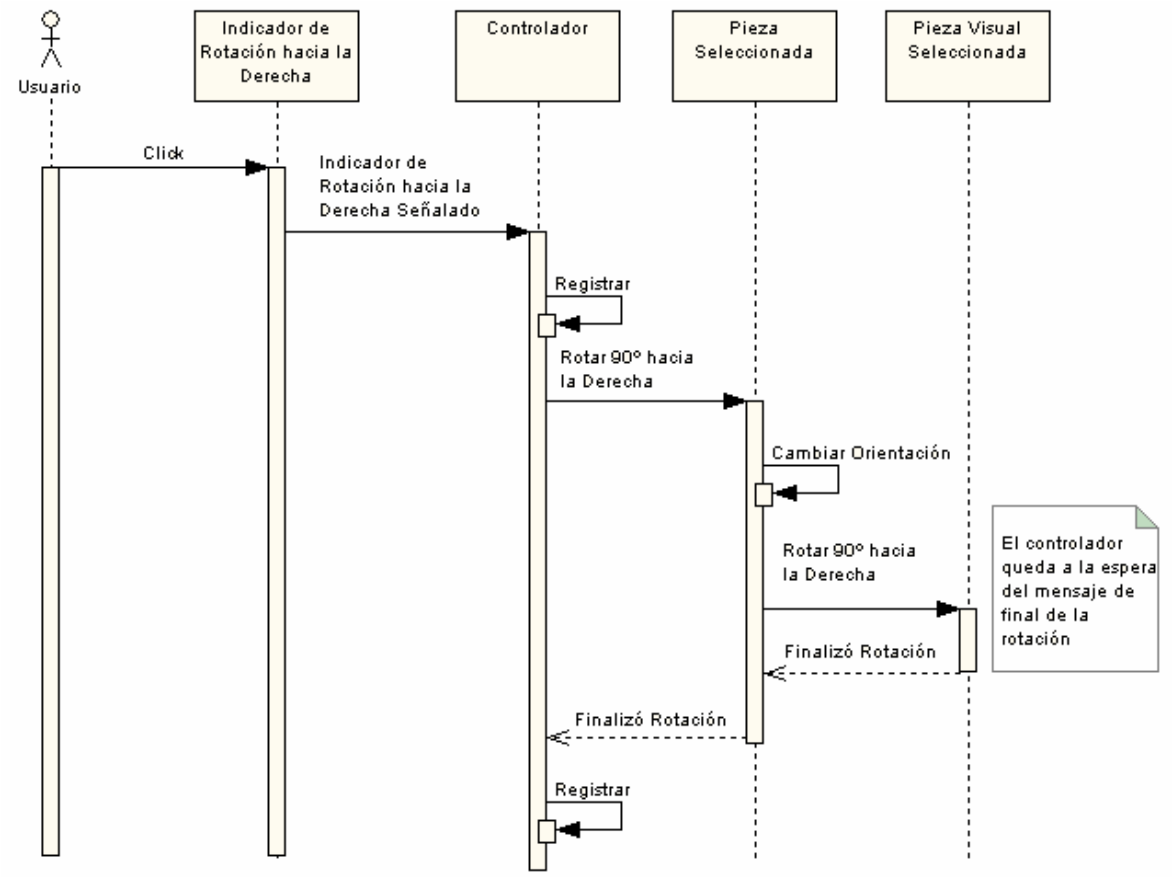


Ilustración 6-10 - Rota Pieza 90º hacia la Derecha

Clases Involucradas: Indicador de rotación hacia la derecha - Controlador – Pieza - Pieza Visual - Registrador de Eventos

Rota Pieza 90º hacia la Izquierda (Ilustración 6-11)

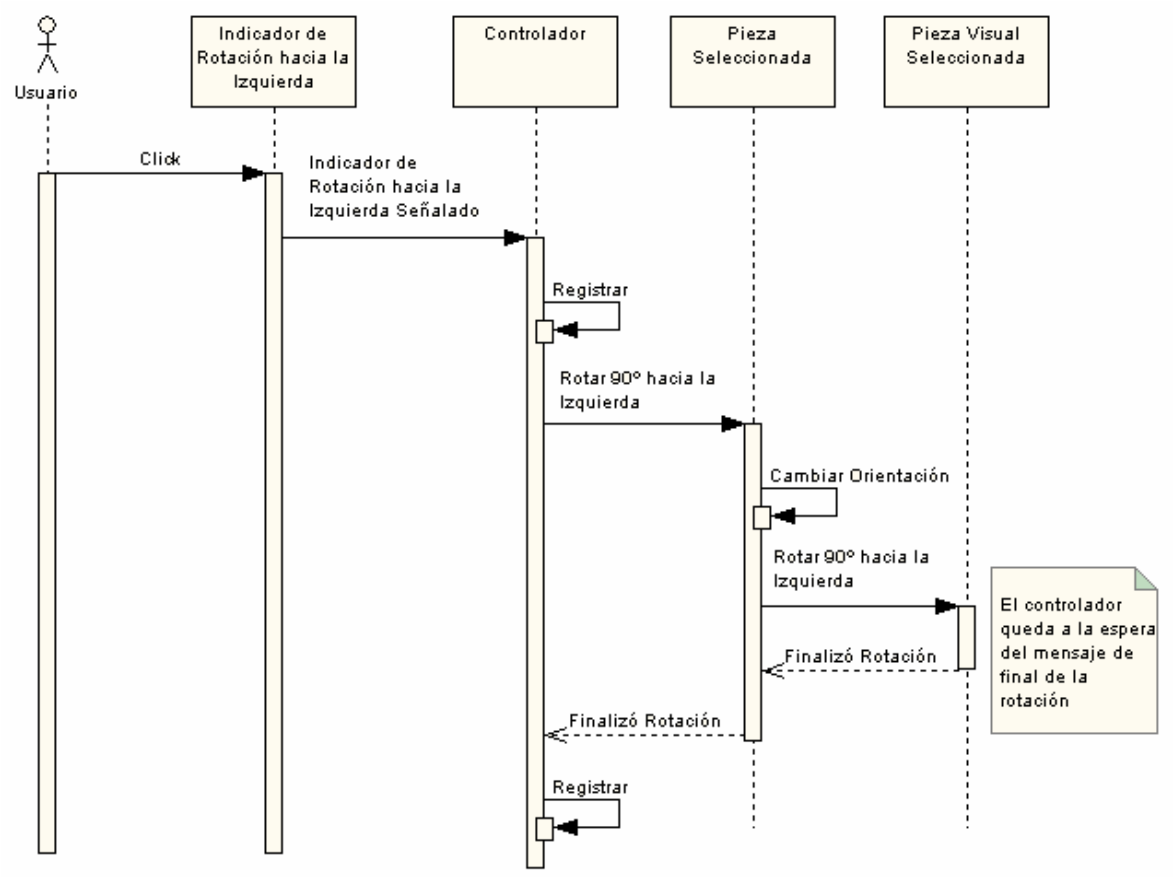


Ilustración 6-11 - Rota Pieza 90º hacia la Izquierda

Clases Involucradas:
Indicador de rotación hacia la izquierda - Controlador – Pieza – Pieza Visual – Registrador de Eventos

Acepta Pieza (Ilustración 6-12)

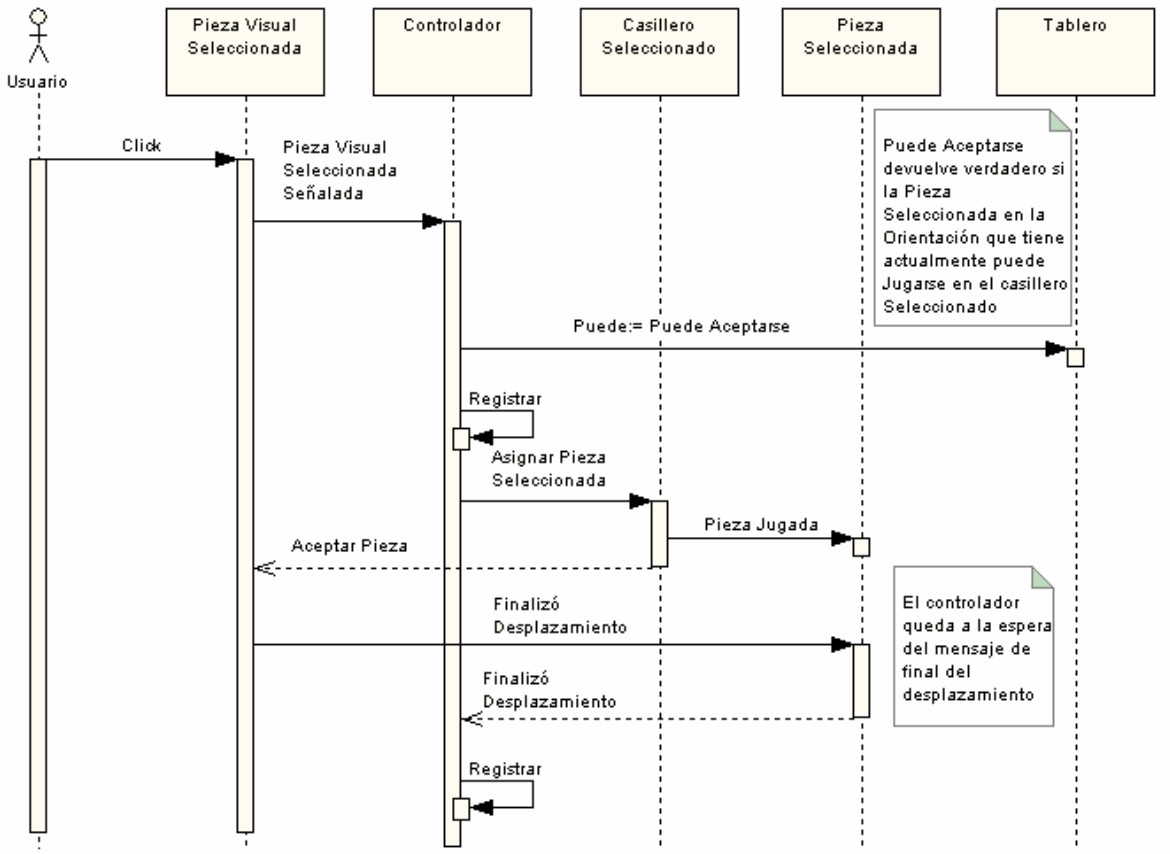


Ilustración 6-12 - Acepta Pieza

Clases Involucradas:
Pieza Visual - Controlador – Casillero – Pieza - Tablero – Registrador de Eventos

Cancela Pieza Seleccionada (Ilustración 6-13)

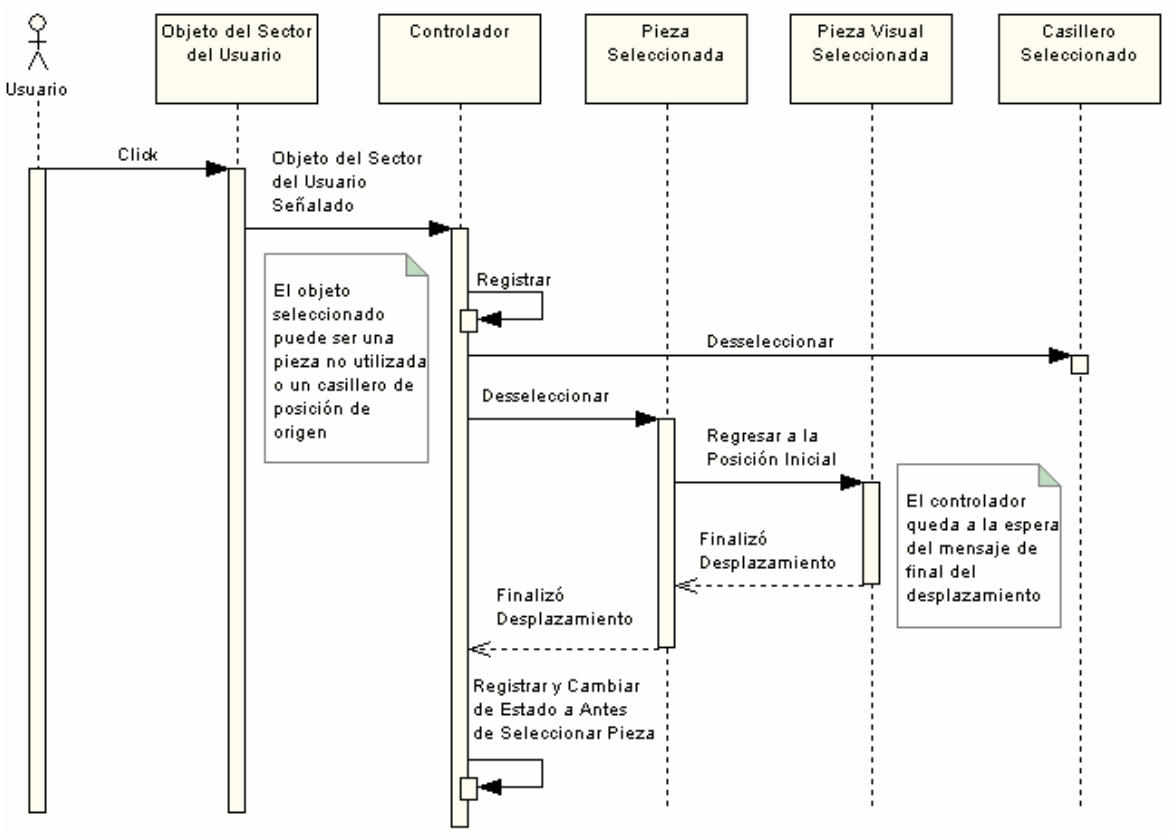


Ilustración 6-13 - Cancela Pieza Seleccionada

Clases Involucradas:
Pieza Visual - Casillero Visual de Posición Inicial - Controlador – Pieza **–** Casillero – Registrador de Eventos

Selecciona Objeto Inapropiado (Ilustración 6-14)

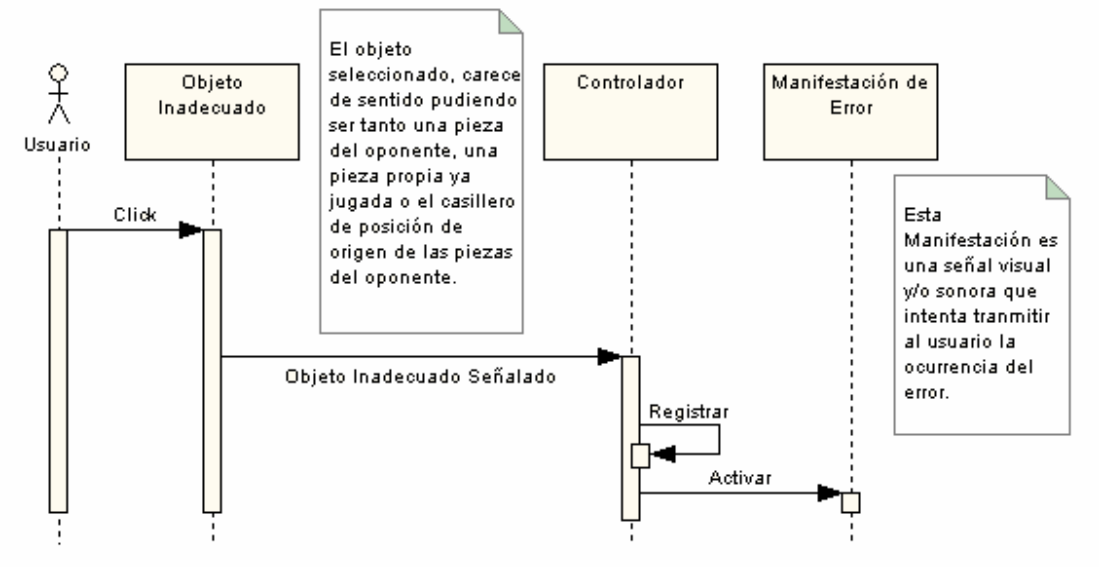


Ilustración 6-14 - Selecciona Objeto Inapropiado

Clases Involucradas:
Controlador – Pieza **-** Casillero Visual de Posición Inicial – Manifestación de Error – Registrador de Eventos

Asiste al Jugador

Asiste al Jugador es un Caso de Uso genérico. Los diagramas que se incluyen, representan las secuencias de acuerdo al estado de la jugada y la condición de "jugable" que tiene la pieza elegida en el casillero elegido, es decir, si se puede o no aceptar la pieza sin necesidad de rotarla.

Asiste al Jugador (Antes de Seleccionar la Pieza) (Ilustración 6-15)

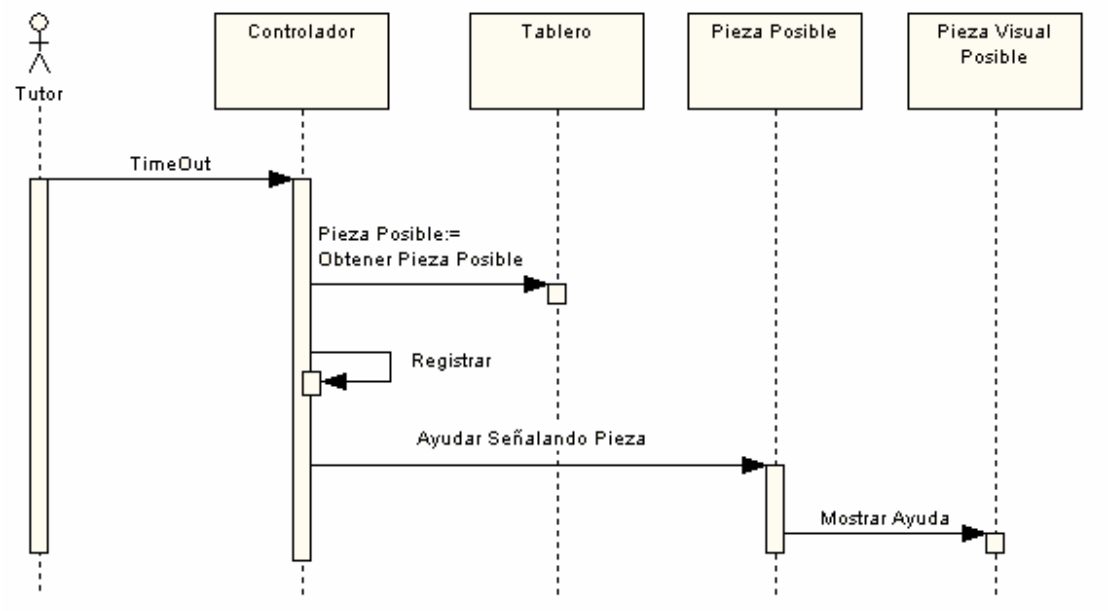


Ilustración 6-15 - Asiste al Jugador (Antes de Seleccionar la Pieza)

Clases Involucradas:
Controlador – Tablero – Pieza – Pieza Visual – Registrador de Eventos

Asiste al Jugador (Pieza Seleccionada) (Ilustración 6-16)

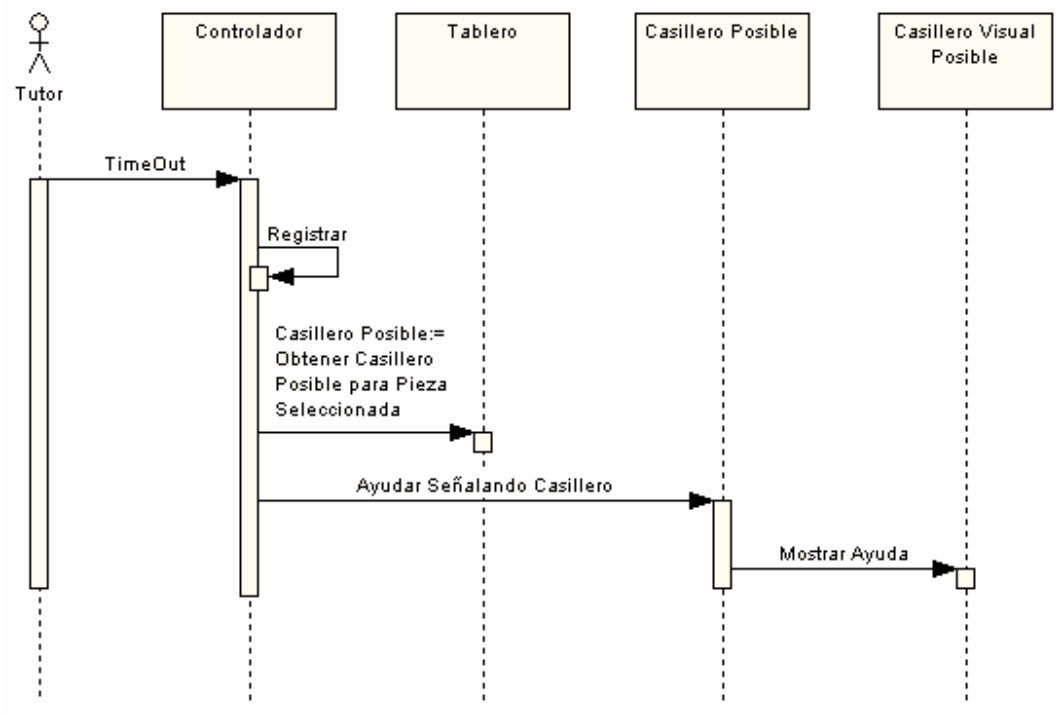


Ilustración 6-16 - Asiste al Jugador (Pieza Seleccionada)

Clases Involucradas:
Controlador – Tablero - Casillero – Casillero Visual – Registrador de Eventos

Asiste al Jugador (Casillero Seleccionado y la pieza puede jugarse sin rotar) (Ilustración 6-17)

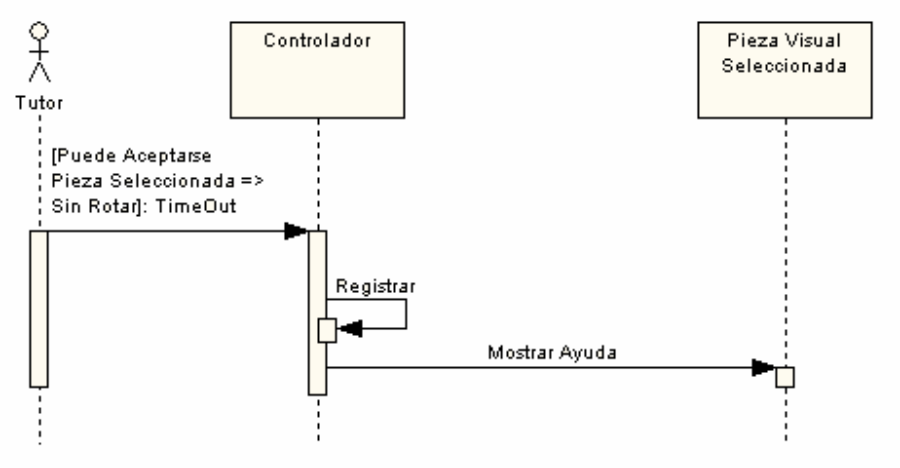


Ilustración 6-17 - Asiste al Jugador (Casillero Seleccionado y la pieza puede jugarse sin rotar)

Clases Involucradas:
Controlador – Pieza – Casillero – Tablero – Registrador de Eventos

Asiste al Jugador (Casillero Seleccionado y la pieza no puede jugarse si no se la rota) (Ilustración 6-18)

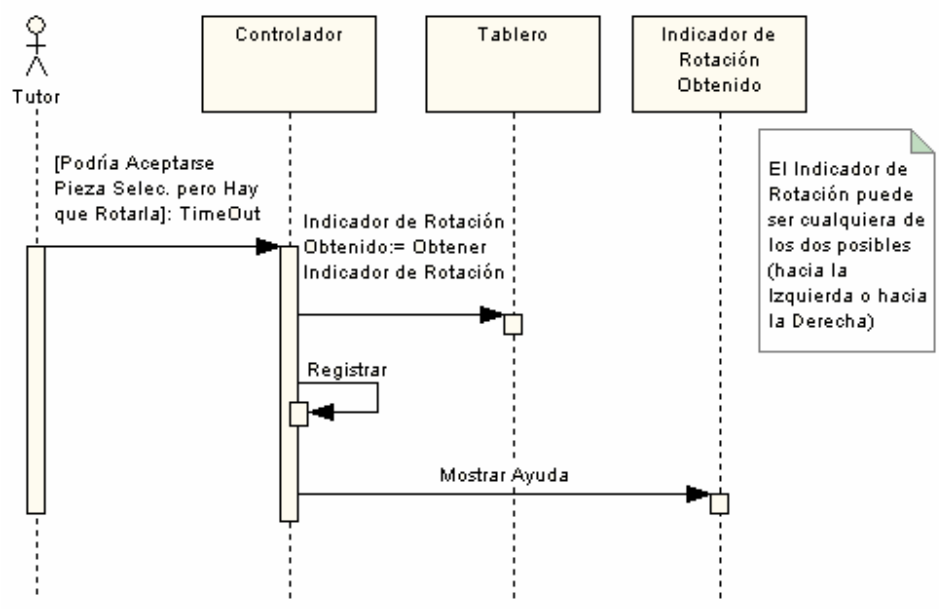


Ilustración 6-18 - Asiste al Jugador (Casillero Seleccionado y la pieza no puede jugarse si no se la rota)

Clases Involucradas:
Controlador – Pieza – Casillero – Tablero – Indicador de Rotación – Registrador de Eventos

Obtiene Trazado (Ilustración 6-19)

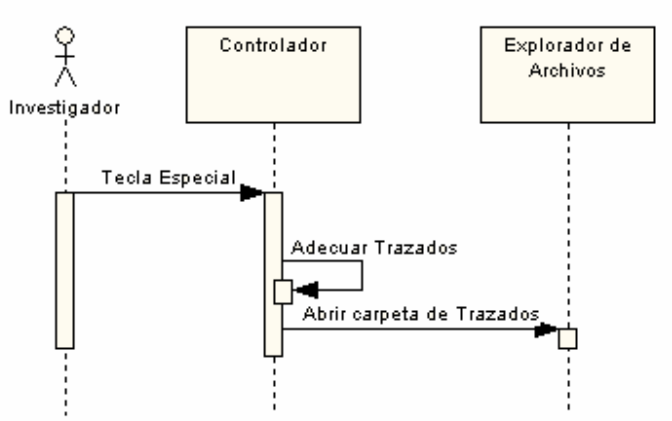


Ilustración 6-19 - Obtiene Trazado

Clases Involucradas:
Controlador – Registrador de Eventos- Explorador de Archivos (clase externa del Sistema Operativo)

Chequear Final de Partida (Ilustración 6-20)

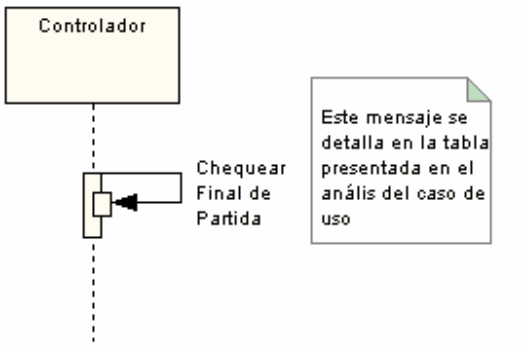


Ilustración 6-20 - Chequear Final de Partida

Clases Involucradas:
Controlador – Tablero

Finalizar Partida (Ilustración 6-21)

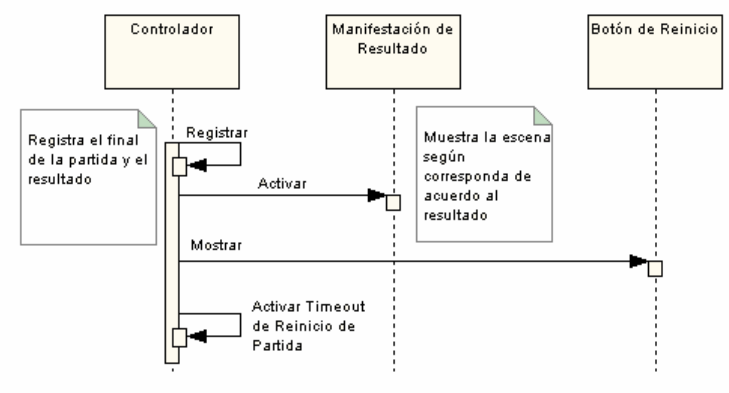


Ilustración 6-21 - Finalizar Partida

Clases Involucradas:
Controlador – Manifestación de Resultado – Botón de Reinicio – Registrador de Eventos

Realizar Jugada del Oponente (Ilustración 6-22)

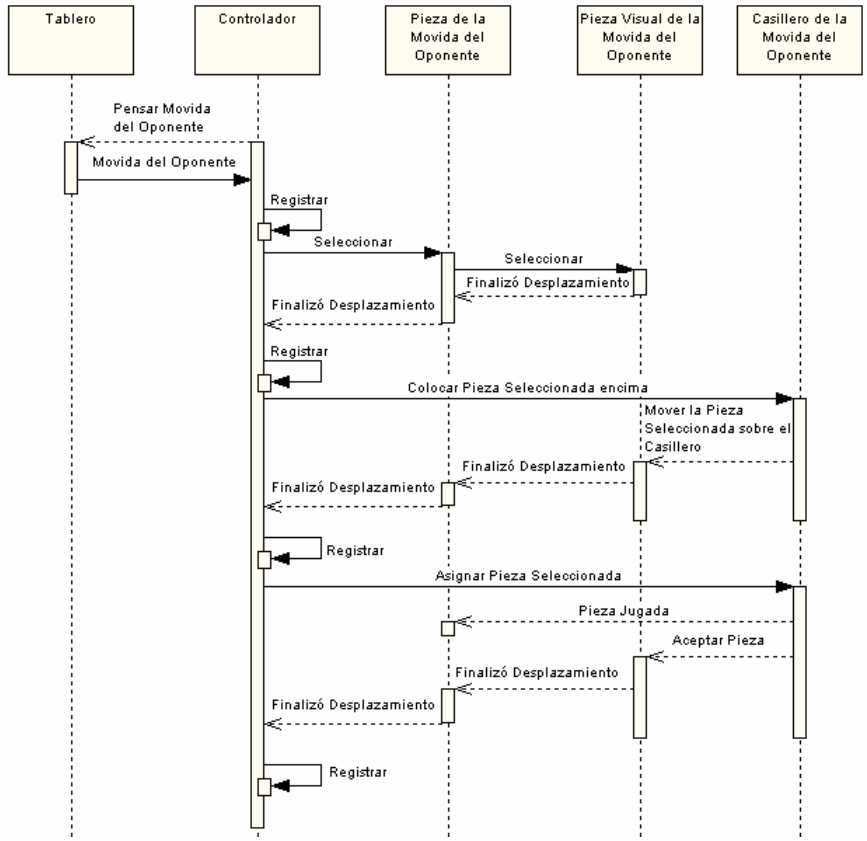


Ilustración 6-22 - Realizar Jugada del Oponente

Clases Involucradas:
Controlador – Tablero – Pieza – Pieza Visual - Casillero – Registrador de Eventos

Pensar Jugada (Ilustración 6-23)

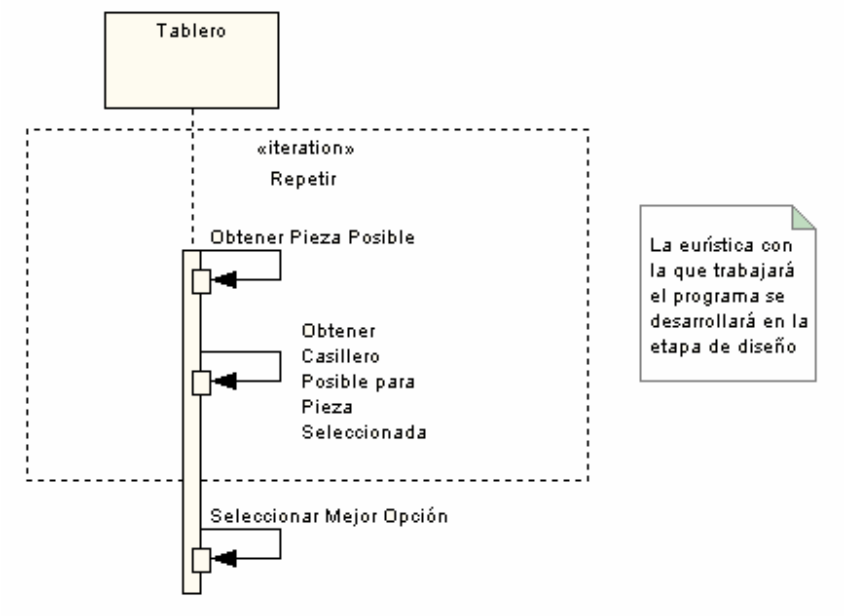


Ilustración 6-23 - Pensar Jugada

Clases Involucradas:
Tablero (el resto de las clases involucradas dependerá de la heurística con la que trabajará el programa y que se desarrollará en la etapa de diseño).

6.5 Análisis de Clases

Durante la presente actividad, se van a analizar las clases que hasta el momento han sido referidas, sus responsabilidades, sus atributos, sus relaciones entre sí.

6.5.1 Identificación de Responsabilidades y Atributos

En cuanto a las responsabilidades y atributos de cada clase:

- Las responsabilidades se obtienen de la participación de la clase en los distintos casos de uso e incluirán las operaciones de la clase.
- Los atributos especifican las propiedades de la clase que aparecen dentro de sus responsabilidades.

La definición se hará desde las clases más simples a las clases más complejas, es decir, desde las clases visuales a las demás.

Se incluyen atributos y operaciones relativas a la posición de los objetos, que en su mayoría no fueron referenciadas previamente en el análisis. Asimismo, se agregan atributos que identifican a cada objeto de la clase en caso de necesitarse.

1. **Casillero Visual de Posición Inicial:**

   Responsabilidades:

   Existe una por cada Pieza Visual y tendrá una posición en el tablero. Visualmente, la Pieza Visual se verá "apoyada" sobre su Casillero Visual de Posición Inicial hasta el momento que se seleccione. Si recibe un click del usuario, deberá transmitirlo al controlador.

   Atributos:

   - **Color**: Será acorde al color de la pieza.
   - **Posición**: Coordenadas de la posición inicial de la pieza.

   Operaciones:

   - Establecer Color
   - Establecer Posición
   - Transmitir Interacción

2. **Manifestación de Error:**

   Responsabilidades:

   Es la señal visual / sonora que advertirá al usuario que está cometiendo un error.

   Atributos:

   - **Sonido**: Indicación sonora.
   - **Escena**: Escena a mostrar. (Se terminará de definir en el diseño).

   Operaciones:

   - Activar

3. **Manifestación de Resultado:**

   Responsabilidades:

   Es la señal visual / sonora que advertirá al usuario que está cometiendo un error.

Atributos:

- **Sonido**: Indicación sonora.
- **Escena**: Escena a mostrar. (Se terminará de definir en el diseño).

Operaciones:

- Activar

4. Casillero Visual:

Responsabilidades:

Será cada uno de los 24 casilleros donde puede hacer click el usuario cuando quiere desplazar la pieza seleccionada. Una vez aceptada la pieza seleccionada, la Pieza Visual se verá "apoyada" sobre su Casillero Visual. Si recibe un click del usuario, deberá transmitirlo al controlador.

Atributos:

- **Color**: Corresponderá al color asignado por diseño.
- **Posición**: Coordenadas de la posición actual.

Operaciones:

- Establecer Color
- Establecer Posición
- Transmitir Interacción
- Mostrar Ayuda

5. Pieza Visual:

Responsabilidades:

Será cada una de las 24 piezas que verá el usuario, la cual se desplazará o rotará de acuerdo a las acciones que vaya tomando el usuario mismo o el oponente a través de la realización de la jugada pensada por el programa. Si recibe un click del usuario, deberá transmitirlo al Controlador.

Atributos:

- **Color**: Será acorde al jugador asignado.
- **Posición Inicia**l: Corresponde a la del Casillero Visual de Posición Inicial (Coordenadas)

- **Posición Actual**: Corresponde a las coordenadas de la posición que ocupa la pieza visual en un momento dado.

Operaciones:

- Establecer Color
- Establecer Posición
- Rotar 90º hacia Izquierda o Derecha
- Transmitir Interacción
- Mostrar Ayuda

6. Botón de Reinicio:

Responsabilidades:

Transmitir Interacción al Controlador a fin de iniciar una nueva partida. Sólo disponible al momento de finalizar una partida y hasta antes de iniciar la próxima.

Atributos:

- **Visible**: Define si el usuario puede ver o no a dicho botón.

Operaciones:

- Mostrar / Ocultar (Establecer Visible).
- Transmitir Interacción

7. Borde de Tablero Visual:

Responsabilidades:

Solamente brinda el marco visual que necesita el usuario para saber que el lado de la pieza que va a colocar en forma contigua, debe respetar igual color.

Atributos:

- **Color**: Color del borde
- **Posición**: Coordenadas
- **Orientación**: Hacia donde se colocará visualmente el borde.

Operaciones:

- Establecer Color
- Establecer Posición

8. Bordes de Tablero Visual:

Responsabilidades:

Es el contenedor de cada uno de los 24 Borde de Tablero Visual.

Atributos:

No están previstos.

Operaciones:

No están previstas.

9. Indicador de rotación:

Responsabilidades:

Será una clase abstracta. Transmitirá la interacción con el usuario al Controlador a fin de realizar la rotación de la pieza en el sentido indicado. Sólo disponible en el estado "Casillero Elegido". Su posición será la de la pieza seleccionada que en dicho estado coincidirá con el del casillero seleccionado.

Atributos:

- **Posición**: Corresponde a las coordenadas de la posición del indicador de rotación. La determinación del tipo de posición (relativa o absoluta), se definirá durante el diseño.
- **Visible**: Define si el usuario puede ver o no a dicho botón.

Operaciones:

- Establecer Posición.
- Mostrar / Ocultar (Establecer Visible).
- Mostrar Ayuda

10. Indicador de rotación hacia la derecha:

Responsabilidades:

Será la clase heredada de Indicador de Rotación que transmitirá la intención del usuario de rotar la pieza seleccionada en el sentido de las agujas del reloj.

Atributos:

- **Posición**: Corresponde a las coordenadas de la posición del indicador de rotación. La determinación del tipo de posición (relativa o absoluta), se definirá durante el diseño.
- **Visible**: Define si el usuario puede ver o no a dicho botón.

Operaciones:

- Establecer Posición.
- Mostrar / Ocultar (Establecer Visible).
- Mostrar Ayuda

11. Indicador de rotación hacia la izquierda:

Responsabilidades:

Será la clase heredada de *Indicador de Rotación* que transmitirá la intención del usuario de rotar la pieza seleccionada en el sentido contrario al de las agujas del reloj.

Atributos:

- **Posición**: Corresponde a las coordenadas de la posición del indicador de rotación. La determinación del tipo de posición (relativa o absoluta), se definirá durante el diseño.
- **Visible**: Define si el usuario puede ver o no a dicho botón.

Operaciones:

- Establecer Posición.
- Mostrar / Ocultar (Establecer Visible).
- Mostrar Ayuda

12. Lados:

Responsabilidades:

Es el contenedor de la información del color de lado que tiene asignada una pieza o un casillero.

Atributos:

- **Arriba**: Color del lado de arriba
- **Derecha**: Color del lado de la derecha

- **Abajo**: Color del lado de abajo
- **Izquierda**: Color del lado de la izquierda.

Operaciones:

- Establecer Arriba
- Establecer Derecha
- Establecer Abajo
- Establecer Izquierda

13. Lados de Pieza Visual:

Responsabilidades:

Clase que hereda de Lados y que agrega los respectivos lados visuales que contendrá la pieza.

Atributos Agregados:

- **Arriba Visual**: Lado de Pieza Visual correspondiente a la posición de arriba de una pieza en su orientación actual.
- **Derecha Visual**: Lado de Pieza Visual correspondiente a la posición de la derecha en su orientación actual.
- **Abajo Visual**: Lado de Pieza Visual correspondiente a la posición de abajo de una pieza en su orientación actual.
- **Izquierda Visual**: Lado de Pieza Visual correspondiente a la posición de la izquierda de una pieza en su orientación actual.

Operaciones:

No tiene operaciones previstas.

14. Pieza:

Responsabilidades:

La pieza será la abstracción que para el modelo del juego tendrá cada una de las piezas que lo componen. Deberá conocer su estado (si ha sido utilizada o no), su orientación actual, seleccionar, desplazar hasta un casillero, saber rotar hacia uno u otro lado, aceptar la pieza en la posición y orientación actual y que todo ello se vea reflejado en su respectiva Pieza Visual.

Atributos:

- **Identificador**: Identifica unívocamente a la pieza.
- **Lados Iniciales**: Lados tal como se ven en la posición Inicial.
- **Lados**: Lados Visuales actuales. Tal como se ven luego de sucesivas rotaciones.
- **Pieza Visual**: Pieza Visual asociada.
- **Usada**: Indicará si ha sido aceptada en algún casillero de juego.
- **Seleccionada**: Indicará si ha sido seleccionada.
- **Orientación**: La que corresponde a alguna de las siguientes: Inicial, Rotada hacia la derecha una vez, Rotada hacia la derecha dos veces o Rotada hacia la derecha tres veces.
- **Jugador**: A qué jugador pertenece la pieza.
- **Casillero Visual de Posición Inicial**: Sobre qué Casillero Visual tomará su posición inicial.

Operaciones:

- Rotar 90º hacia la Izquierda
- Rotar 90º hacia la Derecha
- Obtener Orientación
- Establecer Jugador Dueño
- Establecer Identificador
- Establecer Seleccionada (Seleccionar / "DesSeleccionar")
- Mostrar Ayuda
- Establecer Casillero Visual de Posición Inicial
- Puede Seleccionarse
- Establecer Seleccionada
- Establecer Usada

15. Piezas:

Responsabilidades:

Es el contenedor de las 24 piezas.

Atributos:

No están previstas.

Operaciones:

No están previstas.

16. Casillero:

Responsabilidades:

Es la abstracción que para el modelo del juego tiene cada una de los casilleros posibles dónde se colocan las piezas. Deberá conocer su estado (si ha sido utilizado o no), saber si puede jugarse una pieza dada o si podría jugarse previa rotación como así también, aceptar una pieza.

Atributos:

- **Identificador**: Identifica unívocamente al casillero.
- **Lados**: Lados que inicialmente contendrán la información del los bordes perimetrales según la posición dentro del tablero y que luego se irán completando a medida que las piezas se vayan colocando.
- **Casillero Visual**: Casillero Visual asociado.
- **Pieza**: Pieza que ha sido aceptada en él.
- **Usado**: Indicará si alguna pieza ha sido aceptada en él.
- **Seleccionado**: Indica si ha sido seleccionado.

Operaciones

- Establecer Identificador
- Puede Aceptarse Pieza
- Podría Aceptarse Pieza (previa rotación si fuera necesaria)
- Establecer Pieza
- Rotar hacia la Derecha

- Rotar hacia la Izquierda
- Mostrar Ayuda

17. Casilleros:

Responsabilidades:

Es el contenedor de los 24 casilleros.

Atributos:

No están previstas.

Operaciones:

No están previstas.

18. Jugador:

Responsabilidades:

Es uno de los dos participantes de la partida. Se estableció que el Jugador 1 es el que llevará las piezas blancas y el 2 las negras. Permitirá establecer un modo de juego que en el caso del oponente definirá cómo éste "pensará" su movida y en el caso del usuario, cómo el tutor lo asistirá.

Atributos:

- **Identificador**: 1 para el Jugador 1 y 2 para el otro.
- **Es el Usuario**: Sirve para identificar si el Jugador es el Usuario o No
- **Modo de Juego**: Determina la forma de "pensar" la jugada.
- **Puntos**: Indicará la cantidad de piezas colocadas al momento.

Operaciones:

- Establecer Jugador del Usuario
- Establecer Modo de Juego
- Establecer Puntos

19. Registrador de Eventos:

Responsabilidades:

Se encargará de almacenar en un archivo el registro de la información que el controlador va estableciendo en cada momento.

Atributos:

- **Nombre del Archivo**: Path y nombre completos del archivo donde será almacenada la información del trazado de la partida.
- **Datos a Registrar**: Contiene información a ser adicionada al archivo.

Operaciones:

- **Registrar**

20. **Tablero** (Podría habérselo llamado Tablero Inteligente o alguna denominación similar, que responda más acabadamente a la capacidad de incluir las funciones que permiten hacer respetar reglas de juego y pensar jugada. Se dejó Tablero por simplicidad, pero vale la pena remarcar la presente aclaración).

Responsabilidades:

Será el responsable de hacer cumplir las reglas del juego. Para ello, contendrá la información necesaria que le permita realizar dicha tarea desde la inicialización de cada uno de los objetos pasando por el mantenimiento del estado de cada uno de ellos cuando corresponda, hasta brindar información sobre posibles movimientos.

Atributos:

- **Piezas:** El conjunto de piezas del juego
- **Casilleros:** El conjunto de casilleros del juego
- **Jugador 1:** El jugador de Piezas Blancas.
- **Jugador 2:** El jugador de Piezas Negras.
- **Bordes de Tablero Visual:** El conjunto de los Bordes del tablero.
- **Orientación del Tablero Visual:** Según el usuario esté jugando con piezas blancas o negras.
- **Indicador de Orientación hacia la Derecha:** Referencia al Indicador de Rotación hacia la Derecha
- **Indicador de Orientación hacia la Izquierda:** Referencia al Indicador de Rotación hacia la Izquierda

Operaciones:

- Inicializar Tablero
- Inicializar Jugadores
- Inicializar Casilleros
- Inicializar Piezas
- Obtener Casillero Posible para Pieza Seleccionada
- Obtener Indicador de Rotación
- Obtener Casillero Posible para Pieza Seleccionada
- Obtener Pieza Posible
- Es Final de Partida
- Sugerir Movida para Jugador
- Puede Aceptarse (Una pieza dada con una orientación dada en un casillero dado)
- Mostrar / Ocultar

21. Tablero Visual

Responsabilidades:

Esta es la clase visual, cuya única instancia será el contenedor de todos las demás clases visuales. Esto facilitará por ejemplo que al cambiar el color de piezas asignado al jugador, exclusivamente haya que rotar el tablero 180º.

Atributos:

- **Orientación**: De acuerdo a si el usuario juega con piezas blancas o negras.
- **Visible**: Define si el tablero puede verse en la pantalla.

Operaciones:

- Establecer Orientación.
- Mostrar / Ocultar (Establecer Visible).

22. Controlador:

Responsabilidades:

Esta es la clase que asume todas las responsabilidades del control de la partida y la ayuda. Para ello deberá actuar en diferentes situaciones de acuerdo al estado actual y al objeto visual que el usuario ha indicado con su click del ratón. Dadas las características del actor Tutor que se incorporado y cuya única responsabilidad asignada es la de informar que llegó el momento de ayudar al usuario y que además esa ayuda depende del estado actual, la clase Controlador, incorpora la operación de generar el TimeOut que luego usará en consecuencia.

Atributos:

- **Tablero**: Será el tablero asociado.
- **Estado Actual**: Indicará el estado en el que se encuentra la partida.
- **Objeto Señalado**: Será el objeto que indicado por el usuario define la acción a seguir de acuerdo al estado actual
- **Timer**: Permitirá configurar el TimeOut para el presente estado.
- **Jugador**: Será el jugador que ha sido asignado al usuario (Jugador 1 con piezas blancas o Jugador 2 con piezas negras).
- **Ultima Jugada del Usuario**: Establecida en chequear final de partida a partir de la tabla presentada.
- **Timer de Reinicio**: Permite habilitar el reinicio de partida automático.
- **Pieza Seleccionada**: Referencia a la Pieza elegida por el usuario
- **Casillero Seleccionado:** Referencia al Casillero elegido por el usuario.

Operaciones:

- Iniciar Partida
- Obtener Trazado
- Realizar Jugada del Oponente
- Seleccionar Pieza
- Seleccionar Casillero
- Seleccionar Objeto Inadecuado
- Cancelar Pieza

- Establecer TimeOut
- Habilitar/Deshabilitar Timer
- Chequear Final de Partida
- Registrar evento
- Establecer TimeOut de Reinicio
- Habilitar/Deshabilitar TimeOut de Reinicio
- Cambiar de Estado a Pieza Seleccionada
- Finalizó Desplazamiento
- Finalizó Rotación
- Objeto ha sido señalado
- Tecla ha sido indicada
- Establecer Casillero Seleccionado
- Establecer Pieza Seleccionada
- Establecer Estado
- Atender TimeOut
- Adecuar Trazados
- Obtener Movida del Oponente

6.5.2 Identificación de Asociaciones y Agregaciones

A partir de los mensajes establecidos en los diagramas de secuencias, se presentan a continuación las Asociaciones y Agregaciones identificadas. Esta información será utilizada en la definición de clases de diseño (Ítem 7.3), aclarando que no se realizará nuevamente la identificación de relaciones entre clases durante el proceso de diseño por haberse cumplimentado en el presente proceso.

6.5.2.1. Asociaciones

Las asociaciones que se describen a continuación son todas binarias. Al momento del análisis no surge que ninguna de ellas tenga propiedades por lo que simplemente, se las menciona.

Clase	Card.	Nombre	Card.	Clase Asociada

Clase	Card.	Nombre	Card.	Clase Asociada
Casillero	1	Está limitado por	1	Lados
Casillero	1	Se ve como	1	Casillero Visual
Controlado	1	Cuenta con	1	Tablero
Controlador	1	Puede usar	1	Registrador de Eventos
Jugador	1	Dispone de	12	Pieza
Lados de Pieza Visual	1	Está formado por	4	Lado de Pieza Visual
Pieza	1	Se ve como	1	Pieza Visual
Pieza	1	Se muestra inicialmente	1	Lados
Pieza	1	Se ve actualmente	1	Lados de Pieza Visual
Pieza Visual	1	Inicia en	1	Casillero Visual de Posición Inicial
Tablero	1	Dispone de	1	Botón de Inicio
Tablero	1	Compite	2	Jugador
Tablero	1	Puede invocar	3	Manifestación de Resultados
Tablero	1	Puede invocar	1	Manifestación de Error
Tablero	1	Contiene	1	Piezas
Tablero	1	Contiene	1	Casilleros
Tablero	1	Contiene	1	Bordes de Tablero Visual
Tablero	1	Dispone de	1	Indicador de Rotación

Clase	Card.	Nombre	Card.	Clase Asociada
				hacia la Derecha
Tablero	1	Dispone de	1	Indicador de Rotación hacia la Izquierda

Tabla 6-4 - Asociaciones entre Clases

6.5.2.1 Agregaciones

Se observa el caso particular donde las agregaciones están compuestas por un número dado (24 en todos los casos) de Objetos de una misma clase. Igualmente, se considera una agregación, dado que la clase compuesta, lo será exclusivamente si todos y cada uno de los objetos agregados y ningún otro, la conforman. Ellas son:

Clase	Card.	Nombre	Card.	Clase Agregada
Bordes de Tablero Visual	1	Está formado por	24	Borde de Tablero Visual
Casilleros	1	Está compuesto por	24	Casillero
Piezas	1	Está compuesta por	24	Pieza

Tabla 6-5 - Agregaciones

6.5.3 Identificación de Generalizaciones

En este apartado se consideran aquellas clases que contienen información en común y que pueden generalizarse, para facilitar luego la implementación de herencias.

Se incluye un caso de herencia sin generalización, dónde la clase hija hereda todos los atributos de la clase padre:

- *Indicador de Rotación hacia la Derecha* e *Indicador de Rotación hacia la Izquierda* pueden generalizarse en una clase *Indicador de Rotación*.

- *Lados de Pieza Visual* hereda de *Lados*, ya que los atributos correspondientes a la información guardada en cada uno de los atributos de *Lados*, son estrictamente necesarios en *Lados de Pieza Visual*. Esta última agrega sus propios atributos que serán cada uno de los lados que observará el usuario.

Se menciona a continuación un caso que al momento del análisis no se puede confirmar si será o no una generalización ya que ello dependerá de la arquitectura del sistema, y es el que corresponde a las clases de los objetos visuales como ser: Pieza Visual, Casillero Visual, Casillero de Posición Inicial Visual, Borde de Tablero Visual, Indicadores de Rotación y Lado de Pieza Visual que probablemente hereden de un mismo componente visual. Algo similar podría ocurrir con Manifestación de Resultados y Manifestación de Error, es más, estos podrían no llegar a ser clases, sino procedimientos. Estas cuestiones se resolverán durante el proceso de diseño.

6.5.4 Modelo de Clases de Análisis

A continuación se puede observar el modelo de clases de análisis obtenido (Ilustración 6-24) que por simplicidad, incluye exclusivamente los nombres de las clases, las relaciones y la cardinalidad de las relaciones (no así los atributos y operaciones). Es importante destacar que solamente se indica la cardinadad donde ésta es distinta de 1.

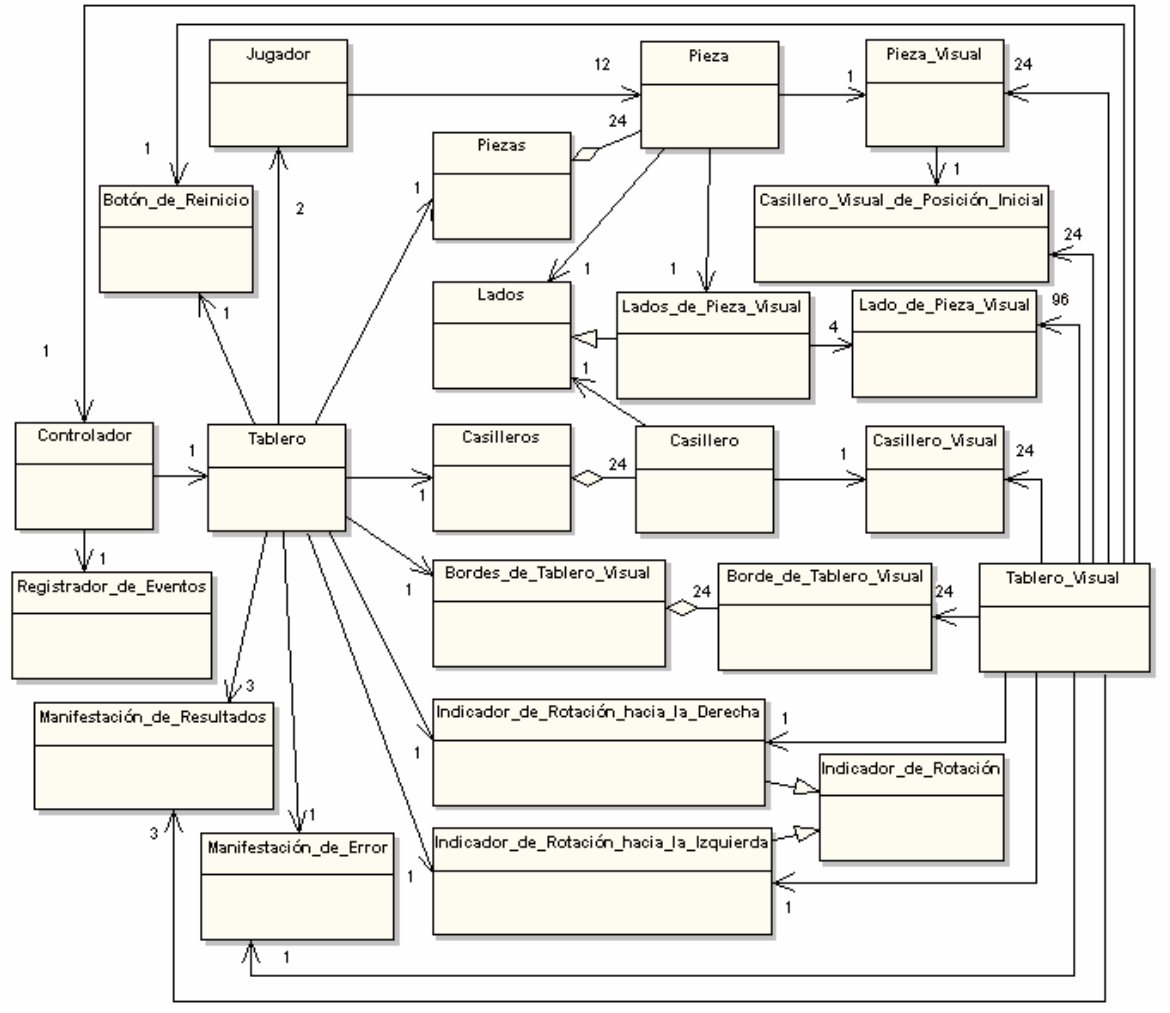


Ilustración 6-24 - Modelo de Clases de Análisis

En el diagrama, cada rectángulo representa una clase de análisis y los conectores se definen de la siguiente manera:

-  : Asociación. La flecha apunta a la clase referenciada desde la que hace la referencia.
-  : Agregación. El lado del rombo indica la clase que agrupa a los objetos de la otra clase.
-  : Generalización. El lado del triángulo apunta hacia la clase que generaliza a todas aquellas desde donde parten las flechas.

6.6 Definición de Interfaces de Usuario

En la presente actividad se completa y unifica la definición de la interfaz que se hizo puntualmente en el análisis de cada caso de uso, donde para cada caso identificado, se describió la interfaz del usuario correspondiente. Los detalles que dependan de la arquitectura no se mencionarán en este ítem ya que se corresponde al Item 7.1.2.3. Cabe aclarar que no se realiza un explotación de interfaces de usuario dado que la misma es única.

6.6.1 Especificación de Principios Generales de la Interfaz

En esta tarea se van a especificar los elementos a tener en cuenta en la definición de la interfaz de usuario (para la interfaz interactiva gráfica). No será de aplicación la impresión de reportes dadas las características y alcances del software.

Directrices generales en cuanto a la interfaz y aspectos generales de interacción.

- El entorno será tridimensional (3D).
- La interacción será con el ratón.
- Solamente se utilizará el click del botón izquierdo como entrada válida, que el sistema deberá interpretar según el estado de la partida y el objeto que se haya indicado.
- El desplazamiento de las piezas no dependerá de las capacidades del equipo en el que está corriendo el software. Esto tiene un límite y deberá especificarse un equipo mínimo.

Principios de composición de pantallas y criterios de ubicación de los distintos elementos dentro de cada formato.

- La distribución del tablero es la que se observa en la Ilustración 6-25 :

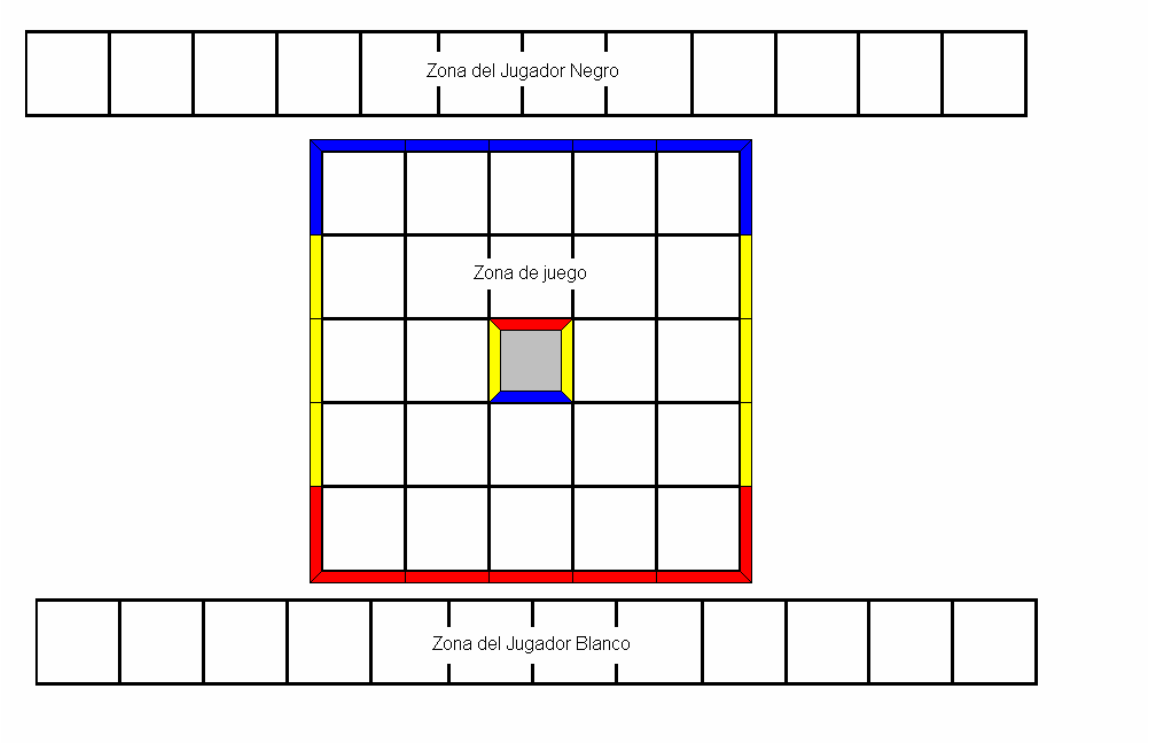


Ilustración 6-25 - Distribución del Tablero

- El tablero estará en posición central.
- Las fichas no utilizadas del usuario en el sector inferior de la pantalla.
- Las fichas no utilizadas del oponente estarán en la parte superior de la pantalla.

Normas para los mensajes de error y aviso, codificación, presentación y comportamientos.

- Los mensajes de error serán visuales y/o sonoros.
- Deberán ser lo más claros posibles, sin hacer uso del lenguaje.

Normas para la presentación de ayudas.

- Las ayudas se especifican en cada caso de uso y aparecerán automáticamente al cumplirse el TimeOut en el estado actual en que el usuario aun no realizó la interacción esperada.
- Las ayudas consistirán en resaltar / indicar el objeto sobre el cual se espera que el usuario haga click.

La tarea Especificación de Formatos Individuales de la Interfaz de Pantalla, no ha de realizarse debido a que se cuenta con una única pantalla la cual dinámicamente se adapta a las interacciones del usuario.

6.6.2 Especificación del Comportamiento Dinámico de la Interfaz

Se reproduce a continuación en forma unificada, el comportamiento dinámico de la interfaz de usuario que fue redactada en los sucesivos casos de uso, para dar una idea global del mismo. La idea global ya ha sido reflejada en el diagrama de estados presentado.

Los movimientos que serán respuesta a interacciones del usuario o correspondientes al oponente cuando corresponda, serán de dos tipos:

Traslación:

- Selección de Pieza: En forma vertical a una posición superior a la original.
- Selección de Casillero: En forma horizontal hasta posarse sobre el mismo quedando a la misma altura que en el caso anterior.
- Aceptación de Pieza: En forma vertical hasta posarse sobre el casillero elegido.
- Cancelación de Pieza: En forma horizontal hasta posarse sobre el casillero de posición inicial y luego en forma vertical hasta posarse sobre el casillero de posición inicial.

Rotación:

- Rotación hacia la Derecha: La pieza rotará 90º en el sentido de las agujas del reloj.
- Rotación hacia la Izquierda: La pieza rotará 90º en el sentido contrario al de las agujas del reloj.

Vale destacar que se impondrá el sincronismo de los movimientos. Esto significa que hasta no finalizar un movimiento, no serán validas las interacciones por parte del usuario (que sí serán registradas). Esto de alguna manera intenta mantener un orden en el desarrollo de la partida.

Las interacciones de ayuda o error, no deberán restringir la interacción del usuario. Por ejemplo, si el usuario recibe ayuda indicándose qué pieza debe elegir, y suponiendo que la duración de dicha indicación sea de 3 segundos, si al segundo número 2, hizo click sobre la pieza, la selección es válida. Será inválida luego, si mientras se está trasladando la pieza a la posición destacada de pieza seleccionada, el usuario hace click sobre el casillero seleccionado.

6.7 Análisis de Consistencia

Con el objeto de garantizar la calidad del sistema, a continuación se presentan las tareas realizados que analizan la consistencia del análisis llevado a cabo.

6.7.1 Verificación de los Modelos

Cada uno de los modelos abstraídos, fue objeto de verificación y corrección. Dicha tarea consistió en la revisión total del presente capítulo, en particular del Modelo y Especificación de los Casos de Uso y en el Modelo de Clases de Análisis.

6.7.2 Análisis de Consistencia entre Modelos

A continuación se presentan las tablas que permitieron analizar la consistencia entre modelos y corregirlos adecuadamente en aquellos casos que fueron necesarios.

6.7.2.1 Mensajes de Diagramas de Interacción de Objetos vs. Operaciones del Modelo de Clases de Análisis.

Esta tabla permite verificar la correspondencia entre los mensajes que aparecen en los diagramas de Interacción y las operaciones del Modelo de Clases de Análisis. Para ello, se obtuvieron todos los mensajes que aparecen en cada diagrama de secuencias y se verificó que la clase del objeto receptor implemente el mensaje y además, que la clase que emite el mensaje pueda tener acceso al objeto receptor.

6.7.2.2 Especificación de Requerimientos Funcionales vs. Modelo de Casos de Uso

Esta tabla permite verificar la correspondencia entre los requerimientos funcionales y casos de uso, es decir, que si existe un requisito, al menos un caso de uso lo contemple y viceversa, si existe un caso de uso, este provenga de un requisito.

		Selecciona Pieza	Selecciona Casillero	Selecciona Orientación de Pieza	Rota Pieza 90º hacia la Izquierda	Rota Pieza 90º hacia la Derecha	Acepta Pieza	Cancela Pieza Seleccionada	Selecciona Objeto Inapropiado	Almacenar Error en trazado	Chequear Final de Partida	Inicia Partida	Finalizar Partida	Realizar Jugada del Oponente	Pensar Jugada	Asiste al Jugador	Obtiene Trazado
Trazar Partida	Inicio de la partida											X					
	Selección de Pieza	X															
	Selección de Casillero		X														

		Selecciona Pieza	Selecciona Casillero	Selecciona Orientación de Pieza	Rota Pieza 90º hacia la Izquierda	Rota Pieza 90º hacia la Derecha	Acepta Pieza	Cancela Pieza Seleccionada	Selecciona Objeto Inapropiado	Almacenar Error en trazado	Chequear Final de Partida	Inicia Partida	Finalizar Partida	Realizar Jugada del Oponente	Pensar Jugada	Asiste al Jugador	Obtiene Trazado
	Indicación de Rotación			X	X	X											
	Aceptación de Pieza en Casillero						X										
	Invocaciones erróneas de Objetos								X								
	Ayudas del sistema															X	
	Resultado												X				
Jugar Partida	Iniciar Partida											X					
	Asignación del Jugador												X				
	Seleccionar Pieza	X															
	Seleccionar Casillero		X														
	Rotar Pieza			X	X	X											
	Aceptar Pieza						X										
	Respetar secuencia																
	Cancelar selección de Pieza elegida							X									
	Cambiar selección de casillero			X													
	Jugar en lugar del oponente													X			
	Detectar condiciones de										X						

		Selecciona Pieza	Selecciona Casillero	Selecciona Orientación de Pieza	Rota Pieza 90º hacia la Izquierda	Rota Pieza 90º hacia la Derecha	Acepta Pieza	Cancela Pieza Seleccionada	Selecciona Objeto Inapropiado	Almacenar Error en trazado	Chequear Final de Partida	Inicia Partida	Finalizar Partida	Realizar Jugada del Oponente	Pensar Jugada	Asiste al Jugador	Obtiene Trazado
	final de juego																
	Ayudas															X	
	Tiempos de espera para mostrar ayudas															X	

Tabla 6-6 - Requerimientos Funcionales vs. Modelo de Casos de Uso

Se observa que el único requisito que no se ha cumplido con casos de uso es el de Respetar Secuencia. Este requisito fue analizado a través del diagrama de estados del Controlador, con lo cual el requisito está cubierto.

6.7.2.3 Modelo de Clases vs. Diagramas Dinámicos

Estas tablas permiten verificar la correspondencia entre el Modelo de Clases y el diagrama dinámico que lo incluye a fin de observar que:

- Cada objeto en el diagrama de interacción de objetos tiene una correspondencia en el modelo de clases (Objetos de Diagramas de Secuencia vs. Clases del Diagrama de Clases).
- Cada mensaje se corresponde con una operación del objeto que lo recibe (Mensajes de Diagramas de Secuencias vs. Operaciones de Clases).
- La clase que recibe un mensaje con petición de datos tiene capacidad para proporcionar esos datos (Mensajes con petición de datos vs. Atributos de Clases).

Objetos de Diagramas de Secuencia vs. Clases del Diagrama de Clases

Esta tabla no se incluye, dado que dicha verificación fue realizada en **Interacción entre Objetos e Identificación de Clases Asociadas** (6.4.1), donde cada diagrama de secuencia fue seguido de las clases involucradas, siendo estas, todas clases válidas incluidas en el diagrama del Modelo de Clases de Análisis.

Mensajes de Diagramas de Secuencias vs. Operaciones de Clases

Diagrama	Objeto	Mensaje	Clase	Operación
Acepta Pieza	Casillero Seleccionado	Asignar Pieza Seleccionada	Casillero Seleccionado	Establecer Usado
Acepta Pieza	Controlador	Finalizó Desplazamiento	Controlador	Finalizó Desplazamiento
Acepta Pieza	Controlador	Pieza Visual Seleccionada Señalada	Controlador	Objeto ha sido señalado
Acepta Pieza	Controlador	Registrar	Controlador	Registrar evento
Acepta Pieza	Controlador	Registrar	Controlador	Registrar evento
Acepta Pieza	Pieza Seleccionada	Finalizó Desplazamiento	Pieza	Finalizó Desplazamiento
Acepta Pieza	Pieza Seleccionada	Pieza Jugada	Pieza	Establecer Usada
Acepta Pieza	Pieza Visual Seleccionada	Aceptar Pieza	Pieza Visual	Establecer Posición
Acepta Pieza	Pieza Visual Seleccionada	Click	Pieza Visual	Transmitir Interacción
Acepta Pieza	Tablero	Puede Aceptarse	Tablero	Puede Aceptarse
Asiste al Jugador (Antes de Seleccionar Pieza)	Controlador	Registrar	Controlador	Registrar evento
Asiste al Jugador (Antes de Seleccionar Pieza)	Controlador	TimeOut	Controlador	Atender TimeOut
Asiste al Jugador (Antes de Seleccionar Pieza)	Pieza Posible	Ayudar Señalando Pieza	Pieza	Mostrar Ayuda
Asiste al Jugador (Antes de Seleccionar Pieza)	Pieza Visual Posible	Mostrar Ayuda	Pieza Visual	Mostrar Ayuda

Diagrama	Objeto	Mensaje	Clase	Operación
Asiste al Jugador (Antes de Seleccionar Pieza)	Tablero	Obtener Pieza Posible	Tablero	Obtener Pieza Posible
Asiste al Jugador (Casillero Seleccionado y la Pieza Puede Jugarse sin Rotar)	Controlador	Registrar	Controlador	Registrar evento
Asiste al Jugador (Casillero Seleccionado y la Pieza Puede Jugarse sin Rotar)	Controlador	TimeOut	Controlador	Atender TimeOut
Asiste al Jugador (Casillero Seleccionado y la Pieza Puede Jugarse sin Rotar)	Pieza Visual Seleccionada	Mostrar Ayuda	Pieza Visual	Mostrar Ayuda
Asiste al Jugador (Casillero Seleccionado y la pieza no puede jugarse si no se la rota)	Controlador	Registrar	Controlador	Registrar evento
Asiste al Jugador (Casillero Seleccionado y la pieza no puede jugarse si no se la rota)	Controlador	TimeOut	Controlador	Atender TimeOut
Asiste al Jugador (Casillero Seleccionado y la pieza no puede jugarse si no se la rota)	Indicador de Rotación Obtenido	Mostrar Ayuda	Indicador de Rotación	Mostrar Ayuda
Asiste al Jugador (Casillero Seleccionado y la pieza no puede jugarse si no se la rota)	Tablero	Obtener Indicador de Rotación	Tablero	Obtener Indicador de Rotación
Asiste al Jugador (Pieza Seleccionada)	Casillero Posible	Ayudar Señalando Casillero	Casillero	Mostrar Ayuda

Diagrama	Objeto	Mensaje	Clase	Operación
Asiste al Jugador (Pieza Seleccionada)	Casillero Visual Posible	Mostrar Ayuda	Casillero Visual	Mostrar Ayuda
Asiste al Jugador (Pieza Seleccionada)	Controlador	Registrar	Controlador	Registrar evento
Asiste al Jugador (Pieza Seleccionada)	Controlador	TimeOut	Controlador	Atender TimeOut
Asiste al Jugador (Pieza Seleccionada)	Tablero	Obtener Casillero Posible para Pieza Seleccionada	Tablero	Obtener Casillero Posible para Pieza Seleccionada
Cancela Pieza Seleccionada	Casillero Seleccionado	Desseleccionar	Casillero	Establecer Seleccionado
Cancela Pieza Seleccionada	Controlador	Finalizó Desplazamiento	Controlador	Finalizó Desplazamiento
Cancela Pieza Seleccionada	Controlador	Objeto del Sector del Usuario Señalado	Controlador	Objeto ha sido señalado
Cancela Pieza Seleccionada	Controlador	Registrar	Controlador	Registrar evento
Cancela Pieza Seleccionada	Controlador	Registrar y Cambiar de Estado a Antes de Seleccionar Pieza	Controlador	Registrar evento - Cambiar de Estado
Cancela Pieza Seleccionada	Objeto del Sector del Usuario	Click	Objeto del Sector del Usuario	Transmitir Interacción
Cancela Pieza Seleccionada	Pieza Seleccionada	Desseleccionar	Pieza	Establecer Seleccionada
Cancela Pieza Seleccionada	Pieza Seleccionada	Finalizó Desplazamiento	Pieza	Finalizó Desplazamiento
Cancela Pieza Seleccionada	Pieza Visual Seleccionada	Regresar a la Posición Inicial	Pieza Visual	Establecer Posición
Chequear Final de Partida	Controlador	Chequear Final de Partida	Controlador	Chequear Final de Partida
Finalizar Partida	Botón de Reinicio	Mostrar	Botón de Reinicio	Mostrar / Ocultar (Establecer Visible).

Diagrama	Objeto	Mensaje	Clase	Operación
Finalizar Partida	Controlador	Activar Timeout de Reinicio de Partida	Controlador	Habilitar/Deshabilitar TimeOut de Reinicio
Finalizar Partida	Controlador	Registrar	Controlador	Registrar evento
Finalizar Partida	Manifestación de Resultado	Activar	Manifestación de Resultado	Activar
Inicia Partida	Botón de Reinicio	Ocultar	Botón de Reinicio	Mostrar / Ocultar (Establecer Visible).
Inicia Partida	Controlador	Desactivar TimeOut de Reinicio	Controlador	Habilitar/Deshabilitar TimeOut de Reinicio
Inicia Partida	Controlador	Realizar Jugada Oponente	Controlador	Realizar Jugada Oponente
Inicia Partida	Controlador	Registrar	Controlador	Registrar evento
Inicia Partida	Tablero	Inicializar Casilleros de Juego	Tablero	Inicializar Casilleros
Inicia Partida	Tablero	Inicializar Jugadores	Tablero	Inicializar Jugadores
Inicia Partida	Tablero	Inicializar Piezas	Tablero	Inicializar Piezas
Inicia Partida	Tablero	Inicializar	Tablero	Inicializar
Inicia Partida	Tablero	Rotar Tablero 180º	Tablero	Orientar Tablero Visual según Jugador
Inicia Partida	Tablero Visual	Mostrar	Tablero Visual	Mostrar / Ocultar
Inicializar Casilleros de Juego	Borde de Tablero Visual	Asignar Color	Borde de Tablero Visual	Establecer Color
Inicializar Casilleros de Juego	Casillero Visual	Indicar Posición	Casillero Visual	Establecer Posición
Inicializar Casilleros de Juego	Lado de Casillero	Asignar Color	Lados	Establecer Arriba, Abajo, Derecha e Izquierda
Inicializar Jugadores	Jugador	Asignar Color	Jugador	Establecer Jugador del Usuario
Inicializar Jugadores	Pieza	Asignar Jugador	Pieza	Establecer Jugador

Diagrama	Objeto	Mensaje	Clase	Operación
				Dueño
Inicializar Jugadores	Pieza Visual	Asignar Color	Pieza Visual	Establecer Color
Inicializar Piezas	Casillero Visual de Posición Inicial	Indicar Posición	Casillero Visual de Posición Inicial	Establecer Posición
Inicializar Piezas	Lado de Pieza	Asignar Color	Lados	Establecer Arriba, Abajo, Derecha e Izquierda
Inicializar Piezas	Lado Visual de Pieza	Asignar Color	Lados	Establecer Arriba, Abajo, Derecha e Izquierda
Inicializar Piezas	Pieza	Indicar Posición Inicial	Pieza	Establecer Casillero Visual de Posición Inicial
Inicializar Piezas	Pieza Visual	Indicar Posición Inicial	Pieza Visual	Establecer Posición
Obtiene Trazado	Controlador	Adecuar Trazados	Controlador	Adecuar Trazados
Obtiene Trazado	Controlador	Tecla Especial	Controlador	Tecla ha sido indicada
Obtiene Trazado	Explorador de Archivos	Abrir carpeta de Trazados	Explorador de Archivos	Abrir carpeta de Trazados
Pensar Jugada	Tablero	Obtener Casillero Posible para Pieza Seleccionada	Tablero	Obtener Casillero Posible para Pieza Seleccionada
Pensar Jugada	Tablero	Obtener Pieza Posible	Tablero	Obtener Pieza Posible
Pensar Jugada	Tablero	Seleccionar Mejor Opción	Tablero	Sugerir Movida para Jugador
Realizar Jugada del Oponente	Casillero de la Movida del Oponente	Asignar Pieza Seleccionada	Casillero	Establecer Pieza
Realizar Jugada del Oponente	Casillero de la Movida del Oponente	Colocar Pieza Seleccionada encima	Casillero	Desplazar Pieza Encima
Realizar Jugada del Oponente	Controlador	Finalizó Desplazamiento	Controlador	Finalizó Desplazamiento

Diagrama	Objeto	Mensaje	Clase	Operación
Realizar Jugada del Oponente	Controlador	Finalizó Desplazamiento	Controlador	Finalizó Desplazamiento

Diagrama	Objeto	Mensaje	Clase	Operación
Realizar Jugada del Oponente	Controlador	Finalizó Desplazamiento	Controlador	Finalizó Desplazamiento
Realizar Jugada del Oponente	Controlador	Movida del Oponente	Controlador	Obtener Movida del Oponente
Realizar Jugada del Oponente	Controlador	Registrar	Controlador	Registrar evento
Realizar Jugada del Oponente	Controlador	Registrar	Controlador	Registrar evento
Realizar Jugada del Oponente	Controlador	Registrar	Controlador	Registrar evento
Realizar Jugada del Oponente	Controlador	Registrar	Controlador	Registrar evento
Realizar Jugada del Oponente	Pieza de la Movida del Oponente	Finalizó Desplazamiento	Pieza	Finalizó Desplazamiento
Realizar Jugada del Oponente	Pieza de la Movida del Oponente	Finalizó Desplazamiento	Pieza	Finalizó Desplazamiento
Realizar Jugada del Oponente	Pieza de la Movida del Oponente	Finalizó Desplazamiento	Pieza	Finalizó Desplazamiento
Realizar Jugada del Oponente	Pieza de la Movida del Oponente	Pieza Jugada	Pieza	Establecer Usada
Realizar Jugada del Oponente	Pieza de la Movida del Oponente	Seleccionar	Pieza	Establecer Seleccionada
Realizar Jugada del Oponente	Pieza Visual de la Movida del Oponente	Aceptar Pieza	Pieza Visual	Establecer Posición
Realizar Jugada del Oponente	Pieza Visual de la Movida del Oponente	Mover la Pieza Seleccionada sobre el Casillero	Pieza Visual	Establecer Posición

Diagrama	Objeto	Mensaje	Clase	Operación
Realizar Jugada del Oponente	Pieza Visual de la Movida del Oponente	Seleccionar	Pieza Visual	Establecer Posición
Realizar Jugada del Oponente	Tablero	Pensar Movida del Oponente	Tablero	Sugerir Movida para Jugador
Rota Pieza 90º hacia la Derecha	Controlador	Finalizó Rotación	Controlador	Finalizó Rotación
Rota Pieza 90º hacia la Derecha	Controlador	Indicador de Rotación hacia la Derecha Señalado	Controlador	Objeto ha sido señalado
Rota Pieza 90º hacia la Derecha	Controlador	Registrar	Controlador	Registrar evento
Rota Pieza 90º hacia la Derecha	Controlador	Registrar	Controlador	Registrar evento
Rota Pieza 90º hacia la Derecha	Indicador de Rotación hacia la Derecha	Click	Indicador de Rotación hacia la Derecha	Transmitir Interacción
Rota Pieza 90º hacia la Derecha	Pieza Seleccionada	Cambiar Orientación	Pieza	Rotar 90º hacia Izquierda o Derecha
Rota Pieza 90º hacia la Derecha	Pieza Seleccionada	Finalizó Rotación	Pieza	Finalizó Rotación
Rota Pieza 90º hacia la Derecha	Pieza Seleccionada	Rotar 90º hacia la Derecha	Pieza	Rotar 90º hacia la Derecha
Rota Pieza 90º hacia la Derecha	Pieza Visual Seleccionada	Rotar 90º hacia la Derecha	Pieza Visual	Rotar 90º hacia la Derecha
Rota Pieza 90º hacia la Izquierda	Controlador	Finalizó Rotación	Controlador	Finalizó Rotación
Rota Pieza 90º hacia la Izquierda	Controlador	Indicador de Rotación hacia la Izquierda Señalado	Controlador	Objeto ha sido señalado
Rota Pieza 90º hacia la Izquierda	Controlador	Registrar	Controlador	Registrar evento

Diagrama	Objeto	Mensaje	Clase	Operación
Rota Pieza 90º hacia la Izquierda	Controlador	Registrar	Controlador	Registrar evento
Rota Pieza 90º hacia la Izquierda	Indicador de Rotación hacia la Izquierda	Click	Indicador de Rotación hacia la Izquierda	Transmitir Interacción
Rota Pieza 90º hacia la Izquierda	Pieza Seleccionada	Cambiar Orientación	Pieza	Rotar 90º hacia Izquierda o Derecha
Rota Pieza 90º hacia la Izquierda	Pieza Seleccionada	Finalizó Rotación	Pieza	Finalizó Rotación
Rota Pieza 90º hacia la Izquierda	Pieza Seleccionada	Rotar 90º hacia la Izquierda	Pieza	Rotar 90º hacia la Izquierda
Rota Pieza 90º hacia la Izquierda	Pieza Visual Seleccionada	Rotar 90º hacia la Izquierda	Pieza Visual	Rotar 90º hacia la Izquierda
Selecciona Casillero	Casillero	Asignar Pieza Seleccionada	Casillero	Establecer Pieza
Selecciona Casillero	Casillero	Colocar Pieza Seleccionada encima	Casillero	Desplazar Pieza Encima
Selecciona Casillero	Casillero	Puede Seleccionarse	Casillero	Podría Aceptarse Pieza (previa rotación si fuera necesaria)
Selecciona Casillero	Casillero	Seleccionar	Casillero	Establecer Seleccionado
Selecciona Casillero	Casillero Visual	Click	Casillero Visual	Transmitir Interacción
Selecciona Casillero	Controlador	Cambiar de Estado a Casillero Seleccionado	Controlador	Establecer Casillero Seleccionado
Selecciona Casillero	Controlador	Casillero Señalado	Controlador	Objeto ha sido señalado
Selecciona Casillero	Controlador	Finalizó Desplazamiento	Controlador	Finalizó Desplazamiento
Selecciona Casillero	Controlador	Registrar	Controlador	Registrar evento
Selecciona Casillero	Controlador	Registrar	Controlador	Registrar evento

Diagrama	Objeto	Mensaje	Clase	Operación
Selecciona Casillero	Pieza Seleccionada	Finalizó Desplazamiento	Pieza	Finalizó Desplazamiento
Selecciona Casillero	Pieza Visual Seleccionada	Mover la Pieza Seleccionada sobre el Casillero	Pieza Visual	Establecer Posición
Selecciona Objeto Inapropiado	Controlador	Objeto Inadecuado Señalado	Controlador	Objeto ha sido señalado
Selecciona Objeto Inapropiado	Controlador	Registrar	Controlador	Registrar evento
Selecciona Objeto Inapropiado	Manifestación de Error	Activar	Manifestación de Error	Activar
Selecciona Objeto Inapropiado	Objeto Inadecuado	Click	Pieza Visual o Casillero Visual de Posición Inicial	Transmitir Interacción
Selecciona Pieza	Controlador	Cambiar de Estado a Pieza Seleccionada	Controlador	Cambiar de Estado a Pieza Seleccionada
Selecciona Pieza	Controlador	Finalizó Desplazamiento	Controlador	Finalizó Desplazamiento
Selecciona Pieza	Controlador	Pieza Señalada	Controlador	Objeto ha sido señalado
Selecciona Pieza	Controlador	Registrar	Controlador	Registrar evento
Selecciona Pieza	Controlador	Registrar	Controlador	Registrar evento
Selecciona Pieza	Pieza	Finalizó Desplazamiento	Pieza	Finalizó Desplazamiento
Selecciona Pieza	Pieza	Puede Seleccionarse	Pieza	Puede Seleccionarse
Selecciona Pieza	Pieza	Seleccionar	Pieza	Establecer Seleccionada
Selecciona Pieza	Pieza Visual	Click	Pieza Visual	Transmitir Interacción
Selecciona Pieza	Pieza Visual	Seleccionar	Pieza Visual	Establecer Posición

Tabla 6-7 - Diagramas de Secuencias vs. Operaciones

Mensajes con petición de datos vs. Atributos de Clases

Diagrama	Objeto	Mensaje	Clase	Atributo
Asiste al Jugador (Antes de Seleccionar Pieza)	Tablero	Obtener Pieza Posible	Tablero	Piezas
Asiste al Jugador (Pieza Seleccionada)	Tablero	Obtener Casillero Posible para Pieza Seleccionada	Tablero	Casilleros
Pensar Jugada	Tablero	Obtener Casillero Posible para Pieza Seleccionada	Tablero	Casilleros
Asiste al Jugador (Casillero Seleccionado y la pieza no puede jugarse si no se la rota)	Tablero	Indicador de Rotación	Tablero	Obtener Indicador de Rotación hacia la Derecha e Izquierda

Tabla 6-8 - Mensajes con Petición de Datos vs. Atributos de Clase

6.8 Especificación del Plan de Pruebas de software

Esta actividad tiene como objetivo definir el plan de pruebas de software, que permitirá verificar que el sistema cumple con las necesidades establecidas por el solicitante. Se incluyen los lineamientos de las pruebas que se realizarán en los siguiente procesos.

6.8.1 Definición del Alcance de las Pruebas

Como el sistema está compuesto por un único módulo las pruebas unitarias coinciden con las del sistema. Ello se debe a que no se realizarán pruebas de integración. A ellas se sumarán las pruebas de implementación correspondientes.

La definición de las pruebas deberá establecerse con el objetivo fundamental de verificar el cumplimiento de las condiciones que establece el diagrama de estados de la partida.

En principio las pruebas serán de "Caja negra", indicándose claramente los objetivos en cada caso, las acciones a tomar y las respuestas esperadas.

Por lo tanto, las pruebas se agruparán de la siguiente manera:

- **Inicio de la Partida**: Verificar la correcta posición de las piezas, la ubicación del tablero de acuerdo al color de piezas asignados al usuario.

- **Selección de Pieza**: Selección de una Pieza, Cancelación, Selección de Otra Pieza, Selección de Objetos Inapropiados, Mostrar Ayuda para Seleccionar Pieza.
- **Selección de Casillero**: Selección de Casillero, Selección de Objetos Inapropiados, Cancelación de otra Pieza, Mostrar Ayuda para Seleccionar Casillero.
- **Rotación y Aceptación de Pieza**: Aceptación de Pieza, Cancelación de Pieza, Cambio de Selección de Casillero, Rotación de Pieza hacia la Derecha e Izquierda, Selección de Objetos Inapropiados, Intentar Aceptar Pieza que necesita ser rotada Previamente.
- **Final de Partida**: Probar cada caso posible haciendo uso de tabla de decisión. Mostrar Manifestación de Resultados.
- **Obtener Trazado**: Obtener el trazado de las partidas realizadas. Abrir trazados con programa para planillas de cálculos.

6.8.2 Definición de Requisitos del Entorno de Pruebas

El entorno de pruebas deberá cumplir con las siguientes características:

Hardware (configuración mínima):

- PC – Procesador 800 Mhz
- 128 Mb RAM
- 10 Mb espacio libre en disco (estimado)
- Placa de Sonido
- Ratón
- Teclado
- Parlantes
- Monitor Color
- Placa Aceleradora de Video

Sistema Operativo:

- Windows 2000 / XP
- OpenGL (es nativo en Windows 2000 y XP)

Otras Condiciones del Entorno de Pruebas:

- El equipo de pruebas no coincidirá con el de desarrollo para las pruebas de "Caja Negra".

6.8.3 Definición de las Pruebas de Aceptación del Sistema

Las pruebas de aceptación del sistema quedan definidas a partir de los requisitos, estableciéndose para este proyecto de la siguiente manera:

- El Investigador juega una partida haciendo las veces del niño.
- El Investigador obtiene el trazado de su partida.
- El Investigador designa a un niño sin NEE para que realice la experiencia de las partidas que sean necesarias hasta que haya demostrado el aprendizaje de las reglas.
- El Investigador obtiene el trazado las partidas experimentadas por el niño.
- Cada trazado de partida podrá analizarse desde un programa para planillas de cálculo, sin necesidad de hacer conversiones ni adaptaciones al archivo.

Las mencionadas pruebas, será evaluadas en el Ítem 8.6.

6.9 Comprobación de Calidad de los Requisitos

Para comprobar la calidad de los rquisitos, se inspeccionó el software con la técnica de lectura con "checklist" [Juristo N., Moreno A., Vegas S. 2005]. La lista se divide en tres partes que sucesivamente comprenden el contenido (Ítem 6.9.1), la completitud (Ítem 6.9.2) y la calidad de los requisitos del software (Ítem 6.9.3).

6.9.1 Contenido de los requisitos

- ¿Están especificadas todas las entradas al sistema, incluyendo su origen, precisión, rango de valores y frecuencia?

  En el Ítem 6.2.2.1 se especifican claramente las entradas posibles al sistema. El diagrama de estados presentado contempla los casos posibles de entradas erróneas. Asimismo se define en el Ítem 6.2.2.2 cómo obtener la salida del sistema que dadas las particularidades del caso es una funcionalidad "oculta" y se realiza con una combinación de teclas (CTRL+T).

- ¿Están especificadas todas las salidas del sistema, incluyendo su destino, precisión, rango de valores, frecuencia y formato?

  En el Ítem 3.3.2.2 se plantean los requerimientos a incluir en la única salida no interactiva del sistema (el trazado de la partida), observándose todas las situaciones que se desean registrar. El resto de las salidas que son respuesta a las indicaciones realizadas con el ratón fueron especificadas en el Ítem 6.2.2.1 en el diagramas de estados de la partida y definidas claramente en las especificaciones en Ítem 6.2.2.2.

  ¿Están todos los formatos de los informes especificados?

Sólo existe un requerimiento al respecto y es que el único modelo del archivo de registro sea posible leerlo desde una planilla de cálculos o sea sencillo de importar desde una base de datos en el Ítem 3.3.1.

- ¿Están especificadas las interfaces con otros sistemas software y hardware externas?

  No existen interfaces externas de hardware ni comunicación con otros sistemas software.

- ¿Están especificados todos los interfaces de comunicación, incluyendo handshaking, chequeo de errores y protocolos de comunicación?

  No corresponden al presente desarrollo.

- ¿Se ha especificado, para todas aquellas operaciones que sea necesario, el tiempo esperado de respuesta desde el punto de vista del usuario?

  En la tarea de identificación de clases y atributos (Ítem 6.5.1), al definirse las responsabilidades de la clase Controlador, se establece que se encargará de gestionar el TimeOut para la ayuda al sistema.[1]

- ¿Se han especificado otras consideraciones de temporización, como el tiempo de procesamiento, transferencia de datos y rendimiento del sistema?

  Se ha especificado como requisito de hardware en el Ítem 6.8.2 que el equipo contenga una placa aceleradora de video (aunque es un estándar en la actualidad), dado que el uso de la biblioteca OpenGL por software haría inutilizable el sistema.

- ¿Se han especificado todas las tareas que el usuario desea realizar?

  Se ha especificado cada caso de uso en el Ítem 6.2.2.2.

- ¿Especifica cada tarea los datos que se usan en la tarea y los datos resultantes de la tarea?

  En el Ítem 6.2.2.2 se define cada caso de uso a través de la descripción del escenario, precondiciones, postcondiciones, interfaz de usuario y condiciones de fallo.

- ¿Se ha especificado el nivel de seguridad?

  En el Ítem 5.2.1 se define la seguridad del producto.

---

[1] Su definición fue completada en la etapa de diseño estableciendo una clase asociada al estado denominada **TdynamicTimeout** definida en el Ítem 7.3.3.4

- ¿Se ha especificado la fiabilidad incluyendo las consecuencias de los fallos del software, información vital que ha de protegerse de fallos, detección de errores y recuperación de los mismos?

  En caso de fallo en la instalación, en el Ítem 5.2.1 se sugiere la reinstalación del software.

- ¿Se ha especificado el equilibrio aceptable entre atributos de calidad contradictorios, por ejemplo entre robustez y corrección?

  No se hicieron apreciaciones específicas al respecto al definirse los atributos de calidad del sistema en el Ítem 5.1.1.

- ¿Se ha especificado el límite de memoria?

  En el Ítem 6.8.2 se define un tamaño de memoria RAM de 128 Mb.

- ¿Se ha especificado el límite de almacenamiento?

  En el Ítem 6.8.2 se define en 10 Mb el espacio libre en disco estimado.

- ¿Se ha incluido la definición de éxito? ¿Y de fallo?

  Se definieron claramente las pruebas de aceptación del sistema en 6.8.3. En este caso particular de un software simple, la definición de fallo es tácita y por oposición sería que no cumpliera alguna de las pruebas de aceptación.

- ¿Se ha especificado la mantenibilidad del sistema, incluyendo la habilidad para responder a cambios en el entorno operativo, interfaces con otro software, precisión, rendimiento y otras capacidades adicionales predichas?

  Dentro de la definición de atributos de calidad (5.1.1) se establece el requisito que la interfaz visual pueda cambiarse por otra si se adoptara otra biblioteca gráfica. Asimismo, se define la portabilidad esperada para el software a desarrollar.

6.9.2 Completitud de los requisitos

- En aquellos puntos en los que no hay información disponible antes de que comience el desarrollo, ¿se han especificado las áreas de incompletitud?

  Aquellos puntos en los cuales la definición depende de la arquitectura y tecnología adoptada, fueron señalados específicamente con leyendas del tipo "Se terminará de definir en el diseño". Ello lo podemos encontrar en casos de uso como Pensar Jugada y clases de interfaz como son Manifestación de error y Manifestación de resultado (Ítem 6.5.1).

- ¿Son los requisitos completos en el sentido de que si un producto satisface todos los requisitos, será aceptable?

En este caso particular, donde el componente visual es parte fundamental del resultado final, para responder a esta pregunta, será necesario superar la etapa de diseño, al menos, lo que se refiera a la interfaz del usuario.

- ¿Surgen dudas acerca de alguna parte de los requisitos? ¿Son algunas partes del sistema imposibles de implementar y se han incluido simplemente para satisfacer al cliente?

  Salvo lo mencionado en la respuesta previa, no existen dudas y no se consideran partes imposibles de implementar.

6.9.3 Calidad de los requisitos

- ¿Están todos los requisitos escritos en el lenguaje del usuario? ¿Piensan eso los usuarios?

  Los mismos han pasado las revisiones previstas, los diagramas de casos de uso y diagramas de secuencia facilitaron este aspecto.

- ¿Evitan todos los requisitos conflictos con otros requisitos?

  Al encararse la definición de requisitos funcionales por casos de uso, se facilita que los mismos tengan definiciones que entren en conflicto entre sí. El resto de los requisitos no presenta conflictos.

- ¿Evitan los requisitos especificar el diseño?

  Ningún requisito especifica diseño.

- ¿Están los requisitos a un nivel bastante consistente? ¿Debería especificarse algún requisito con más detalle? ¿Debería especificarse algún requisito con menos detalles?

  Revisión mediante, el resultado final al completar el proceso de análisis dejó especificaciones por completar exclusivamente en aquellos casos dependientes de definiciones de diseño.

- ¿Son los requisitos lo suficientemente claros para poderse enviar a un grupo independiente para su implementación y que lo entiendan?

  Si. Esto presupone que los integrantes del grupo tengan conocimientos básicos de interpretación de diagramas UML.

- ¿Es cada requisito relevante al problema y a su solución? ¿Se puede trazar cada uno al origen en el entorno del problema?

  En el Ítem 6.7.2.2 Especificación de Requerimientos Funcionales vs. Modelo de Casos de Uso se presenta una tabla en la cual a cada requerimiento funcional, se asigna el correspondiente caso de uso especificado.

- ¿Es cada requisito comprobable? ¿Sería posible para un grupo independiente de pruebas determinar si cada requisito se ha satisfecho?

  Se definieron en el Ítem 6.8.1 el alcance de las pruebas a realizar, y en ella cada caso de uso considerado que sin ninguna duda podrían ser realizadas por un grupo independiente

- ¿Se han especificado todos los cambios posibles a los requisitos, incluyendo la probabilidad de cada cambio?

Por las características del proyecto, se han especificado los cambios en los documentos en la gestión de configuración (Ïtem 5.3.1) sin hacer distinción respecto

6.10 Aprobación del Análisis del Sistema de Información

6.10.1 Presentación y Aprobación del Análisis del Sistema de Información

Luego de la correspondiente revisión, se da por aprobado el Análisis del Sistema.

7 Diseño del Sistema de Información

7.1 Definición de la Arquitectura del Sistema

La presente actividad tiene como objetivo principal definir la arquitectura del sistema a desarrollar abarcando las particiones físicas del sistema, la descomposición lógica en subsistemas de diseño y la ubicación de cada subsistema en cada partición.

7.1.1 Definición de Niveles de Arquitectura

Simplemente existe un único nivel de arquitectura del sistema. Por lo tanto, existe un único nodo. Dentro del nodo coexisten dos componentes, el software a desarrollar y la biblioteca gráfica OpenGL. Se descarta DirectX (Biblioteca 3D desarrollada por Microsoft, dada que no cumple el requisito de portabilidad a otras plataformas). OpenGL (podría traducirse como "biblioteca de gráficos abierta") es el nexo entre el software y la interfaz de hardware gráfico, que consta de más de 100 funciones básicas que especifican las operaciones necesarias para producir aplicaciones gráficas 3D. OpenGL es independiente del hardware con lo cual, resulta apropiado para el desarrollo de sistemas multiplataformas. Al respecto, en su página web (http://www.opengl.org/) puede leerse el lema de la organización: "from games to virtual reality, mobile phones to supercomputers". Como desventaja puede mencionarse que los objetos que permite modelar son simples (puntos, líneas, polígonos), con lo cual el uso de la biblioteca en el entorno de programación se realizará a partir de un componente que facilite operaciones de alto nivel (GLScene, se describe en el próximo apartado). OpenGL viene incluida en el sistema operativo Windows XP y Windows 2000, con lo cual, no será necesario tenerlo en cuenta en la instalación.

Con lo expuesto, el diagrama de componentes queda reducido a un solo nodo que representa al hardware del usuario e incluye el software resultante del proyecto y la biblioteca OpenGL, como puede observarse a continuación (Ilustración 7-1).

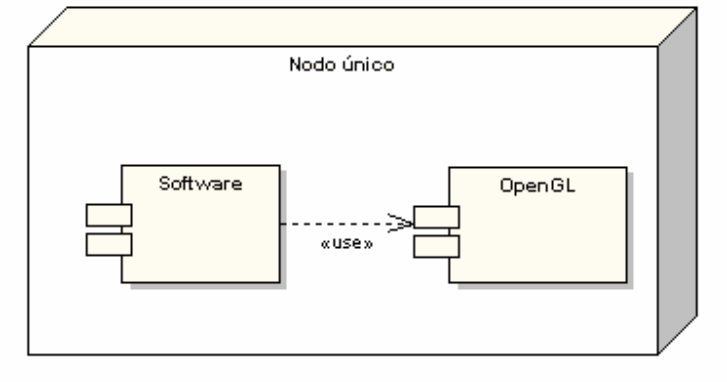


Ilustración 7-1 - Diagrama de Componentes

7.1.2 Identificación de Requisitos de Diseño y Construcción

Dada la arquitectura elegida, se presenta una lista de aquellas herramientas o componentes necesarios para el proceso de desarrollo.

7.1.2.1 Lenguaje de Programación

La experiencia del tesista en el desarrollo de aplicaciones bajo entorno Windows, permitó decidir que la construcción del software se realice con Delphi (cuyo fabricante es Borland), un entorno que compila el lenguaje Pascal orientado a objetos (object Pascal). El uso de la versión 7.0 asegura que el producto pueda migrarse a entorno Linux, sin necesidad de reescribir el código. El uso de bibliotecas 3D debe adoptarse externamente, dado que la instalación de Delphi v. 7 no cuenta con este tipo de componentes. Existe un producto denominado Lazarus (http://www.lazarus.freepascal.org) que cuenta con un set de herramientas y entorno similar al de Delphi, pero de uso gratuito, que compila código Free Pascal (http://www.freepascal.org/). La ventaja de Free Pascal es que compila código para Intel x86, Amd64, x86 64, PowerPC y Sparc, es decir que extiende la capacidad multplataforma de Delphi aumentando la portabilidad del producto final. Sin embargo, la versión de Lazarus vigente (0.9.10), aun no asegura un entorno estable. Las pruebas de migración de Delphi a Lazarus realizadas, no implicaron demasiado esfuerzo, con lo cual, una vez finalizado el desarrollo en Delphi, podrá probarse nuevamente su compilación bajo este entorno de distribución libre y fuentes abiertas, probablemente sobre versiones más estables.

7.1.2.2 Componente OpenGL

Previamente se describió la biblioteca OpenGL y se introdujo la necesidad de acceder a la misma a través de un componente que facilite operaciones de alto nivel.

Traducido de su página web (http://glscene.sourceforge.net/index.php), podemos leer "GLScene es una biblioteca 3D basada en OpenGL para Delphi. Provee objetos y componentes visuales permitiendo su descripción y la ejecución de escenas 3D de manera sencilla y eficaz a la vez." GLScene es un componente de distribución gratuita y fuentes abiertas. La documentación disponible es apropiada para la utilización de las herramientas, además existen tutoriales completos y foros de discusión tanto en inglés como en español, que permitirán utilizar las herramientas de forma adecuada.

El componente GLScene soporta ambos entornos, Delphi y Lazarus. Por ello es que se decidió efectuar un grupo de pruebas de tecnología que ayudaron a determinar su aceptación. Las mismas se describen a continuación:

a) Pruebas básicas:

- Crear un cubo en tiempo de diseño y observarlo en la pantalla en tiempo de ejecución.
- Pintar el cubo de color negro en tiempo de diseño.
- Cambiar el color del cubo a rojo en tiempo de ejecución al presionar un botón externo.

- Cambiar la posición del cubo al presionar un botón externo a la escena. No es una traslación, dado que el objeto desaparece de la posición inicial para aparecer en la siguiente.
- Presentar el cubo girado en un ángulo dado al presionar un botón externo a la escena. No es una rotación por igual razón que el caso anterior.

b) Capturas de eventos sobre los objetos, traslaciones y rotaciones:

- Cambiar el color del cubo a rojo al hacer clic sobre el mismo.
- Trasladar en forma animada al cubo al hacer clic sobre el mismo.
- Rotar en forma animada al cubo al hacer clic sobre el mismo.

Cabe aclarar que los objetos visuales por sí mismos no generan un evento al ser indicados por el click del ratón, sino, el que lo hace es el componente TGLSceneViewer, devolviendo las coordenadas x, y del punto. Ya dentro de ese evento, podemos pedirle al componente TGLSceneViewer que indique cual ha sido el objeto indicado con GetPickedObject(x, y) y nos devolverá un objeto del la clase TGLCustomSceneObject. A partir de ahí, con el nombre del objeto devuelto, se podrá identificar sobre qué objeto interactuó el usuario.

Usar objetos de formas libres (TGLFreeForm para GLScene):

- Repetir los casos anteriores reemplazando al cubo por un objeto 3D de forma irregular obtenido de los ejemplos de GLScene. El formato elegido es el que genera el 3D Studio Max (archivos con extensión .3ds).

El análisis de las pruebas da como conclusión, que el comportamiento del componente es adecuado habiendo explorado sólo algunas de las funcionalidades básicas.

7.1.2.3 Editor de Objetos 3D

Es una necesidad para este proyecto contar con un editor de objetos 3D. Esto se debe a que las formas geométricas básicas provistas por GLScene (cubo, esfera, conos, cilindros, planos, toroides), no cubren lo necesario para la composición de las piezas.

Tratando de evitar la complejidad de productos como Autocad, 3D Studio Max y Rhinoceros entre otros, se evaluó una herramienta denominada Wings 3D (http://wings3d.com). Wings3D es de uso gratuito y fuentes abiertas y cuenta con manual de usuario completo, foros y tutoriales adecuados.

Wings 3D es un producto de baja complejidad de uso, que permite la creación de objetos complejos a partir de pocas funciones básicas aplicadas sobre objetos regulares (cubos, esferas, conos, cilindros). Mediante divisiones de lados, uniones entre los puntos generados, cortes y borrados de sectores, se pueden realizar objetos geométricos complejos (como es el caso de las piezas) en forma relativamente sencillas.

En la revisión descrita, se observó y comprobó el funcionamiento del componente de audio a través del cual se incluirán las emisiones de sonidos.

Por lo anterior, luego de pruebas satisfactorias realizadas junto a la posibilidad de exportar a formato 3D Studio, se ha seleccionado a Wings 3D (versión 0.98.26) para el diseño de los objetos del juego.

7.1.2.4 Herramientas de Diseño

Para el diseño se utilizará el entorno mismo del lenguaje de programación. Para que las definiciones se puedan realizar adecuadamente, se utilizará una herramienta de generación de documentación automática denominada JADD (Just Another Delphi Doc) de distribución libre y fuentes abiertas (www.delphidoc.sourceforge.net). La misma permite además la generación de diagramas de clases. Su funcionamiento se basa en un "parser" que organiza la estructura de unidades y clases, incluyendo los comentarios asociados como descripciones. También asocia las descripciones de campos de clases (atributos), propiedades, funciones y sus resultados, procedimientos y parámetros. Se desechó la herramienta utilizada en la etapa de diseño (Enterprise Architect) debido a errores en el "parser" de código en Delphi de la misma.

7.1.3 Especificación de Excepciones

Dadas las particularidades del desarrollo, la excepción detectada como posible es que el sistema se abandone antes de finalizar una partida o se apague el equipo. En este tipo de situación, al iniciar nuevamente el software cabría la posibilidad de poder reanudar la partida al momento en que se cerró la aplicación por última vez o empezar una nueva partida. Se especifica esta última opción, por lo cual al arrancar el programa, siempre se iniciará una nueva partida. Sin embargo, se asegurará que aunque la partida haya quedado inconclusa, el registro de la misma quede almacenado hasta el momento en que se abortó.

7.1.4 Especificación de Estándares y Normas de Diseño y Construcción

El uso de unos lenguajes de programación que dispone de tantas facetas como Delphi, sugiere el uso de guías que sirven a la claridad del código fuente generado. Existen guías disponibles por Internet, pero se ha elegido "Delphi 4 Developer's Guide Coding Standards Document" (http://www.econos.de/delphi/cs.html#GeneralRules) de Xavier Pacheco and Steve Teixeira. En ella se puede encontrar desde cómo nombrar al archivo, pasando por cómo identar un lazo de control hasta con qué letra iniciar el nombre de los componentes.

Por razones de usos y costumbre, menor longitud de palabras y universalidad, los nombres de clases, variables, propiedades, operaciones, entre otros, estarán en idioma inglés.

7.1.5 Identificación de Subsistemas de Diseño

El objetivo buscado en esta subdivisión del diseño es la de facilitar el mantenimiento. Por ejemplo, si dado el caso en futuras versiones, el soporte de OpenGL no es el adecuado para nuevos sistemas operativos, y se decide migrar a otra biblioteca 3D, solamente deberán reemplazarse los subsistemas que hacen referencia a dicha biblioteca y no el

resto. Para ello, el patrón de diseño que mejor se aproxima al problema es el Modelo-Vista-Controlador. A continuación se presenta el diagrama que subdivide el sistema desde el punto de vista del diseño del software (Ilustración 7-2):

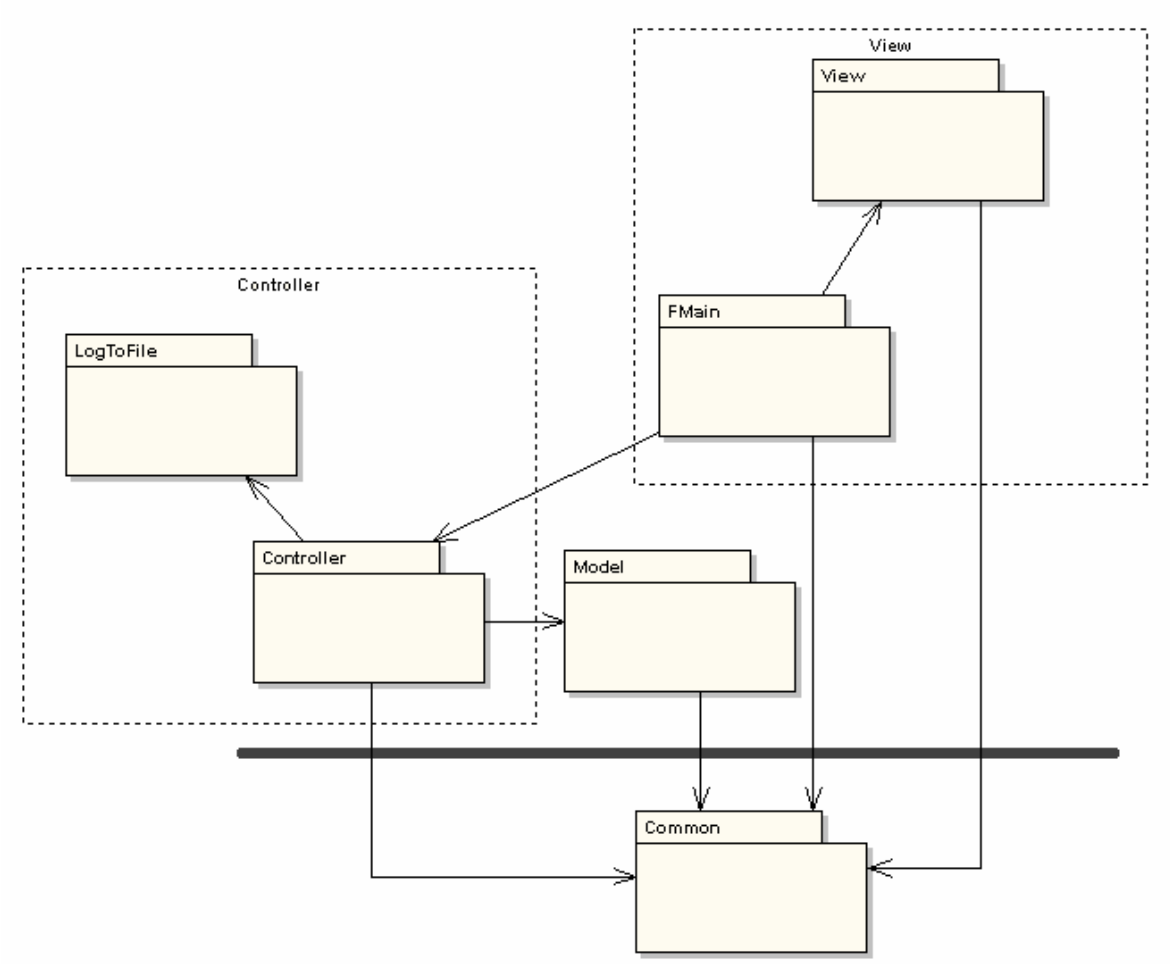


Ilustración 7-2 - Diagrama de Subsistemas de Diseño

En el diagrama se incluyen los subsistemas de diseño previstos, divididos por una línea horizontal entre aquellos que hacen a la funcionalidad del software y el de soporte. La línea punteada rodea a lo que el patrón Modelo-Vista-Controlador reconoce como Vista, que ha sido subdividido para incluir la separación que se manifiesta entre lo que va a ser el formulario principal de interacción (FMain) y el contenedor de las clases específicas del proyecto. También se observa dentro de la línea punteada Controller al Controller propiamente dicho y a LogToFile que contendrá la clase para el registro de los sucesos. Cada uno se describe a continuación:

- **Common:** Se incluyen constantes y funcionalidades a usar por el resto de los subsistemas.
- **Controller:** Este será el subsistema que interactúa entre la vista y el modelo. Su clase principal que en análisis llamamos Controlador, recibe los mensajes de

interacción del usuario y los canaliza por sí mismo o con ayuda del modelo. Asimismo cubrirá las tareas de tutoría del sistema. De esta forma, al tener que comunicarse por las acciones del usuario o ausencia de ellas, con FMain, deberá hacerlo a través de eventos que dispara y que el receptor deberá suscribir con procedimientos que den respuesta al entorno del juego.

- **FMain:** Será el formulario principal (y único) que se diseñará directamente desde el entorno que provee el lenguaje de programación.
- **LogToFile:** Contendrá la clase que se encargará de almacenar la información necesaria en cada momento en el archivo de trazado.
- **Model:** En este subsistema se definen las clases abstractas y sus funcionalidades, desde las reglas del juego hasta la obtención de jugadas posibles.
- **View:** Será el contenedor de las clases visuales que no soporta GLScene y que son específicas del sistema.

Una vez definidos los subsistemas, definimos cómo es su interacción: Las flechas indican que un subsistema (el que indica) "usa" a otro (el que es indicado). Se observa que todos (salvo LogToFile) hacen uso del subsistema auxiliar Common, para compartir definiciones y procedimientos comunes a todos ellos. View le provee a FMain de las clases visuales. Controller usa a Model para desentenderse la definición de las clases abstractas del juego y sus reglas; también usa a LogToFile para guardar el registro en cada momento. Finalmente, FMain usa a Controller proveyéndolo de todos los eventos del usuario. Desde el punto de vista del lenguaje de desarrollo, existirá una unidad (unit) que compone el proyecto para cada subsistema, a excepción de FMain que será un formulario (form).

7.1.6 Entorno Tecnológico del Sistema

Podemos subdividir el entorno tecnológico en Hardware y Software. Excluimos comunicaciones por quedar definido el sistema en un único nodo.

- **Hardware:**
  - Procesador: Pentium IV / AMD 2400 MHz
  - RAM: 512 Mbytes
  - Disco Rígido con Espacio disponible de 10 Mbytes
  - Placa aceleradora de video
- **Software**:
  - Sistema operativo Windows 2000 o XP.
  - Librerías gráficas 3D OpenGL

Se destaca que el sistema funcionará con menores prestaciones de hardware, sin embargo, se relativiza la utilidad del software en caso de no contar con una placa aceleradora de video dada la lentitud en el repintado de los componentes visuales.

7.2 Subsistema de Soporte

El subsistema de soporte Common, será extremadamente simple constituyendo por sí mismo una unidad (Common.pas) que contendrá:

- **Nombres de Archivos y Directorios**: Se definen los nombre de los archivos de configuración y directorios.
- **Prefijos de Clases**: El prefijo asignado a cada clase visual que permita estandarizar los nombres de los objetos visuales y que será completado con el índice que corresponde al objeto dentro del array de objetos (piezas, casilleros). Entre el prefijo y la clase del objeto, la relación es de uno a uno.

  Por ejemplo: Para las piezas, el nombre de la clase será **TGLPiece**, y cada instancia será nombrada 'GLPiece' seguido de '_' y el Id de la Pieza abstracta (Tpiece) que le dio origen. El guión será devuelto por las funciones de conversión como se indica en el próximo ítem.
- **Funciones de conversión**: Aquellas que permitirán componer el nombre de un objeto visual y obtener su prefijo o su índice.
  - getNameStr: Dado el prefijo de clase y el Id del Objeto, devuelve el Nombre.
  - extractNameStr: Dado el Nombre del Objeto devuelve el prefijo de clase.
  - extractNameNum: Dado el Nombre del Objeto devuelve el Id del Objeto.
- **Arrays de definiciones**: Son aquellos que establecen el color de cada pieza, el color de cada lado de pieza, el borde de casilleros. El modelo y la vista usarán estas mismas definciones, evitando así errores posibles en la duplicación de las asignaciones mencionadas.

Cabe aclarar que el uso de prefijo de clases y se adoptó para aislar realmente el controlador de la vista. De esta manera, no importará qué clase use la vista para un objeto visual dado, mientras respete el nombre del objeto. Por lo tanto, en los mensajes entre ellos, en lugar de enviarse las referencias a los objetos visuales, se enviarán directamente los nombres y cada uno entenderá de qué objeto se trata haciendo uso de las funciones de conversión previamente mencionadas.

7.3 Resultado del Diseño de Clases

A continuación se presenta el resultado generado por la herramienta de diseño y desarrollo utilizada. Por cada unidad definida, se presentarán las definiciones generales de la misma, el diagrama de clases si corresponde (en el caso de Common, no habrá diagrama) y las definiciones correspondientes a cada una de las clases definidas.

Aclaración: Las definiciones de unidades y clases que se observan más adelante en esta sección, fueron procesadas con la herramienta para documentación del diseño y desarrollo, luego de la construcción del software. Esto significa que la información de diseño se ha extendido a fin de incluir secciones como "Used by" (usado por) que

enriquecen la descripción de Unidades y Clases presentadas, aunque resulten avanzadas para el proceso de diseño.

Se extrae del reporte el resumen general de unidades:

- Independent Units
    - **Common**
    - **Controller**
    - **FMain**
    - **LogToFile**
    - **Model**
    - **View**

Se extrae del reporte el resumen general de clases que incluye el árbol de herencias y que no discrimina la unit en la que fue definida cada una de las clases:

- System.TObject
    - **TBoard**
    - **TBorderReference**
    - **TController**
    - **TControllerConfig**
    - **TCurrentInfo**
    - **TDynamicTimeout**
    - **TGLActionToDo**
    - **TGLBoard**
    - **TLogFile**
    - **TMove**
    - **TMovementToDo**
    - **TPiece**
    - **TPieceSides**
    - **TPlayer**
    - **TSides**
    - **TSquare**
    - **TStatus**
- TForm
    - **TMainForm**

- TGLActor
    - **TGLBasicForm**
        - **TGLDynamicForm**
            - **TGLPiece**
        - **TGLRotationArrows**
        - **TGLSide**
            - **TGlBorder**
            - **TGLBottomSide**
            - **TGLLeftSide**
            - **TGLRightSide**
            - **TGLTopSide**
        - **TGLSquare**
            - **TGLPlayFieldSquare**
            - **TGLStartPositionSquare**
        - **TGLTurnLeftArrow**
        - **TGLTurnRightArrow**
- TObjectList
    - **TGLActionToDoList**
    - **TGLBorders**
    - **TGLPieces**
    - **TGLPlayFieldSquares**
    - **TMovementToDoList**
    - **TPieces**
    - **TSquares**

7.3.1 Definiciones generales

7.3.1.1 Nombres utilizados

Al pasar del análisis al diseño se realizó una transformación en los nombres de clases, atributos y operaciones del idioma español al inglés. Para facilitar la comprensión, se agregaron en las definiciones marcas que indican cuál era el respectivo nombre asignado en el análisis de clases. Por ejemplo: La clase Pieza ahora será definida como TPiece. Por lo tanto, cuando se defina TPiece, se incluirá la leyenda <A.C.: Pieza> indicando así cual era su denominación en el análisis de clases. En los casos relevante, se incluye <A.C.: No Definida> para indicar que es producto del diseño.

7.3.1.2 Referencias

La visibilidad de cada miembro (campos, propiedades, procedimientos, funciones) es indicada por la herramienta con una referencia según se describe a continuación:

[-] Private

[#] Protected

[+] Public

[^] Published

[ro] Read Only

[ui] Unit Interface

[ul] Unit Local

7.3.2 Diseño de Unidades

Para favorecer la legibilidad del presente documento, el diseño de las unidades se incluye en el Anexo III – Diseño de Unidades y Clases.

7.4 Verificación y Aceptación de la Arquitectura del Sistema

Durante esta actividad se realizaron las siguientes tareas:

- **Verificación de las Especificaciones de Diseño**: Se revisaron las especificaciones de diseño y el resultado generado en el análisis de clases y se comprobó el cumplimiento de las mismas.
- **Análisis de consistencia de las Especificaciones de Diseño**: Se revisaron una por una las clases, con sus propiedades y operaciones, definidas en el análisis, verificándose que todas tuviesen su correlación en las clases de diseño, verificándose que todas se encontraran. Para esto fue de ayuda la asignación previa en cada clase, propiedad y operación de diseño, incluir la referencia del origen correspondiente en el análisis.
- **Aceptación de la Arquitectura del Sistema**: Se da por aceptada la arquitectura del sistema.

7.5 Generación de especificaciones de construcción

En esta actividad, se completan las especificaciones realizadas en 7.1 - Definición de la Arquitectura del Sistema, dado que la forma de encarar el presente proyecto, en la definición de la arquitectura, quedó prácticamente definida la construcción. Ya se definió en ella el lenguaje de programación, las librerías de soporte gráfico con las cuales se realizaron pruebas de tecnología, se utilizaron nombres de clases normados y se definieron las clases: sus campos, propiedades, procedimientos y funciones en el mismo entorno que se utilizará durante la programación (Delphi 7). Por lo anterior, también quedaron definidas las unidades de contrucción, es decir:

- Common.pas
- Controller.pas
- FMain.pas
- LogToFile.pas
- Model.pas
- View.pas

Como guía se detallan los pasos a seguir en la construcción de unidades componentes del sistema:

- Construcción de FMain.pas conteniendo los componente visuales de GLScene definidos
- Implementación de funciones de LogToFile.pas
- Implementación de funciones de Common.pas
- Implementación de funciones y procedimientos de Model.pas
- Generación de los archivos en formato 3ds que serán utilizados por el sistema
- Implementación de funciones y procedimientos de View.pas
- Implementación de funciones y procedimientos de FMain.pas relacionados con View.pas
- Implementación de funciones y procedimientos de Controller.pas
- Implementación de funciones y procedimientos (en particular para la suscripción a los eventos definidos) de FMain.pas relacionados con Controller.pas

7.6 Especificación Técnica del Plan de Pruebas

A continuación se detallan las pruebas a realizar durante el proceso de construcción del software de acuerdo a lo especificado durante el análisis. Asimismo se definen aquellos datos que deberán contener las planillas de pruebas realizadas. El entorno de hardware y software especificado coincide con definido durante la especificación de requisitos del sistema. Las pruebas propuestas son exhaustivas y se repiten para cada pieza o casillero en cada caso, debido a que cada uno de ellos es definido individualmente y se podría cometer un error que dejaría al juego inutilizable. Se sugiere configurar los tiempos de espera de ayuda en 5 segundos para el coeficiente constante para no demorar su realización en cada estado como se muestra a continuación:

```
Antes_de_Seleccionar_la_Pieza_ConstantCoef=5
Antes_de_Seleccionar_la_Pieza_LinearCoef=0
Antes_de_Seleccionar_la_Pieza_QuadraticCoef=0
Pieza_Seleccionada_ConstantCoef=5
Pieza_Seleccionada_LinearCoef=0
Pieza_Seleccionada_QuadraticCoef=0
Casillero_Seleccionado_ConstantCoef=5
Casillero_Seleccionado_LinearCoef=0
Casillero_Seleccionado_QuadraticCoef=0
```

Las pruebas a realizar serán las siguientes (Tablas 7-1, 7-2, 7-3, 7-4 y 7-5):

- **Inicio de la Partida**: Verificar la correcta posición inicial de piezas en el tablero:

Acción	Efecto
Iniciar la partida como jugador de piezas blancas	Las Piezas se muestran en posición inicial correcta
Iniciar la partida como jugador de piezas negras	Las Piezas se muestran en posición inicial correcta

Tabla 7-1 - Pruebas de Inicio de la Partida

- **Selección de Pieza**: Por cada pieza del jugador de piezas blancas realizar los siguientes movimientos:

Acción	Efecto
Seleccionar la Pieza	La Pieza se destaca desplazándose verticalmente
Cancelar la selección eligiendo el casillero de posición inicial	La Pieza regresa a su posición original
Seleccionar nuevamente la Pieza	La Pieza se destaca desplazándose verticalmente
Cancelar la selección al elegir otra Pieza	La Pieza elegida inicialmente regresa a su posición original y la otra se destaca desplazándose verticalmente
Seleccionar Objetos inapropiados (bordes, casilleros de juego, piezas del oponente)	Se manifiesta el error en el objeto y en forma sonora
Esperar ayuda de pieza	Al cumplirse el tiempo de espera se observa la ayuda sonora y visual sobre una pieza adecuada

Tabla 7-2 - Pruebas de Selección de Pieza

Repetir secuencia previa para todas las piezas negras iniciando la partida como jugador de dichas piezas.

- **Selección de Casillero**: Por cada casillero, iniciando la partida con piezas blancas, realizar los siguientes movimientos (de ser necesario, reiniciar la partida):

Acción	Efecto
Seleccionar una pieza que pueda jugarse en el casillero **sin necesidad de rotarse**	La Pieza se destaca desplazándose verticalmente

Seleccionar el Casillero posible	La Pieza se desplaza horizontalmente hasta quedar sobre el casillero indicado
Seleccionar objetos Inapropiados	Se manifiesta el error en el objeto y en forma sonora
Cancelar la selección de la pieza seleccionando un casillero de posición inicial	La pieza se desplaza horizontalmente hasta colocarse sobre su posición de origen y luego verticalmente hasta su posición inicial
Seleccionar una pieza que pueda jugarse	La Pieza se destaca desplazándose verticalmente
Esperar ayuda de casillero	Al cumplirse el tiempo de espera se observa la ayuda sonora y visual sobre un Casillero adecuado
Seleccionar el Casillero indicado por la ayuda	La Pieza se desplaza horizontalmente hasta quedar sobre el Casillero indicado
Seleccionar otro Casillero posible	La Pieza se desplaza horizontalmente hasta quedar sobre el nuevo Casillero indicado
Seleccionar otro Casillero donde no se pueda colocar la pieza	Se manifiesta el error en el objeto y en forma sonora

Tabla 7-3 - Pruebas de Selección del Casillero

Repetir secuencia previa para todas las piezas negras iniciando la partida como jugador de dichas piezas.

- **Rotación y Aceptación de Pieza**: Por cada pieza, iniciando la partida con piezas blancas, realizar los siguientes movimientos (de ser necesario, reiniciar la partida):

Acción	Efecto
Seleccionar una pieza que pueda jugarse en el casillero **sin necesidad de rotarse**	La Pieza se destaca desplazándose verticalmente
Seleccionar el Casillero posible	La Pieza se desplaza horizontalmente hasta quedar sobre el casillero indicado
Aceptar la pieza	La Pieza se posa sobre el casillero de juego elegido desplazándose verticalmente
Seleccionar una pieza que pueda jugarse en el Casillero **si se la rota previamente**	La Pieza se destaca desplazándose verticalmente

Seleccionar dicho Casillero	La Pieza se desplaza horizontalmente hasta quedar sobre el casillero indicado
Esperar ayuda de Rotación o Aceptación	Al cumplirse el tiempo de espera se observa la ayuda sonora y visual sobre el Indicador de Giro adecuado
Girar la pieza en el sentido indicado por la ayuda	La pieza rota en el sentido requerido, quedando a 90º de su orientación previa
Esperar ayuda de Rotación o Aceptación	Al cumplirse el tiempo de espera se observa la ayuda sonora y visual sobre el Indicador de Giro adecuado o sobre la Pieza
Si la ayuda se manifiesta sobre un indicador de giro, girar la pieza en el sentido indicado por la ayuda. Sino, Aceptar la pieza	La pieza rota en el sentido requerido, quedando a 90º de su orientación previa en caso de girarse, o se posa sobre el casillero de juego elegido desplazándose verticalmente en caso de Aceptarla
Seleccionar una pieza que pueda jugarse en el Casillero **si se la rota previamente**	La Pieza se destaca desplazándose verticalmente
Seleccionar dicho Casillero	La Pieza se desplaza horizontalmente hasta quedar sobre el casillero indicado
Aceptar la Pieza	Se manifiesta el error en la Pieza y en forma sonora

Tabla 7-4 - Pruebas de Rotación y Aceptación de Piezas

Repetir secuencia previa para todas las piezas negras iniciando la partida como jugador de dichas piezas. Las pruebas con rotación no serán necesarias en piezas con los cuatro lados de igual color.

- **Final de Partida**: Probar cada caso posible haciendo uso de la tabla de decisión (Tabla 6-3) y mostrar Manifestación de Resultados. De ser necesario, desarrollar funcionalidades que permitan guardar partidas y cargarlas a fin de facilitar estas pruebas.
- **Obtener Trazado**: Obtener el trazado de las partidas realizadas. Abrir trazados con programa para planillas de cálculos.

Acción	Efecto
Invocar las teclas de obtención de trazados <ctrl> + T	Se abre un explorador de windows en la carpeta donde se encuentran los registros de partidas

Acción	Efecto
Abrir una partida con el programa de planilla de cálculos elegido	Verificar la información.

Tabla 7-5 - Pruebas de Obtener Trazado

7.7 Establecimiento de requisitos de implantación

En esta actividad se realizan las tareas correspondientes a la documentación del usuario del sistema y a los requisitos de implantación, que en esta caso se adaptarán a las particularidades del sistema que se aparta un poco del desarrollo de sistemas habitual.

7.7.1 Especificación de Requisitos de Documentación del Usuario

Simplemente se establecerá un manual de uso para el investigador que contendrá la información correspondiente a la configuración de los parámetros del sistema. Se supone que el usuario del "juego", aprenderá a partir de las ayudas y la corrección de los errores cometidos.

7.7.2 Especificación de Requisitos de Implantación

En este aspecto, el sistema también es particular en estos aspectos, dado que su instalación probablemente se realice por los padres de los usuarios y no hay que presuponer un conocimiento avanzado de sistemas.

Se prevé entonces, la provisión de una instalación lo más automatizada posible, que permita agregar el archivo configurado por el investigador en caso que sus valores se aparten de los que se definan por defecto.

7.8 Aprobación del Diseño del Sistema de Información

7.8.1 Presentación y Aprobación del Diseño del Sistema de Información

Luego de la correspondiente revisión, se da por aprobado el Diseño del Sistema.

8 Construcción e Implantación del Sistema de Información

8.1 Preparación del Entorno de Generación y Construcción

El entorno de Generación y Construcción quedó previamente preparado en el proceso de diseño, cuando se realizaron las pruebas de tecnología y la definición de las unidades componentes del sistema, es decir, se cuenta con el "esqueleto" del Software disponible y a su vez, se dispone de la herramienta para la construcción de los objetos visuales.

8.2 Generación del Código de los Componentes y Procedimientos

A continuación se describen las tareas realizadas durante la presente actividad:

- *Implementación de procedimientos y funciones de FMain.pas* conteniendo los componente visuales de GLScene definidos:

  Se crean los objetos GLScene (administrador de objetos visuales), GLSceneViewer (contenedor visual de objetos visuales) y GLCadencer (componente que dispone de un evento "OnProgress" que se ejecuta al finalizar la presentación de cada escena que se utilizará para la animación)

- *Implementación de funciones de LogToFile.pas*:

  Se implementan las funciones auxiliares NextName y NowInFormat, el constructor y destructor de la clase y el procedimiento LogToFile de acuerdo al diseño.

- *Implementación de funciones de Common.pas:*

  Se implementan las funciones getNameStr, extractNameStr y extractNameNum que permiten aislar el Modelo del Controlador. Se definen las constantes para nombres de cada clase, las de nombres de directorios y archivos de registro y de cantidades de piezas y casilleros. Se asignan valores a los arreglos bidimensionales (colores de cada lado de casillero y piezas) y unidimensional (color de cada pieza) con sus valores correspondientes (para las piezas) y los valores iniciales de la partida (en el caso de los casilleros).

- *Implementación de funciones y procedimientos de Model.pas:*

  Se implementan sucesivamente los constructores, destructores, procedimientos y funciones de las clases TPlayer, TPiece, TPieces, TSquare, TSquares y TBoard.

- *Generación de los archivos en formato 3ds que serán utilizados por el sistema:*

  Con el uso de Wings3D, se generan los archivos correspondientes a los cuerpos visuales del juego, a saber: ArrowTurnLeft.3ds, ArrowTurnRight.3ds, Piece.3ds, Side.3ds, Square.3ds. A dichos cuerpos 3D, no se le definió ni material ni textura, dejando que dichas propiedades se administren directamente por programa.

- *Implementación de funciones y procedimientos de View.pas:*

  Se implementan las funciones auxiliares getQuadrant y getGLClassName. Se implementan sucesivamente los constructores, destructores, procedimientos y funciones de las clases TGLBasicForm, TGLSide, TGLTopSide, TGLRightSide, TGLBottomSide, TGLLeftSide, TPieceSides, TGLSquare, TGLStartPositionSquare, TGLPlayFieldSquare, TBorderReference, TGlBorder, TGLBorders, TGLDynamicForm, TGLTurnLeftArrow, TGLTurnRightArrow, TGLRotationArrows, TGLActionToDo, TGLActionToDoList, TGLPiece, TGLPieces, TGLPlayFieldSquares, TGLBoard.

- *Implementación de funciones y procedimientos de FMain.pas relacionados con View.pas:*

  Se implementa el procedimiento que captura el evento del botón de inicio de partida, creando al tablero visual FGLBoard.

- *Implementación de funciones y procedimientos de Controller.pas:*

  Se implementan sucesivamente los constructores, destructores, procedimientos y funciones de las clases TDynamicTimeout, TStatus, TMovementToDo, TMovementToDoList, TControllerConfig, TCurrentInfo, Tcontroller.

- *Implementación de funciones y procedimientos (en particular para la suscripción a los eventos definidos) de FMain.pas relacionados con Controller.pas:*

  Se implementan los procedimientos asociados con los eventos publicados por el Controlador, a saber: ShowError, ShowHelp, GameEnded, DoMovement. Se completa el procedimiento asociado al botón de inicio de partida agregando la creación del objeto FController. Se asocian dichos procedimientos a los respectivos eventos publicados por el Controlador.

8.3 Ejecución de las pruebas

En esta sección se hará un recuento resumido de las pruebas realizadas de acuerdo a lo estipulado en el proceso de diseño en el item 7.6.

Cabe acotar que además de lo observable en las pruebas que se detallarán a continuación, el proceso de construcción se realizó de forma tal de ir incorporando sucesivamente las funcionalidades a las clases, en un orden tal (detallado en 8.2) que permitió que las primeras clases implementadas eran testeadas informalmente en las sucesivas etapas, dónde las misma eran utilizadas.

También el uso informal del sistema ayudó a corregir errores, en su mayoría debido a la necesidad de buscar un sincronismo en el desenvolvimiento de las partidas, sin que por ello, resulte aburrida la espera para poder tomar una próxima acción. Las técnicas de prueba se han estandarizado mucho, sin embargo no viene mal en estos casos dejar a niños que realicen tantos clics con el mouse continuos que dejan por ejemplo a todos los objetos en estado de error, a ver cómo salen del mismo. Con las pruebas formales exclusivamente, quizás no se hubiera detectado este tipo de problemas.

8.3.1 Preparación del entorno de las Pruebas

Se realiza la preparación del entorno de pruebas de la siguiente manera:

- En la PC de desarrollo se instala el software en la sesión de otro usuario (Windows XP permite tener múltiples usuarios y es sencillo cambiar de sesión entre ellos sin necesidad de reiniciar el sistema, conservando las aplicaciones en uso al regresar a la misma).

- Se introducen en el archivo de configuración los valores sugeridos de tiempos de espera en la Especificación Técnica del Plan de Pruebas (ítem 7.6).

- Los resultados serán acompañados de imágenes que resultarán más descriptivas que una explicación al respecto.

Las pruebas fueron diseñadas en forma evolutiva. Con ello, se fue realizando cada etapa de pruebas, analizándolo y corrigiendo el software de ser necesario.

8.3.2 Realización de las pruebas

A continuación se transcribe el resumen de las pruebas realizadas.

8.3.2.1 Pruebas de inicio de partida

Acción	Efecto	Resultado
Iniciar la partida como jugador de piezas blancas	Las Piezas se muestran en posición inicial correcta	Aprobada (Ver Ilustración 8-1)
Iniciar la partida como jugador de piezas negras	Las Piezas se muestran en posición inicial correcta	Aprobada (Ver Ilustración 8-2 donde se observa el inicio de la jugada inicial de las piezas blancas)

Tabla 8-1 - Resultado de las Pruebas de Inicio de Partida

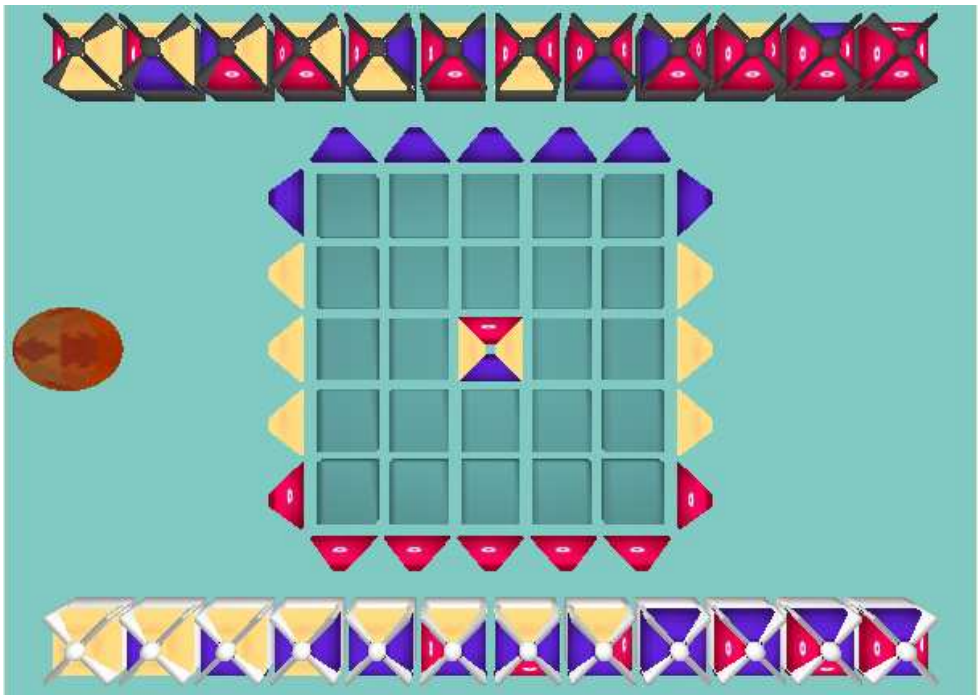


Ilustración 8-1 - Inicio de Partida con Piezas Blancas

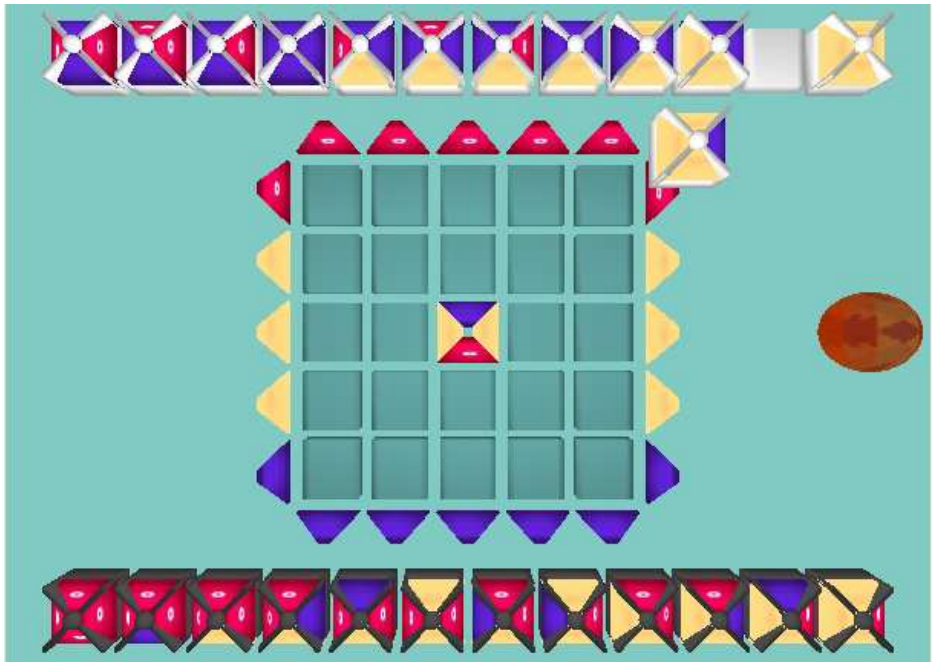


Ilustración 8-2 - Inicio de Partida con Piezas Negras

Análisis de las pruebas

El comportamiento ha sido el esperado. Las piezas se muestran en correcta posición inicial. En el segundo caso, se observa el inicio del desplazamiento de la pieza blanca, dado que la partida comienza de esa manera, con el movimiento del oponente en este caso.

8.3.2.2 Pruebas de Selección de Piezas

Se repitieron para cada Jugador y cada pieza las siguientes pruebas:

Acción	Efecto	Resultado
Seleccionar la Pieza	La Pieza se destaca desplazándose verticalmente	Aprobado (de la Ilustración 8-3 pasa a la Ilustración 8-4)
Cancelar la selección eligiendo el casillero de posición inicial	La Pieza regresa a su posición original	Aprobado
Seleccionar nuevamente la Pieza	La Pieza se destaca desplazándose verticalmente	Aprobado
Cancelar la selección al elegir otra Pieza	La Pieza elegida inicialmente regresa a su posición original y la otra se destaca desplazándose verticalmente	Aprobado (Ilustración 8-5, mientras se desplaza hacia abajo la pieza cancelada se eleva la nueva pieza elegida)
Seleccionar Objetos inapropiados (bordes, casilleros de juego, piezas del oponente)	Se manifiesta el error en el objeto y en forma sonora	Aprobado (en la Ilustración 8-6 se ve el error en el casillero seleccionado en momento inadecuado)
Esperar ayuda de pieza	Al cumplirse el tiempo de espera se observa la ayuda sonora y visual sobre una pieza adecuada	Aprobado (ver Ilustración 8-7)

Tabla 8-2 - Resultados de las Pruebas de Selección de Piezas

Capturas de Pantalla

Ilustración 8-3 - Pieza en Reposo

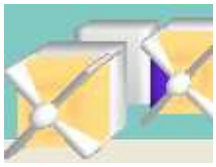


Ilustración 8-4 - Pieza Seleccionada

Ilustración 8-5 - Cambio de Selección de Pieza

Ilustración 8-6 - Manifestación de Error en un Casillero

Ilustración 8-7 - Manifestación de Ayuda sobre una Pieza Posible

Análisis de las pruebas

Se ha podido seleccionar y cancelar de distintos modos cada una de las piezas. Se pudieron observar las manifestaciones de error y de ayudas tanto visuales como sonoras en todos los casos. Los desplazamientos ocurrieron a velocidades adecuadas. La altura a la que se desplaza la pieza para quedar "destacada" fue ajustada durante la prueba quedando finalmente en la que se observa en las Ilustración 8-4 que parece apropiada.

8.3.2.3 Pruebas de Selección de Casilleros

Se realizaron las siguientes pruebas por cada casillero tanto jugando tanto con Piezas Negras como con Piezas Blancas.

Acción	Efecto	Resultado
Seleccionar una pieza que pueda jugarse en el casillero **sin necesidad de rotarse**	La Pieza se destaca desplazándose verticalmente	Aprobado (En Ilustración 8-8 se indica con una flecha el casillero donde se jugará la pieza seleccionada. La flecha es de la ilustración y no del juego)
Seleccionar el Casillero posible	La Pieza se desplaza horizontalmente hasta quedar sobre el casillero indicado	Aprobado (En Ilustración 8-9 se observa el desplazamiento y en 8-10 se observa la Pieza sobre el Casillero elegido).
Seleccionar objetos Inapropiados	Se manifiesta el error en el objeto y en forma sonora	Aprobado (En Ilustración 8-15 se observa una selección errónea.
Cancelar la selección de la pieza seleccionando un casillero de posición inicial	La pieza se desplaza horizontalmente hasta colocarse sobre su posición de origen y luego verticalmente hasta su posición inicial	Aprobado.
Seleccionar una pieza que pueda jugarse	La Pieza se destaca desplazándose verticalmente	Aprobado.
Esperar ayuda de casillero	Al cumplirse el tiempo de espera se observa la ayuda sonora y visual sobre un Casillero adecuado	Aprobado (Ver Ilustración 8-11)

Acción	Efecto	Resultado
Seleccionar el Casillero indicado por la ayuda	La Pieza se desplaza horizontalmente hasta quedar sobre el Casillero indicado	Aprobado (Ver Ilustración 8-12)
Seleccionar otro Casillero posible	La Pieza se desplaza horizontalmente hasta quedar sobre el nuevo Casillero indicado	Aprobado (En Ilustración 8-13 se observa a la pieza durante el desplazamiento y en 8-14 ya se encuentra en la posición buscada)
Seleccionar otro Casillero donde no se pueda colocar la pieza	Se manifiesta el error en el objeto y en forma sonora	Aprobado

Tabla 8-3- Pruebas de Selección de Casillero

Capturas de Pantalla

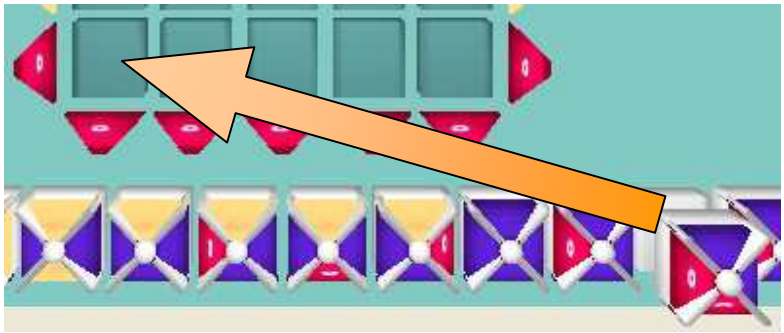


Ilustración 8-8 - Selección de Pieza a Jugar sin Necesidad de Rotarla

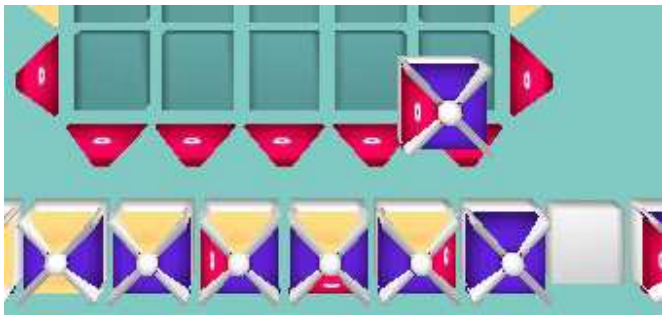


Ilustración 8-9 - Desplazamiento hacia el Casillero Seleccionado

Ilustración 8-10 - Pieza sobre Casillero Seleccionado

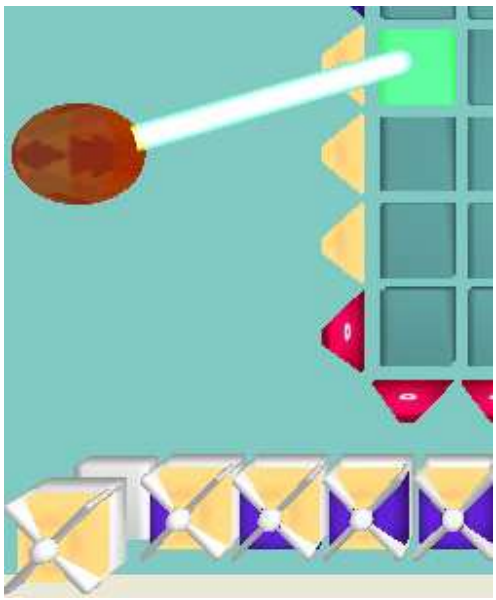


Ilustración 8-11 - Manifestación de Ayuda sobre Casillero Posible

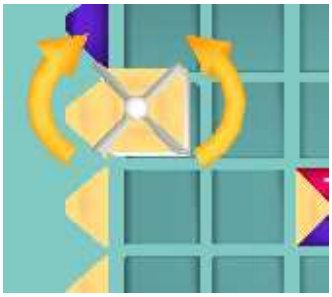


Ilustración 8-12 - Pieza Desplazada sobre el Casillero indicado por la Ayuda

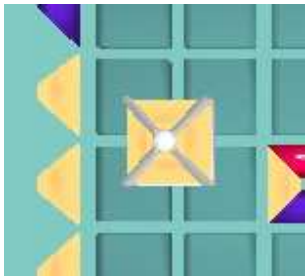


Ilustración 8-13 - Pieza en desplazamiento hacia el nuevo Casillero elegido

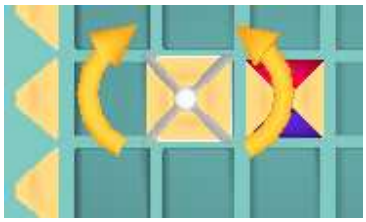


Ilustración 8-14 - Pieza sobre el nuevo Casillero elegido

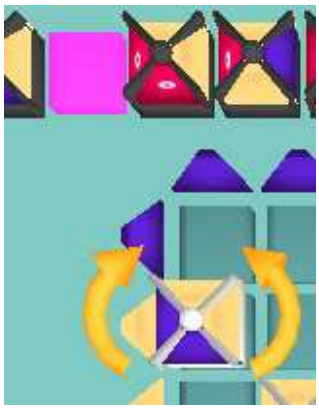


Ilustración 8-15 - Manifestación de error al elegir un Casillero libre del oponente

Análisis de las pruebas

Se ha podido seleccionar y cancelar de distintos maneras la selección de cada uno de los casilleros. Se pudieron observar las manifestaciones de error y de ayudas tanto visuales como sonoras en todos los casos.

Los velocidades de desplazamiento se ajustaron a 3 lados de pieza por segundo, lo que significa que aproximadamente una pieza tarda 1 segundo en recorrer 3 casilleros en el sentido de sus lados.

8.3.2.4 Pruebas de Rotación y Aceptación de Piezas

Acción	Efecto	Resultado
Seleccionar una pieza que pueda jugarse en el casillero **sin necesidad de rotarse**	La Pieza se destaca desplazándose verticalmente	Aprobado (ver Ilustración 8-16)
Seleccionar el Casillero posible	La Pieza se desplaza horizontalmente hasta quedar sobre el casillero indicado	Aprobado (ver Ilustración 8-17 en desplazamiento y 8-18 ya sobre el casillero apropiado)
Aceptar la pieza	La Pieza se posa sobre el casillero de juego elegido desplazándose verticalmente	Aprobado (ver Ilustración 8-19)
Seleccionar una pieza que pueda jugarse en el Casillero **si se la rota previamente**	La Pieza se destaca desplazándose verticalmente	Aprobado (ver Ilustración 8-20)
Seleccionar dicho Casillero	La Pieza se desplaza horizontalmente hasta quedar sobre el casillero indicado	Aprobado (ver Ilustración 8-21)
Esperar ayuda de Rotación o Aceptación	Al cumplirse el tiempo de espera se observa la ayuda sonora y visual sobre el Indicador de Giro adecuado	Aprobado (ver Ilustración 8-22)
Girar la pieza en el sentido indicado por la ayuda	La pieza rota en el sentido requerido, quedando a 90º de su orientación previa	Aprobado (En Ilustración 8-23 se observa la pieza en rotación)
Esperar ayuda de Rotación o Aceptación	Al cumplirse el tiempo de espera se observa la ayuda sonora y visual sobre el Indicador de Giro adecuado o sobre la Pieza	Aprobado (ver Ilustración 8-24 con ayuda sobre la pieza)
Si la ayuda se manifiesta sobre un indicador de giro, girar la pieza en el sentido indicado por la ayuda. Sino, Aceptar la pieza	La pieza rota en el sentido requerido, quedando a 90º de su orientación previa en caso de girarse, o se posa sobre el casillero de juego elegido desplazándose verticalmente en caso de Aceptarla	Aprobado (En Ilustración 8-25 se observa la Pieza Aceptada)

Acción	Efecto	Resultado
Seleccionar una pieza que pueda jugarse en el Casillero **si se la rota previamente**	La Pieza se destaca desplazándose verticalmente	Aprobado (Ver Ilustración 8-26)
Seleccionar dicho Casillero	La Pieza se desplaza horizontalmente hasta quedar sobre el casillero indicado	Aprobado (Ver Ilustración 8-27)
Aceptar la Pieza	Se manifiesta el error en la Pieza y en forma sonora	Aprobado (Ver Ilustración 8-28)

Tabla 8-4 - Pruebas de Rotación y Aceptación de Piezas

Capturas de Pantalla

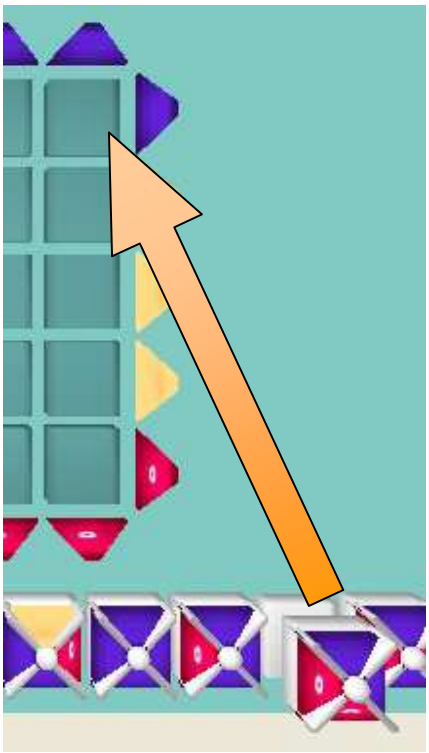


Ilustración 8-16 - Selección de Pieza a Jugar sin necesidad de ser rotada

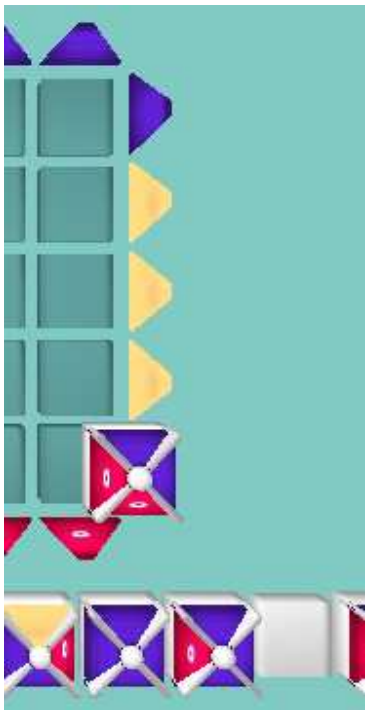


Ilustración 8-17 - Pieza en desplazamiento al casillero elegido

Ilustración 8-18 - Pieza lista a ser Aceptada

Ilustración 8-19 - Pieza Aceptada

Ilustración 8-20 - Selección de Pieza a Jugar previa rotación

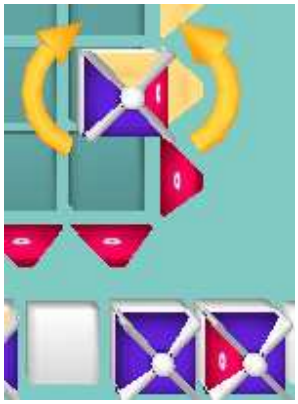


Ilustración 8-21 - Pieza sobre el casillero seleccionado

Ilustración 8-22 – Manifestación de Ayuda para Rotación de pieza en sentido horario

Ilustración 8-23 - Pieza en rotación en sentido horario

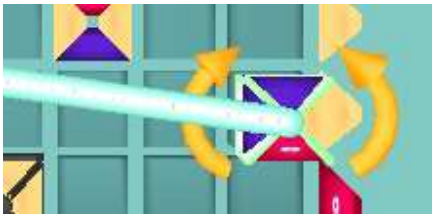


Ilustración 8-24 - Manifestación de Ayuda para aceptación de Pieza tras ser rotada

Ilustración 8-25 - Pieza Aceptada

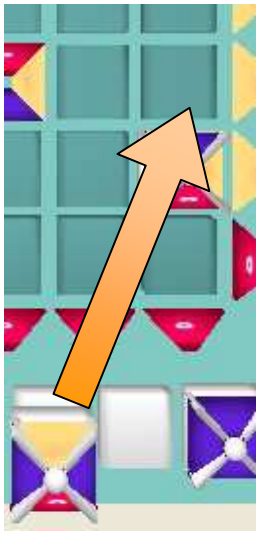


Ilustración 8-26 - Selección de pieza

Ilustración 8-27 - Pieza sobre casillero que para aceptarse necesita rotación previa

Ilustración 8-28 - Manifestación de Error al intentar Aceptar la pieza sin rotarla previamente

Análisis de las pruebas

Se ha podido rotar la pieza hacia uno y otro lado, aceptarla si el elegido era un casillero posible para la orientación actual de la pieza. Se pudieron observar ayudas tanto para girar la pieza como para aceptarla. Se comprobó la manifestación de error al intentar aceptar la pieza en un casillero donde no se puede efectuar dicho movimiento o que para poder hacerlo se necesita rotarla previamente. Las velocidades de rotación se ajustaron a 90° por segundo.

8.3.2.5 Pruebas de Final de Partida

El final de la partida corresponde verificarlo en el estado Casillero Seleccionado, es decir, que si se recibe un evento de Aceptación de Pieza por parte del usuario, se deberá verificar dicha condición. Si bien la cantidad de casos posibles (ocho) no es grande, llevar las partidas a esas condiciones, puede resultar una tarea difícil de acotar en el tiempo. Se resolvió para ello desarrollar un Simulador de Partida de la siguiente manera:

- Se desarrolló una nueva aplicación.
- Se definieron las entradas que utiliza el procedimiento EventOccursSquareSelectedSt de la clase Tcontroller en la pantalla de interfaz (Ilustración 8-29).
- Se copió el código fuente del mencionado procedimiento al correspondiente al evento onClick del botón Test.
- Se reemplazaron en el código las entradas originales por las de la interfaz de prueba (UserPieceColor, OpponentHasPlayed, UserCanPlay).
- Se reemplazó la salida en el código(FnextMoveSituation) por la de la interfaz de Prueba.
- Se modificaron las acciones de forma tal que sólo se evalúen los movimientos de pieza aceptada (es decir que la pieza está en un casillero posible y en la orientación adecuada para ser aceptada), descartándose aquellos que no son de interés evaluar (Selección de otro casillero, Selección de otra pieza, Diferentes casos de error, Rotaciones de pieza).
- Se comentaron las acciones de registro para el trazado de la partida.
- Se compila la aplicación.

De esta forma, tenemos la posibilidad de evaluar el final de la partida, sin necesidad de realizar las jugadas que llevaran a cada situación.

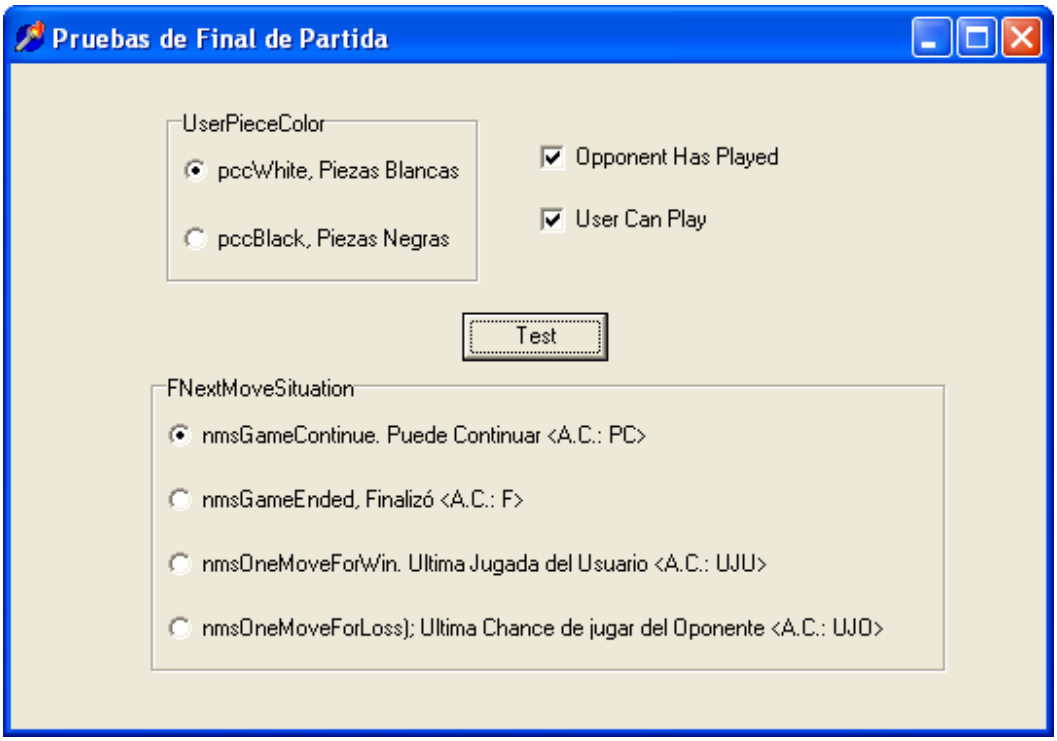


Ilustración 8-29 - Simulador para Pruebas de Final de Partida

Cabe aclarar que en la tarea de pasar del diseño a la implementación, se agregó como entrada el estado de la variable FNextMoveSituation en la Movida previa, con lo cual FNextMoveSituation se transforma en una variable de salida y entrada a la vez. Esto ocurrió debido a que lo que se diseño fue el proceso de identificación de final de partida, en cambio, al implementarse, dicho proceso forma parte de un proceso más amplio que se desata ante el evento que captura la acción del usuario cuando el estado es "Casillero Elegido". Existen solamente dos situaciones posibles de entrada, ya que tanto nmsGameeEnded como nmsOneMoveForLoss determinan que la partida ha finalizado en el primer caso o va a finalizar sin que el usuario vuelva a jugar en el segundo. Es decir, las únicas entradas posibles serán nmsGameContinue(PC) y nmsOneMoveForWin(UJU). En los casos agregados, Cualquiera sea el resto de las entradas, la situación en la movida previa fue nmsOneMoveForWin(UJU) y la nueva salida será nmsGameContinue(PC).

La tabla de los casos evaluados considerando la modificación expresada, puede verse a continuación (Tabla 8-5)

Entradas \ Casos	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Color de Pieza del Usuario	B	B	B	B	N	N	N	N	B	B	B	B	N	N	N	N
Oponente Jugó	Sí	No	Sí	No	Sí	No	Sí	No	Sí	No	Sí	No	Sí	No	Sí	No

Entradas \ Casos	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Usuario Puede Seguir Jugando	Sí	Sí	No	No	Sí	Sí	No	No	Sí	Sí	No	No	Sí	Sí	No	No
Entrada: Sit. Movida previa	PC	PC	PC	PC	PC	PC	PC	PC	UJU	UJU	UJU	UJU	UJU	UJU	UJU	UJU
Salida: Situación	PC	F	UJO	F	PC	UJU	F	F	F	F	F	F	F	F	F	F
Resultado de la Prueba	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**

Tabla 8-5 - Casos de Prueba de Final de Partida

A: Aprobado

Vale la pena aclarar que este análisis es exclusivo de la condición de final de partida. El resultado surge simplemente de comparar los puntos del usuario y del oponente virtual.

Capturas de Pantalla

A continuación se muestran algunas capturas de Pantalla (Ilustración 8-30, 8-31, 8-32, 8-33, 8-34 y 8-35) para distintas situaciones de final de partida que de manera informal complementan las pruebas que con rigurosidad se verificaron en la presente sección.

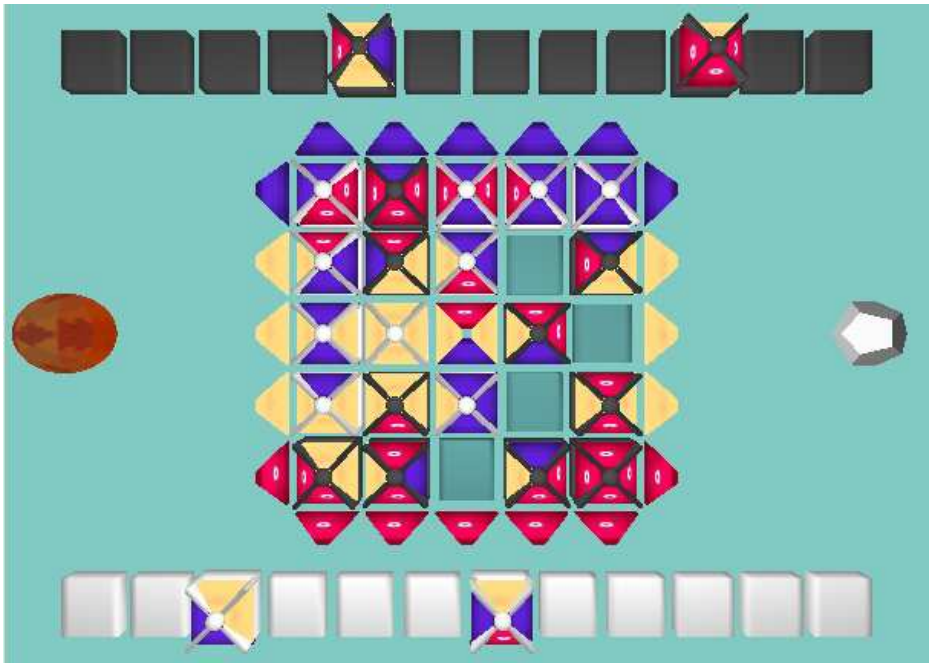


Ilustración 8-30 – Con Piezas Blancas: Empate

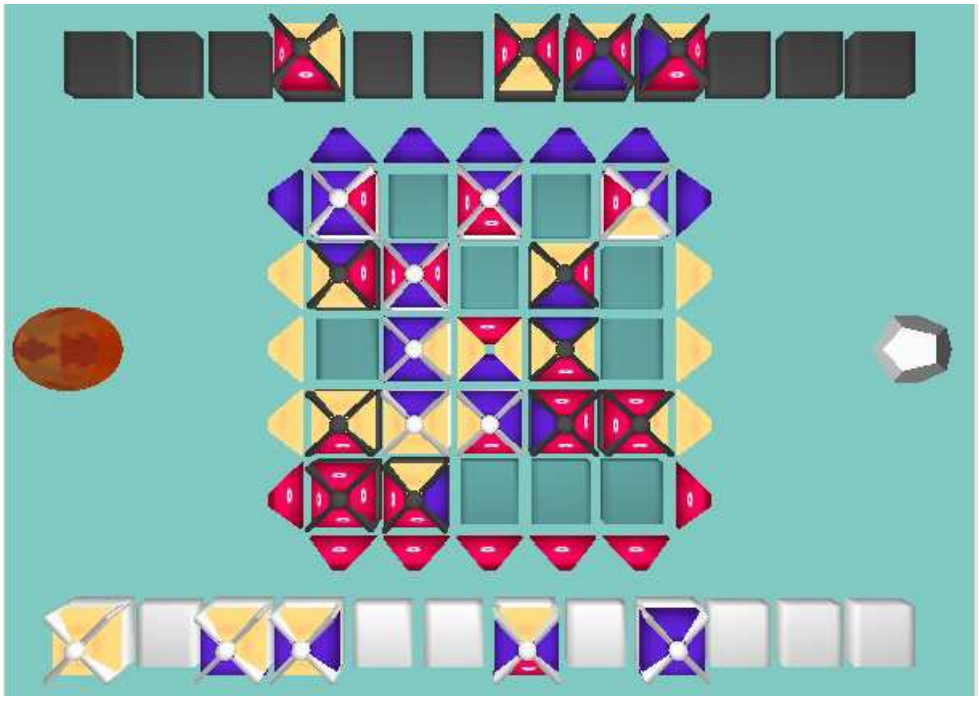


Ilustración 8-31 - Con Piezas Blancas: Negras Ganan

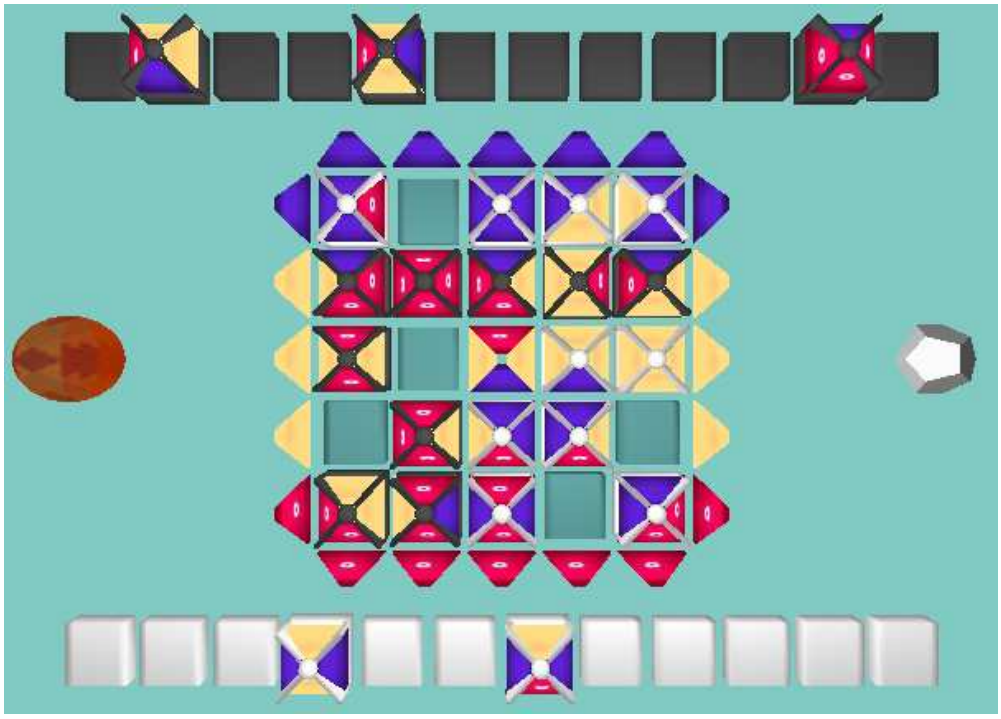


Ilustración 8-32 - Con Piezas Blancas: Blancas Ganan

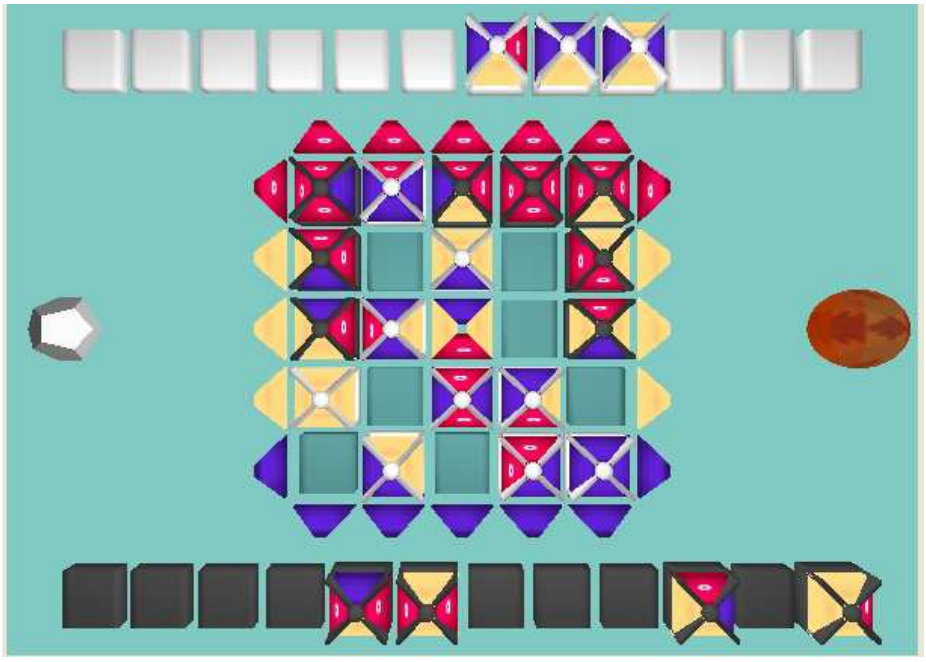


Ilustración 8-33 - Con Piezas Negras: Blancas Ganan

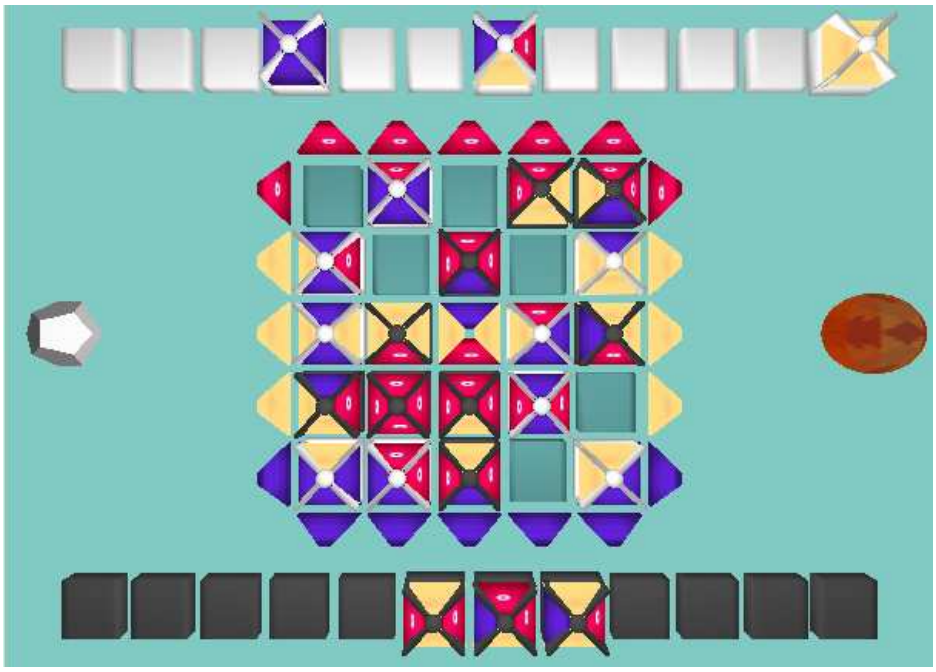


Ilustración 8-34 - Con Piezas Negras: Empate

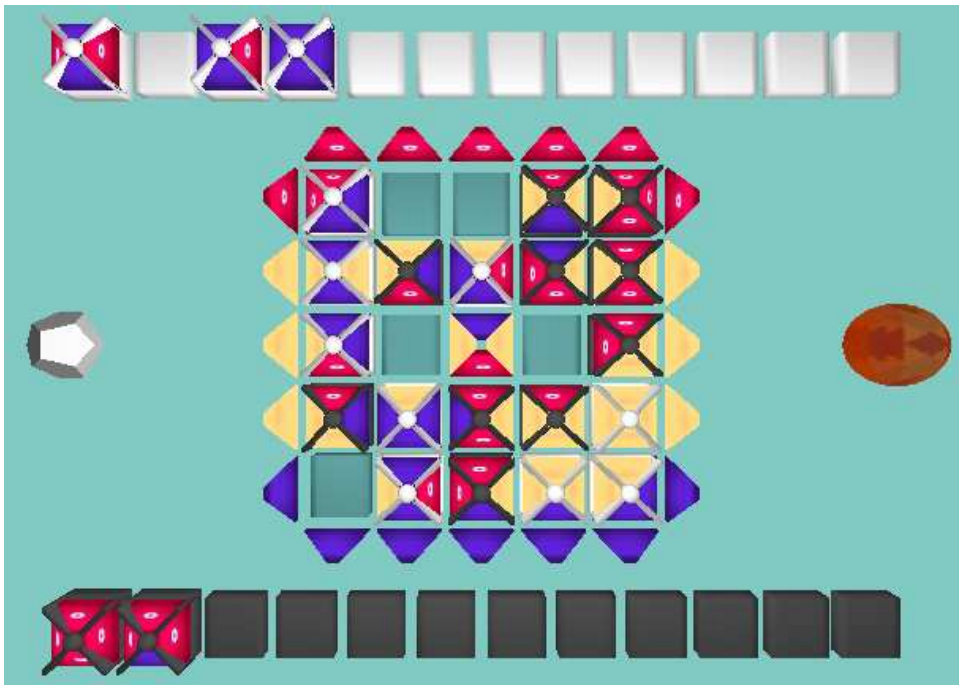


Ilustración 8-35 - Con Piezas Negras: Negras Ganan

Análisis de las pruebas

El resultado obtenido con el método de comprobación propuesto ha sido efectivo y ha permitido comprobar todas las situaciones posibles. Cabe acotar que este tipo de pruebas, conllevan la tarea extra de las pruebas del programa de pruebas. En este caso se optó por la revisión detallada del código, dado que su complejidad era relativamente baja. Igualmente, la planificación paso a paso de la migración del código fuente del proyecto al código fuente de pruebas, fue asegurando la calidad de su resultado.

8.3.2.6 Pruebas de Obtener Trazado

Acción	Efecto	Resultado
Invocar las teclas de obtención de trazados <ctrl> + T	Se abre un explorador de windows en la carpeta donde se encuentran los registros de partidas	Aprobado (ver Ilustración 8-36)
Abrir una partida con el programa de planilla de cálculos elegido	Verificar la información.	Aprobado (ver Ilustración 8-37)

Tabla 8-6 - Pruebas de Obtener Trazado

Capturas de Pantalla

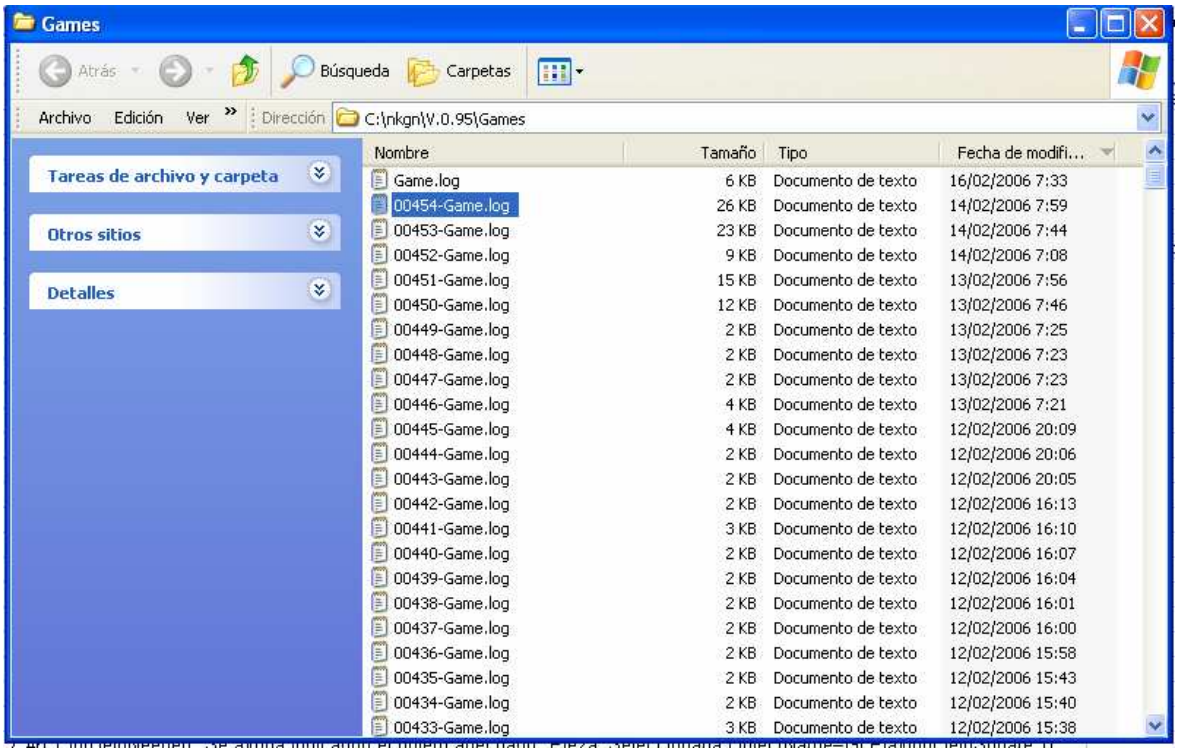
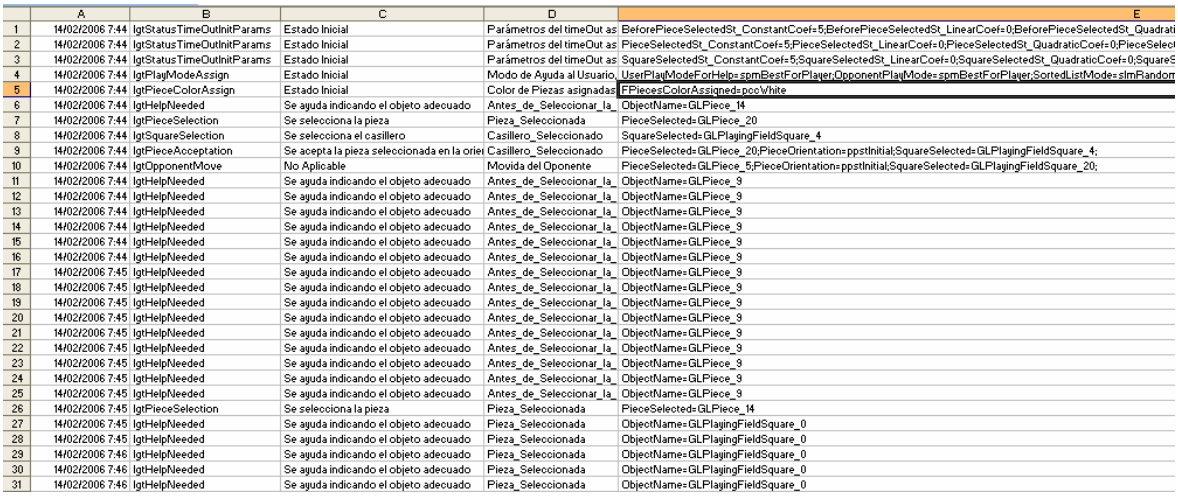


Ilustración 8-36 - Apertura del Explorador al presionar las teclas <Ctrl> + T



	A	B	C	D	E
1	14/02/2006 7:44	IgtStatusTimeOutInitParams	Estado Inicial	Parámetros del timeOut as	BeforePieceSelectedSt_ConstantCoef=5;BeforePieceSelectedSt_LinearCoef=0;BeforePieceSelectedSt_Quadrati
2	14/02/2006 7:44	IgtStatusTimeOutInitParams	Estado Inicial	Parámetros del timeOut as	PieceSelectedSt_ConstantCoef=5;PieceSelectedSt_LinearCoef=0;PieceSelectedSt_QuadraticCoef=0;PieceSelect
3	14/02/2006 7:44	IgtStatusTimeOutInitParams	Estado Inicial	Parámetros del timeOut as	SquareSelectedSt_ConstantCoef=5;SquareSelectedSt_LinearCoef=0;SquareSelectedSt_QuadraticCoef=0;SquareS
4	14/02/2006 7:44	IgtPlayModeAssign	Estado Inicial	Modo de Ayuda al Usuario	UserPlayModeForHelp=spmBestForPlayer;OpponentPlayMode=spmBestForPlayer;SortedListMode=slmRandom
5	14/02/2006 7:44	IgtPieceColorAssign	Estado Inicial	Color de Piezas asignadas	FPiecesColorAssigned=pccWhite
6	14/02/2006 7:44	IgtHelpNeeded	Se ayuda indicando el objeto adecuado	Antes_de_Seleccionar_la_	ObjectName=GLPiece_14
7	14/02/2006 7:44	IgtPieceSelection	Se selecciona la pieza	Pieza_Seleccionada	PieceSelected=GLPiece_20
8	14/02/2006 7:44	IgtSquareSelection	Se selecciona el casillero	Casillero_Seleccionado	SquareSelected=GLPlayingFieldSquare_4
9	14/02/2006 7:44	IgtPieceAcceptation	Se acepta la pieza seleccionada en la orie	Casillero_Seleccionado	PieceSelected=GLPiece_20;PieceOrientation=ppstInitial;SquareSelected=GLPlayingFieldSquare_4;
10	14/02/2006 7:44	IgtOpponentMove	No Aplicable	Movida del Oponente	PieceSelected=GLPiece_5;PieceOrientation=ppstInitial;SquareSelected=GLPlayingFieldSquare_20;
11	14/02/2006 7:44	IgtHelpNeeded	Se ayuda indicando el objeto adecuado	Antes_de_Seleccionar_la_	ObjectName=GLPiece_9
12	14/02/2006 7:44	IgtHelpNeeded	Se ayuda indicando el objeto adecuado	Antes_de_Seleccionar_la_	ObjectName=GLPiece_9
13	14/02/2006 7:44	IgtHelpNeeded	Se ayuda indicando el objeto adecuado	Antes_de_Seleccionar_la_	ObjectName=GLPiece_9
14	14/02/2006 7:44	IgtHelpNeeded	Se ayuda indicando el objeto adecuado	Antes_de_Seleccionar_la_	ObjectName=GLPiece_9
15	14/02/2006 7:44	IgtHelpNeeded	Se ayuda indicando el objeto adecuado	Antes_de_Seleccionar_la_	ObjectName=GLPiece_9
16	14/02/2006 7:44	IgtHelpNeeded	Se ayuda indicando el objeto adecuado	Antes_de_Seleccionar_la_	ObjectName=GLPiece_9
17	14/02/2006 7:45	IgtHelpNeeded	Se ayuda indicando el objeto adecuado	Antes_de_Seleccionar_la_	ObjectName=GLPiece_9
18	14/02/2006 7:45	IgtHelpNeeded	Se ayuda indicando el objeto adecuado	Antes_de_Seleccionar_la_	ObjectName=GLPiece_9
19	14/02/2006 7:45	IgtHelpNeeded	Se ayuda indicando el objeto adecuado	Antes_de_Seleccionar_la_	ObjectName=GLPiece_9
20	14/02/2006 7:45	IgtHelpNeeded	Se ayuda indicando el objeto adecuado	Antes_de_Seleccionar_la_	ObjectName=GLPiece_9
21	14/02/2006 7:45	IgtHelpNeeded	Se ayuda indicando el objeto adecuado	Antes_de_Seleccionar_la_	ObjectName=GLPiece_9
22	14/02/2006 7:45	IgtHelpNeeded	Se ayuda indicando el objeto adecuado	Antes_de_Seleccionar_la_	ObjectName=GLPiece_9
23	14/02/2006 7:45	IgtHelpNeeded	Se ayuda indicando el objeto adecuado	Antes_de_Seleccionar_la_	ObjectName=GLPiece_9
24	14/02/2006 7:45	IgtHelpNeeded	Se ayuda indicando el objeto adecuado	Antes_de_Seleccionar_la_	ObjectName=GLPiece_9
25	14/02/2006 7:45	IgtHelpNeeded	Se ayuda indicando el objeto adecuado	Antes_de_Seleccionar_la_	ObjectName=GLPiece_9
26	14/02/2006 7:45	IgtPieceSelection	Se selecciona la pieza	Pieza_Seleccionada	PieceSelected=GLPiece_14
27	14/02/2006 7:45	IgtHelpNeeded	Se ayuda indicando el objeto adecuado	Pieza_Seleccionada	ObjectName=GLPlayingFieldSquare_0
28	14/02/2006 7:45	IgtHelpNeeded	Se ayuda indicando el objeto adecuado	Pieza_Seleccionada	ObjectName=GLPlayingFieldSquare_0
29	14/02/2006 7:46	IgtHelpNeeded	Se ayuda indicando el objeto adecuado	Pieza_Seleccionada	ObjectName=GLPlayingFieldSquare_0
30	14/02/2006 7:46	IgtHelpNeeded	Se ayuda indicando el objeto adecuado	Pieza_Seleccionada	ObjectName=GLPlayingFieldSquare_0
31	14/02/2006 7:46	IgtHelpNeeded	Se ayuda indicando el objeto adecuado	Pieza_Seleccionada	ObjectName=GLPlayingFieldSquare_0

Ilustración 8-37 - Edición de un archivo de trazado desde una planilla de cálculos

Análisis de las pruebas

Se ha podido comprobar que todas las partidas quedan registradas. Se abrió una de ellas a modo de planilla de cálculos y se analizó el registro, encontrándose todos los pasos realizados durante la partida.

8.3.3 Evaluación del Resultado de las Pruebas del Sistema

8.4 Manual de Usuario

Cómo se indicó en el ítem 7.7.1, el manual de uso del sistema tiene por finalidad ayudar al investigador a configurar el sistema y comprender el contenido de los archivos de trazado, dado que el usuario de juego, aprenderá a partir de las guías que el software le da a medida que va haciendo uso del mismo.Queda en manos del investigador complementar este manual con directivas propias para la administración y evaluación, constituyendo así un procedimiento de configuración, administración y evaluación.

8.4.1 Introducción

El objetivo del sistema es el de permitir el trazado de la interacción de un niño con el sistema en el cual los estímulos y las capacidades requeridas son fundamentalmente visuales y hacen uso de secuencias. El niño interacciona con los distintos objetos que presenta el software y a partir de sus inacciones y errores, el sistema lo va guiando al aprendizaje de las reglas del juego en forma práctica.

No está limitado a una sola partida, es decir que el niño podrá jugar de acuerdo a la definido por el investigador. La instalación del Software es sencilla y por ello, el investigador podrá entregar una instalación para que el niño pueda jugar en su casa y sencillamente recibir los archivos de trazado para su posterior análisis.

En este manual se indicará por un lado cómo configurar el sistema de forma tal de adecuarlo a las condiciones de evaluación buscadas por un lado y una vez obtenidos los datos, tener la posibilidad de analizarlos.

8.4.2 Configuración del Sistema

El sistema se configura modificando un archivo que se encuentra en el directorio de instalación del sistema, por ej. en:

C:\Archivos de programa\HLME

El nombre del archivo es *HLME.conf* y el tipo de archivo es de configuración de windows. Este tipo de archivos son de sencilla interpretación y se componen de secciones, entradas y valores para las entradas. En este caso, existe una única sección denominada General que puede observarse en la primer línea entre corchetes (las secciones se indican de esa forma).

8.4.2.1 Configuraciones Generales

Vale la pena aclarar los 2 modos posibles que tiene el software de "pensar jugadas" tanto para las movidas del oponente como para las ayudas al usuario, al momento son:

- **Primera Encontrada**: Implica el menor esfuerzo de cálculo, con lo cual, la búsqueda de la pieza se detiene al encontrar aquella que pueda ser jugada en el casillero evaluado.
- **Mejor Jugada**: Usa una heurística sencilla para decidir la pieza en el casillero evaluado.

Los valores a configurar son los siguientes:

- **ColorAssingModeOrd**: Permite elegir el modo de asignación del color de piezas al usuario. Cada caso representa una posible asignación configurable por el investigador de acuerdo a su criterio, a saber:
    - **0**: El usuario siempre jugará con Piezas Blancas.
    - **1**: El usuario siempre jugará con Piezas Negras.
    - **2**: El usuario jugara en forma alternada una partida con Piezas blancas, la siguiente negras y así siguiendo.
    - **3**: El sistema decide en forma aleatoria el color de las Piezas al iniciar la partida.
- **UserPlayModeForHelp**: Modo de "pensar jugadas" al momento de ayudar al usuario:
    - **0**: Modo **Primera Encontrada**
    - **1**: Modo **Mejor Jugada**

- **OpponentPlayMode**: Modo de "pensar jugadas" al momento de realizar la jugada del oponente:
    - **0**: Modo **Primera Encontrada**
    - **1**: Modo **Mejor Jugada**
- **SortedListMode**: Posibles Modos de Ordenamiento de la listas de piezas sin usar y de casilleros sin usar que ingresan al proceso de "pensar jugada" e influyen en el mismo.
    - **0**: Modo **Secuencial**
    - **1**: Modo **Aleatorio**
- **UserPieceColorOrd**: Indica cual fue el último color de pieza utilizado. Solamente interesa para el caso en que **ColorAssingModeOrd** sea = 2.

8.4.2.2 Configuraciones de Tiempo de Espera para Ayudar al Usuario

El resto de las configuraciones tiene que ver con los tiempos de espera del sistema para ayudar al usuario.

Existen 3 estados del sistema en los cuales el usuario puede recibir ayuda, ellos son:

- **Antes de Seleccionar la Pieza**: Ocurre en cada jugada antes que el usuario haga clic sobre una pieza no utilizada de su sector.
- **Pieza Seleccionada**: Ocurre en cada jugada una vez que el usuario selecciona una pieza y la misma se eleva destacándose del resto.
- **Casillero Seleccionado**: Ocurre en cada jugada una vez que el usuario selecciona un casillero de juego.

Los tiempos de espera pueden ser configurados en forma individual para cada uno de ellos. Esto es así para simular al tutor, que cuando ve que le cuesta más una tarea al usuario, participa más seguido y va espaciando la ayuda a medida que se notan avances.

La fórmula a utilizar en cada estado es una ecuación cuadrática de la forma:

$$\textbf{Tiempo de Espera} = A + B \cdot x + C \cdot x^2$$

Donde **x** (Término independiente) será calculado de la siguiente manera para cada uno de los estados:

- Si durante la movida el usuario recibe ayuda, **x** reduce su valor en 1 (salvo que ya valga 0, valor mínimo que podrá tomar).
- Si durante la movida el usuario no recibe ayuda, **x** aumenta su valor en 1.

A continuación se muestran los valores configurable en *HLME.conf*:

- **Antes de Seleccionar la Pieza**:
    - **A: Antes_de_Seleccionar_la_Pieza_ConstantCoef**
    - **B: Antes_de_Seleccionar_la_Pieza_LinearCoef**
    - **C: Antes_de_Seleccionar_la_Pieza_QuadraticCoef**

    - **x: Antes_de_Seleccionar_la_Pieza_IndependentTerm**
- **Pieza Seleccionada**:
    - **A: Pieza_Seleccionada_ConstantCoef**
    - **B: Pieza_Seleccionada_LinearCoef**
    - **C: Pieza_Seleccionada_QuadraticCoef**
    - **x: Pieza_Seleccionada_IndependentTerm**
- **Casillero Seleccionado**:
    - **A: Casillero_Seleccionado_ConstantCoef**
    - **B: Casillero_Seleccionado_LinearCoef**
    - **C: Casillero_Seleccionado_QuadraticCoef**
    - **x: Casillero_Seleccionado_IndependentTerm**

El valor de **x**, se va actualizando en cada partida.

Por lo expuesto, si el investigador desea configurar la ayuda para que siempre aparezca a una determinada cantidad de segundos, para un estado dado, elegirá tanto el Coeficiente Lineal como el cuadrático en 0 (cero). El valor elegido en ConstantCoef, será entonces el valor mínimo que podrá tomar el tiempo de espera para la ayuda. Un valor de 1 o más al LinearCoef, hará que a medida que mejora el desempeño, junto a la mejora se expandirá el tiempo de espera para la ayuda. Un valor de 1 o más en QuadraticCoef, surtirá igual efecto que LinearCoef, pero con variaciones más importantes.

Es importante en las distribuciones, que se realicen, que el término independiente se encuentre en cero.

8.4.3 Archivo de Trazado de la Partida

Los archivos de trazado de una partida se encuentran en la carpeta *Games* justo debajo de la carpeta donde está instalado el sistema.

La denominación del archivo está dada por un número de 5 (cinco) cifras seguido de *–games*. Por ejemplo:

00015-game.csv

El formato del archivo es "valores separados por comas" (comma separated value file format). Esto significa que cada columna viene separada de la siguiente por una coma.

El archivo tiene cinco columnas, a saber:

- **Fecha y hora del suceso**: Con precisión al segundo.
- **Tipo de registro**: Uno de los siguientes:
    - **lgtError**: El Objeto señalado es inadecuado siempre

- **lgtErrorForThisStatus**: El Objeto señalado es inadecuado para el estado actual (pero podría ser válido en otro estado)
- **lgtGameEnded**: Finalizó la partida
- **lgtHelpNeeded**: El usuario necesita ayuda
- **lgtLeftRotation**: Se indica rotar a la izquierda
- **lgtMovementInAdvance**: Se adelanta en el movimiento, es decir, no espera que finalice el desplazamiento de piezas propias o del oponente
- **lgtOpponentMove**: Movida del oponente
- **lgtPieceAcceptation**: Se acepta la pieza seleccionada en el casillero seleccionado
- **lgtPieceCancelation**: Se cancela la pieza seleccionada
- **lgtPieceColorAssign**: Color de piezas asignadas al jugador
- **lgtPieceSelection**: Se selecciona una pieza
- **lgtPlayModeAssign**: Modo de juego del usuario y del oponente
- **lgtRightRotation**: Se indica rotar a la derecha
- **lgtSquareAndPieceCancelation**: Se cancela la selección del casillero y de la pieza
- **lgtSquareChanging**: Se cambia el casillero seleccionado por otro
- **lgtSquareSelection**: Se selecciona un casillero
- **lgtStatusTimeOutInitParams**: Parámetros iniciales del TimeOut correspondiente al estado referido

- **Descripción**: Complementa la referencia dada en la columna anterior adecuándola a cada situación.

- **Parámetros**: Información complementaria relevante para el trazado que se adapta a cada situación, que permitirá en el futuro de ser necesario, realizar una simulación completa de la partida.

8.5 Instalación del Software

Se ha realizado la instalación auto asistida del software (ver Ilustraciones 8-38 a 8-43) de forma tal de cubrir dos visiones distintas bajo una misma instalación. Por un lado, la instalación básica para el investigador, que le permitirá instalar el software para realizar sus propias pruebas y por el otro, la correspondiente a la PC del niño del cual se desea analizar su desempeño ante los estímulos brindados por el software.

En el primer caso, el archivo a utilizar es el denominado *InstalarHLME.EXE*, el cual realiza la instalación incluyendo el archivo de configuración *hlme.conf* con los siguientes parámetros iniciales:

```
[GENERAL]

Antes_de_Seleccionar_la_Pieza_ConstantCoef=5
Antes_de_Seleccionar_la_Pieza_LinearCoef=2
Antes_de_Seleccionar_la_Pieza_QuadraticCoef=1
Antes_de_Seleccionar_la_Pieza_IndependentTerm=0
Pieza_Seleccionada_ConstantCoef=5
Pieza_Seleccionada_LinearCoef=2
Pieza_Seleccionada_QuadraticCoef=1
Pieza_Seleccionada_IndependentTerm=0
Casillero_Seleccionado_ConstantCoef=5
Casillero_Seleccionado_LinearCoef=2
Casillero_Seleccionado_QuadraticCoef=1
Casillero_Seleccionado_IndependentTerm=0

ColorAssingModeOrd=0
UserPieceColorOrd=0
UserPlayModeForHelp=1
OpponentPlayMode=1
SortedListMode=1
```

En el segundo caso, en la versión que distribuye el investigador, difiere en que puede ser acompañada por un archivo *hlme.conf* con los parámetros que él mismo ha configurado. La instalación reconoce al archivo de configuración si está presente y lo copia al directorio de instalación.

Ilustración 8-38 - Instalación Paso 1: Bienvenida

Ilustración 8-39 - Instalación Paso 2: Selección de la carpeta de instalación del software

Ilustración 8-40 - Instalación Paso 3: Aviso de instalación lista a iniciarse

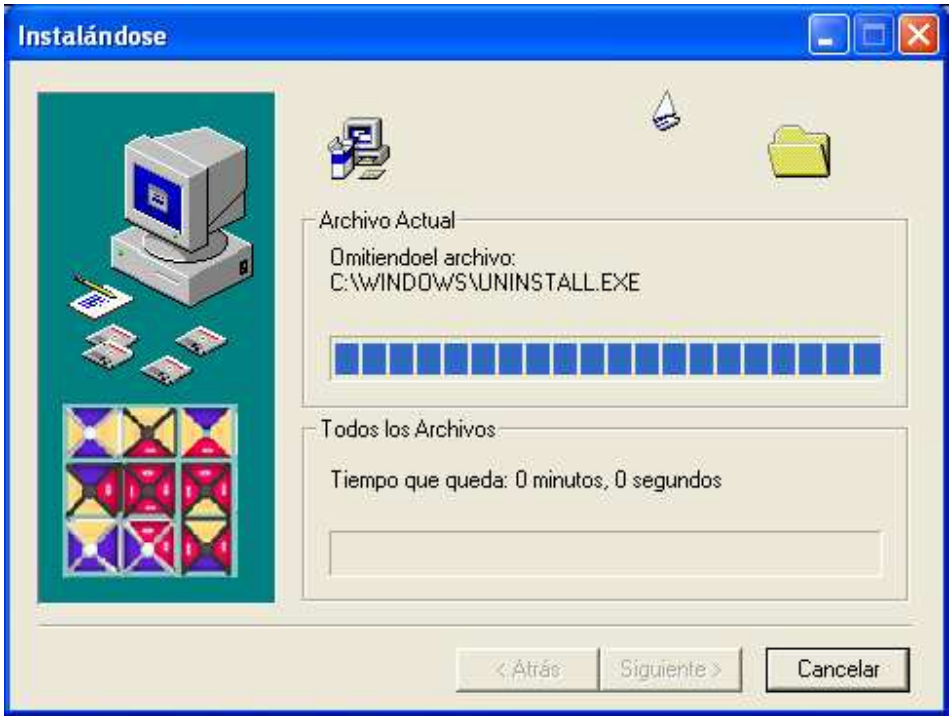


Ilustración 8-41 - Proceso de Instalación de archivos y Configuración del Sistema

Ilustración 8-42 - Instalación Paso 5: Instalación finalizada

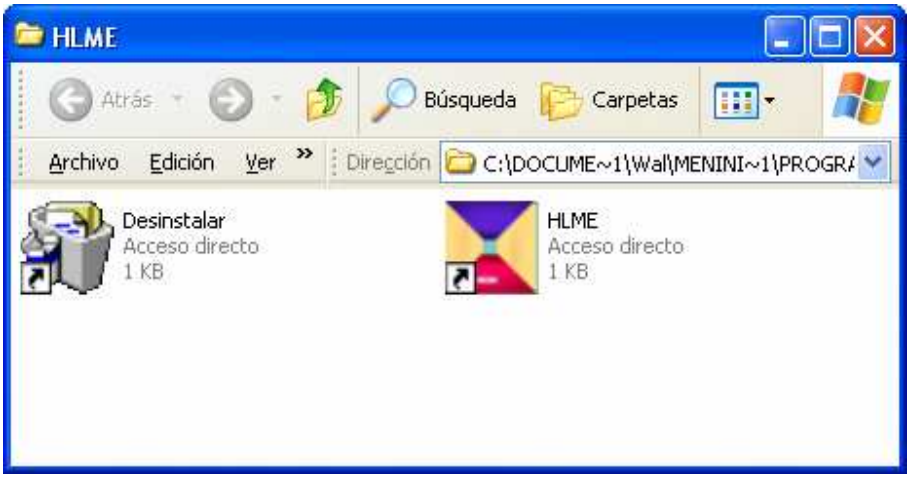


Ilustración 8-43 - Grupo de Programas incluyendo la desinstalación del mismo

8.6 Pruebas de Aceptación del Sistema

A continuación se transcribe el resultado de las pruebas de Aceptación del Sistema definidas en el Ítem 6.8.3.

Prueba	Resultado
El Investigador juega una partida haciendo las veces del niño.	Aprobado. Realiza 6 partidas, 3 de ellas con piezas blancas y 3 con piezas negras.
El Investigador obtiene el trazado de su partida.	Aprobado. Un par de archivos son revisados y el investigado empieza a familiarizarse no la nomenclatura utilizada a partir del uso del manual.
El Investigador designa a un niño sin NEE para que realice la experiencia de las partidas que sean necesarias hasta que haya demostrado el aprendizaje de las reglas.	Aprobado. El niño designado (de 9 años de edad), juega con entusiasmo y en siete partidas demuestra haber comprendido las reglas.
El Investigador obtiene el trazado las partidas experimentadas por el niño.	Aprobado. Ahora va evaluando qué otra información podrá extraer a partir de los datos observados.
Cada trazado de partida podrá analizarse desde un programa para planillas de cálculo, sin necesidad de hacer conversiones ni adaptaciones al archivo.	Aprobado. Se verificó en todas las oportunidades en que se realizó el proceso de abrir los archivos de trazado.

Tabla 8-7 - Pruebas de Aceptación del Sistema

Dados los resultados positivos, se da por Aceptado el Sistema de información. Las mejoras indicadas por el investigador fueron incorporadas como nuevas líneas de investigación en el Ítem 9.3.

9 Métricas, Conclusiones y Nuevas Líneas de Investigación

9.1 Métricas del sistema

En general, obtener las métricas de un sistema sirve para mejorar la calidad del equipo de desarrollo. Los objetivos de mejora continua en la calidad del software, pueden ser llevados a cabo a partir de la obtención de métricas a sucesivos sistemas desarrollados y su posterior comparación. Si para un grupo de desarrollo, se obtiene por ejemplo una relación Líneas de Código/Líneas de Comentarios igual a 10, podremos poner como objetivo de mejora en la calidad llegar a un valor de 5 en una primera etapa y luego a 3 en una segunda.

A continuación se detallan (Tabla 9-1) algunas métricas resultantes del análisis del código generado con el programa CodeLens v.0.9 (http://sourceforge.net/projects/codelens), herramienta de fuentes abiertas y distribución gratuita.

Unidades	
Unidades de Biblioteca	49
Unidades del Proyecto	6
Total de Unidades	**55**
Líneas de Código del Proyecto	
Líneas de Código del Proyecto	3294
Líneas de Código en blanco	397
Líneas de Comentarios	1274
Total de Líneas del Proyecto	**4965**
Clases	
Clases del Proyecto	43
Total de Clases del Proyecto	**43**

Tabla 9-1 - Métricas del Proyecto

Con esta información, podemos tener que la relación Líneas de Código/Líneas de Comentarios es de **2,59**. Si aceptamos como ideal el valor 1 para dicha relación, vemos que el valor obtenido en el proyecto es más que aceptable.

9.2 Conclusiones

Las conclusiones se presentarán de forma tal de cubrir los aspectos correspondientes al proyecto en sí y al software resultante.

9.2.1 Conclusiones sobre el proyecto

- **Duración del Proyecto Total**: El proyecto se completó en la fecha prevista (01/03/2006). De la tabla comparativa de planificación de Procesos y su duración real (Tabla Anexo I – 1), se observa que hubo una diferencia de 14 días (algo más de un 5% del tiempo previsto), que fue absorbido aumentando la cantidad de horas semanales de dedicación en las últimas etapas. El uso de Staffing Size como método de estimación de tiempos y esfuerzos con las modificaciones incluidas por el autor, resultaron adecuados al proyecto (Ítem 4.2.1).

- **Duración del Proceso de Análisis**: Puede observarse que su duración se excedió a lo planificado. Esto puede explicarse en que en el proceso se aprendió a hacer uso de diagramas UML y en particular, fue dificultoso encontrar ejemplos adecuados de diagramas de casos de uso de juegos del estilo del proyecto.

- **Duración del Proceso de Diseño**: Aquí, también se observa que su duración se excedió a lo planificado, sin embargo, la justificación en este caso se debe al cambio en de herramienta de diseño utilizada como fue explicado en el Ítem 7.1.2.4. Sin embargo, puede observarse que la mitad de la demora fue compensada en el proceso de desarrollo. Esto es lógico, debido a que el proyecto se diseñó directamente en el entorno de desarrollo.

- **Calidad del Proyecto**: En este aspecto, el tesista, acostumbrado al uso de metodologías de desarrollo ágiles como es la "programación extrema" (http://www.programacionextrema.org/), puso especial atención en la construcción del código fuente. Si bien en programación extrema intenta mejorarse la calidad de construcción a partir de la escritura de a pares del código fuente (cuatro ojos ven más que dos), las técnicas de lectura y re-lectura de lo escrito, facilitan la detección de errores.

- **Pruebas del Sistema**: Complementando lo expresado previamente, las pruebas fueron realizadas principalmente sobre el producto terminado, y su refutado fue satisfactorio. Permitieron realizar ajustes en la interfaz y en la determinación de tiempos de espera para mostrar la ayuda.

- **Uso de Librerías Gráficas**: El uso de GLScene como librería gráfica de interfaz con las limitadas funciones que publica OpenGL (aunque eficientes), permitieron por un lado obtener un producto de buena calidad gráfica, pero por sobre todo, facilitarán su evolución futura mejorando la animación y presentación general en el futuro (ver 9.3.1).

- **Metodología**: El uso de la metodología Métrica v.3 ha sido de utilidad. Su flexibilidad y adaptabilidad permitieron que el proyecto haya tenido una estructura adecuada. Sus guías fueron fundamentales a la hora de definir y formalizar los sucesivos procesos, actividades y tareas. Se pudo formalizar la presentación de los distintos documentos en el presente haciendo las adaptaciones que en cada caso se consideraron apropiadas.

9.2.2 Conclusiones sobre el software

- Una de las cuestiones planteadas al inicio del proyecto, fueron saber si el software era o no un Sistema Tutorial Inteligente (STI). Una vez finalizado el mismo, se podrá disentir en cuánto se acerca o no a un tutorial inteligente, ya que funcionalmente pareciera serlo. Lo que si se puede corroborar es lo dificultoso que resulta el diseño e implementación de una interfaz gráfica que cumpla con los requisitos que en general tiene un Sistema de estas características.

- La comunicación visual alcanzada ha sido más que satisfactoria. Las ayudas resultan claras como así los desplazamientos.

- Para resumir la forma de aprendizaje de las reglas del juego por parte del niño, podemos decir que lo hace en base a pruebas y error con ayudas, es decir, encuadra en la línea del Descubrimiento Guiado, uno de las categorías de STIs presentadas en el Ítem 2.1.3.1.

- Finalmente, el juego tiene ciertas similitudes respecto de otros juegos de mesa como son el ajedrez y las damas. Existen piezas blancas y piezas negras. Las blancas comienza a jugar. Se juega alternadamente una pieza por jugador. Esto implica que la variable conocimiento de juegos similares sea de interés para el investigador.

9.3 Nuevas Líneas de Investigación

A continuación se establecen las Nuevas líneas de Investigación y mejoras del software aportadas por el investigador como fue señalado en el Ítem 8.6 y por el tesista.

9.3.1 Interfaz Multimedia

Esta es un área donde mucho se puede mejorar. Desde hacer que las piezas encastren (como piezas de puzzles), es decir, que no sólo relacionen colores sino formas adaptativas, hasta la inclusión de personajes animados que ayuden, guíen, jueguen por el oponente, festejen las jugadas correctas y todo aquello que pueda aportar a mejorar la interacción.

Desde el punto de vista de los sonidos, se podría agregar música de fondo contextualizada, por ejemplo que aumente el volumen a medida que se acerca el momento de ayudar al usuario. Asimismo, se podrán mejorar los sonidos que indican el error o la ayuda.

9.3.2 "Inteligencia" del Modelo

La implementación de la operación "Pensar Jugada" se implementó con una simple heurística consistente en contar la cantidad de posibles jugadas que tiene cada pieza y quedarse con la que menos tiene. El algoritmo funciona adecuadamente, pero adolece del problema que sobre todo en las primeras movidas muchas piezas se equiparan en el número mencionado, con lo que la resolución adoptada se toma en forma aleatoria.

Se estudian heurísticas mucho más complejas, pero que permitirán anticipar jugadas y evaluar tanto en qué estado quedan las piezas propias como las del oponente. Con ello, se podrá seleccionar el perfil de personalidad del Jugador en agresivo y conservador o aún en que grado de uno y otro perfil se encuadra. Cabe destacar que las combinatorias posibles, exceden las capacidades de cálculo de las PCs para el problema dado, de ahí el hecho de utilizar técnicas alternativas al recorrer todas las soluciones posibles del problema.

9.3.3 Análisis de los datos obtenidos

En este sentido, se ha hecho lo básico, es decir, el trazado de cada una de las partidas. Pero el análisis que se puede hacer individualmente podrá profundizarse si dicha información se registra en una base de datos. Dicha base de datos será administrada por otra aplicación que administrada por el investigador, pueda realizar desde estadísticas complejas hasta el uso de técnicas de Data Mining.

Para ello, deberán realizarse un par de tareas previamente. Por un lado, el agregado de la información correspondiente al niño al que se evalúa (edad, escolaridad, conocimientos de otros juegos, experiencia en la computadora, etc.). Por el otro, se tratará de sintetizar la información correspondiente a partir del proceso de cada partida (duración, si fue completada o no, cantidad de errores, cantidad de veces que se ayudó en cada estado, etc.) y posteriormente al grupo de partidas evaluadas (sumas, promedios, desvíos de las mencionadas).

9.3.4 Interacción Investigador – Usuario

No existe al momento una función que permita al padre del niño que instala en su casa el software que exporte los registros cronológicos en forma automática. Lo deseable, sería que el sistema pueda realizar dicha tarea por email, sin más que pulsar una combinación de teclas adecuadas.

Otra posibilidad interesante, es que el investigador pueda hacer una evaluación a distancia de los avances del niño, jugando el mismo una partida. Para ello, habría que facilitar la conexión por red entre las aplicaciones de ambos actores.

9.3.5 ¿Por dónde seguir entonces?

En estos momentos el paso inicial está dado: el sistema funciona, el registro es completo y queda a disposición de los investigadores para usarlo en experiencias preliminares. Del resultado de las mismas, se verá en que orden se tomarán las distintas líneas de investigación mencionadas con la tranquilidad que brinda una arquitectura adecuada que permitirá adaptar el software a cualquiera de los cambios previstos.

Anexo I: Comparación entre Planificación Estimada y Tiempos Aplicados

A continuación se presenta la Tabla con los distintos procesos realizados, donde las duraciones se miden en días y la diferencia resulta de realizar la resta entre lo real y lo planificado (un valor negativo implica que el proceso llevó menos tiempo que el planificado).

El análisis de los datos se realiza en el Ítem 9.2.1

Procesos, Actividades y Tareas	Planif. Dura-ción	Planif. Comienzo	Planif. Fin	Real. Dura -ción	Dife-rencia
EVS Estudio de Viabilidad del Sistema	13	01/03/05	17/03/05	10	-2
GPI Gestión del Proyecto	6	18/03/05	25/03/05	5	-1
GC Gestión de la Configuración del Sistema	5	28/03/05	01/04/05	5	0
CAL Gestión de la Calidad del Sistema	5	04/04/05	08/04/05	5	0
ASI Análisis del sistema de información	76	11/04/05	25/07/05	88	12
DSI Diseño del sistema de información	62	26/07/05	19/10/05	76	14
CSI Construcción del Sistema de Información	67	20/10/05	20/01/06	61	-7
TES Tesis	18	06/02/06	01/03/06	16	-3
Total	**252**			**265**	**-14**

Tabla Anexo I - 1 Comparación entre Tiempos Estimados y Reales

Anexo II: Gestión de Configuración

A continuación se presenta la Tabla Anexo II- 1 con los distintos componentes del proyecto gestionados y sus sucesivas versiones con la descripción correspondiente.

Gestión de Configuración de Documentos

Se detallan todos los documentos generados y sus sucesivas versiones según lo especificado en el Ítem 5.3.1.

Nombre	Título	Descripción
wok-Cap1v1.0.a	Introducción	Versión Inicial
wok-Cap1v1.0.b	Introducción	Correcciones de Estilo y Formato por revisión
wok-Cap1v1.0.c	Introducción	Corrección de referencias al resto de los documentos (capítulos)
wok-Cap2v1.0.a	Estudio de Viabilidad del Sistema	Versión Inicial
wok-Cap2v1.0.b	Estudio de Viabilidad del Sistema	Correcciones de Estilo y Formato por revisión
wok-Cap3v1.0.a	Especificación de requerimientos del sistema	Versión Inicial
wok-Cap3v1.0.b	Especificación de requerimientos del sistema	Correcciones de Estilo y Formato por revisión. Redefinición de sinónimos, definiciones y abreviaturas.
wok-Cap4v1.0.a	Gestión del Proyecto	Versión Inicial
wok-Cap4v1.0.b	Gestión del Proyecto	Correcciones de Estilo y Formato por revisión
wok-Cap5v1.0.a	Calidad, Seguridad y Gestión de Configuración	Versión Inicial
wok-Cap6v1.0.a	Análisis del sistema	Versión Inicial
wok-Cap6v1.0.b	Análisis del sistema	Correcciones de Estilo y Formato por revisión. Corrección de análisis de final de partida

Nombre	Título	Descripción
wok-Cap6v1.0.c	Análisis del sistema	Nuevas correcciones de Estilo y Formato por revisión. Adaptación del diagrama de modelo del dominio
wok-Cap7v1.0.a	Diseño del sistema	Versión Inicial
wok-Cap7v1.0.b	Diseño del sistema	Correcciones de Estilo y Formato por revisión. Adaptación del formato de producto de diseño obtenido por la herramienta utilizada al del documento.
wok-Cap8v1.0.a	Construcción e Implantación del Sistema de Información	Versión Inicial
wok-Cap8v1.0.b	Construcción e Implantación del Sistema de Información	Correcciones de Estilo y Formato por revisión. Se agrega Instalación del Software
wok-Cap8v1.0.c	Construcción e Implantación del Sistema de Información	Nuevas correcciones de Estilo y Formato por revisión. Se agregan las pruebas de aceptación del sistema
wok-Cap9v1.0.a	Métricas, Conclusiones y Nuevas Líneas de Investigación	Versión Inicial
wok-Cap9v1.0.b	Métricas, Conclusiones y Nuevas Líneas de Investigación	Correcciones de Estilo y Formato por revisión. Se agrega conclusión sobre metodología
wok-Anx Iv1.0a	Comparación entre Planificación Estimada y Tiempos Aplicados	Versión Inicial
wok-Anx IIv1.0a	Gestión de Configuración	Versión Inicial
wok-Anx IIIv1.0a	Bibliografía y Software	Versión Inicial
wok-Tesis v1.0a	Documento de Tesis unificado	Versión Inicial. Inclusión de todos los capítulos en un solo documento
wok-Tesis v1.0b	Documento de Tesis unificado	Adaptaciones a Documento único
wok-Tesis v1.0c	Documento de Tesis unificado	Correcciones de Estilo y Formato por revisión.

Nombre	Título	Descripción
wok-Tesis v1.0d	Documento de Tesis unificado	Nuevas Correcciones de Estilo y Formato por revisión.
wok-Tesis v1.1a	Documento de Tesis unificado	Nuevas Correcciones de Estilo y Formato por revisión. Se agrega el presente Anexo. Constituye la versión definitiva

Tabla Anexo II - 1 - Gestión de Configuración de Documentos

Gestión de Configuración de Archivos

En la tabla Anexo II – 2 se detallan todos los archivos generados y sus sucesivas versiones según lo especificado en el Ítem 5.3.2. según el tipo (ver tabla Anexo II – 3)

Nombre	Tipo	Título	Descripción
Common v1.0a	Código	Definiciones Comunes	Versión inicial a partir del diseño
Common v2.0a	Código	Definiciones Comunes	Código implementado
Controller v1.0a	Código	Clases del Controlador	Versión inicial a partir del diseño
Controller v2.0a	Código	Clases del Controlador	Versión inicial a partir del diseño
Controller v2.0b	Código	Clases del Controlador	Código implementado
FMain v1.0a	Código	Formulario Principal	Versión inicial a partir del diseño
FMain v2.0a	Código	Formulario Principal	Código implementado
FMain v2.0b	Código	Formulario Principal	Agregado de texturas a los lados de piezas para diferenciar el color azul del rojo en impresiones monocromáticas. Agregado de presentación del programa.
LogToFile v1.0a	Código	Clase del Registro cronológico de la Información	Versión inicial a partir del diseño
LogToFile v2.0a	Código	Clase del Registro cronológico de la Información	Código implementado

Nombre	Tipo	Título	Descripción
Model v1.0a	Código	Clases del Modelo	Versión inicial a partir del diseño
Model v2.0a	Código	Clases del Modelo	Código implementado
Model v1.0a	Código	Clases del Modelo	Versión inicial a partir del diseño
Model v2.0a	Código	Clases del Modelo	Código implementado
View v1.0a	Código	Clases de la Vista	Versión inicial a partir del diseño
View v2.0a	Código	Clases de la Vista	Código implementado
View v2.0b	Código	Clases de la Vista	Corrección de las rutinas ShowHelp y ShowError para adecuarse a la situación de muchos clics consecutivos.
View v2.0c	Código	Clases de la Vista	Adaptaciones de velocidades de traslación y rotación a partir de las pruebas realizadas.
ArrowTurnLeft v1.0a	3D	Flecha para giro hacia la izquierda	Versión inicial
ArrowTurnRight v1.0a	3D	Flecha para giro hacia la derecha	Versión inicial
Piece v1.0a	3D	Pieza	Versión inicial
Side v1.0a	3D	Lado de pieza y de borde	Versión inicial
Square v1.0a	3D	Casillero	Versión inicial
Error v1.0a	Sonido	Error	Versión inicial
Final v1.0a	Sonido	Final de partida	Versión inicial
Ayuda v1.0a	Sonido	Error	Versión inicial
InstalarHLME v1.0a	Instalación	Instalaciòn del Sistema	Versión inicial

Tabla Anexo II - 2 - Gestión de Configuración de Archivos

Los tipos de códigos utilizados fueron los siguientes (tabla Anexo II – 3):

Tipo	Descripción
Código	Archivo en lenguaje Pascal (.pas) integrante del proyecto
3D	Archivo tridimensional en formato de 3D Studio (3ds)
Sonido	Archivo de sonido en formato "wave" (.wav)
Ejecutable	Archivo de instalación del sistema (.exe)

Tabla Anexo II - 3 – Tipos de Archivos Gestionado

Anexo III – Unidades y Clases de Diseño

AIII.1 Unit Common

Used Units:

Used in interface: System, SysInit, sysutils
Used in implementation: *None*

Used by:
In Interface of Units: Controller, **FMain**, **Model**, **View**

Description:

Esta Unidad contiene todas las definiciones comunes a todos los submodelos. En ella se manifiestan las funciones de armado (y desarmado) de nombres de objeto de forma tal que no importe la implementación de ellos que haga la vista, el controlador podrá indicarlo. De igual forma se hará cuando tenga que hacerlo con el modelo. Para ello, cada clase visual indicable por los usuarios, tendrá asignada una constante.

Se incluyen además las constantes de directorios y archivos de registro y de cantidad total de piezas y casillero.

Además se definen los siguientes enumerados:

- Color de la Pieza
- Color de Lado
- Lados
- Comandos (Movimientos)

Finalmente se definen los arreglos bidemensionales (colores de cada lado de casillero y piezas) y unidimensional (color de cada pieza)

List of Variables:

Global Variables: InitialSquaresSideColor, **PiecesColors**, **PiecesSideColors**
File-Local Variables: *None*

List of Constants:

Global Constants: CONFIG_FILE_NAME, **GL_BORDER_STR**, **GL_PIECE_NAME_STR**, **GL_ROTATION_ARROWS_NAME_STR**, **GL_SQUARE_NAME_STR**, **GL_START_POSITION_NAME_STR**, **GL_TURN_LEFT_ARROW_NAME_STR**, **GL_TURN_RIGHT_ARROW_NAME_STR**, **LOG_FILE_DIR**, **LOG_FILE_NAME**, **movtAcceptPiece**, **movtGoOverSquare**, **movtReturnToHome**, **movtRotateLeft**, **movtRotateRight**, **movtSelect**, **movtUnSelect**, **NUMBER_OF_PIECES**, **NUMBER_OF_SQUARES**, **pccBlack**, **pccWhite**, **sdBottom**, **sdcBlue**, **sdcNone**, **sdcRed**, **sdcYellow**, **sdLeft**, **sdRight**, **sdTop**
File-Local Constants: *None*

List of Simple Types:

Global Simple Types: TMovement, **TPieceColor**, **TSide**, **TSideColor**
File-Local Simple Types: *None*

List of Functions:

Global Functions:
[ui] function extractNameNum
Extrae el Id del objeto a partir del nombre
[ui] function extractNameStr
Extrae la denominación de la clase a partir del nombre
[ui] function getNameStr
Nombre del Objeto a partir de la denominacion de la clase y del identificador asignado.
File-Local Functions:
[ul] procedure initialization
[ul] procedure setInitialSquaresSideColors
Establece el Color de cada lado de cada casillero.
[ul] procedure setPiecesColors
Asigna el color de pieza (blanco o negro) a cada una de ellas
[ul] procedure setPiecesSideColors
Asigna el color de cada lado a cada pieza

Variables:

[ui] InitialSquaresSideColor: **array** [0 .. **NUMBER_OF_SQUARES** - 1, **TSide**] **of TSideColor**
Arreglo bidimensional de asignación de colores a los lados de cada casilleror
- Id del Casillero
- Lado
Used in
TBoard.**InitializeSquares**, **setInitialSquaresSideColors**
[ui] PiecesColors: **array** [0 .. **NUMBER_OF_PIECES** - 1] **of TPieceColor**
Arreglo unidimensional de asignación del color de cada pieza
- Id de la Pieza
Used in
TBoard.**InitializePlayers**, **TController**.**SetNewPickedObject**, **setPiecesColors**
[ui] PiecesSideColors: **array** [0 .. **NUMBER_OF_PIECES** - 1, **TSide**] **of TSideColor**
Arreglo bidimensional de asignación de colores a los lados de cada pieza
- Id de la Pieza
- Lado
Used in
TBoard.**InitializePieces**, **TGLBoard**.**InitializePieces**, **setPiecesSideColors**

Constants:

[ui] CONFIG_FILE_NAME = 'HLME.conf'
Nombre del archivo de configuración
Used in
TController.**Create**
[ui] GL_BORDER_STR = 'GLBorder'
Used in
TGLBorders.**Create**, **getGLClassName**
[ui] GL_PIECE_NAME_STR = 'GLPiece'
Constantes de identificación de nombres de clases GL
Used in

TController.**DoNextMovement**, **TController**.**DoOpponentMove**,
TController.**EventOccursBeforePieceSelectedSt**,
TController.**EventOccursPieceSelectedSt**,
TController.**EventOccursSquareSelectedSt**, **TController**.**SetNewPickedObject**,
TController.**TimeOutBeforePieceSelectedSt**,
TController.**TimeOutPieceSelectedSt**, **TController**.**TimeOutSquareSelectedSt**,
TGLPieces.**Create**, **getGLClassName**

[ui] GL_ROTATION_ARROWS_NAME_STR = 'GLRotationArrows'
Used in
TGLRotationArrows.**SetNames**

[ui] GL_SQUARE_NAME_STR = 'GLPlayingFieldSquare'
Used in
TController.**DoNextMovement**, **TController**.**DoOpponentMove**,
TController.**EventOccursPieceSelectedSt**,
TController.**EventOccursSquareSelectedSt**, **TController**.**SetNewPickedObject**,
TController.**TimeOutPieceSelectedSt**, **TController**.**TimeOutSquareSelectedSt**,
TGLPlayFieldSquares.**Create**, **TMainForm**.**getGLObjectByName**, **getGLClassName**

[ui] GL_START_POSITION_NAME_STR = 'GLStartPositionSquare'
Used in
TController.**SetNewPickedObject**, **TGLPieces**.**Create**, **getGLClassName**

[ui] GL_TURN_LEFT_ARROW_NAME_STR = 'GLTurnLeftArrow'
Used in
TController.**EventOccursSquareSelectedSt**, **TController**.**SetNewPickedObject**,
TController.**TimeOutSquareSelectedSt**, **TGLRotationArrows**.**SetNames**,
getGLClassName

[ui] GL_TURN_RIGHT_ARROW_NAME_STR = 'GLTurnRightArrow'
Used in
TController.**EventOccursSquareSelectedSt**, **TController**.**SetNewPickedObject**,
TController.**TimeOutSquareSelectedSt**, **TGLRotationArrows**.**SetNames**,
getGLClassName

[ui] LOG_FILE_DIR = 'Games'
Directorio de Registros
Used in
TController.**Create**, **TController**.**GetTrace**

[ui] LOG_FILE_NAME = 'Game.log'
Nombre básico del archivo de registro
Used in
TController.**Create**

[ui] movtAcceptPiece = **TMovement**(6)
Aceptar Pieza: La pieza pasa desde la posición Destacada a la de reposo desplazándose sólo en el eje z
Used in
TController.**AcceptPiece**, **TController**.**DoNextMovement**,
TController.**SetCompleteMovement**, **TGLPiece**.**doMovement**

[ui] movtGoOverSquare = **TMovement**(3)
Ir hasta casillero: La pieza debe desplazar respecto del eje x y el eje y hasta posarse sobre el casillero indicado
Used in
TController.**DoNextMovement**, **TController**.**MovePieceOverSquare**,
TController.**SetCompleteMovement**, **TGLPiece**.**doMovement**

[ui] movtReturnToHome = **TMovement**(2)

Regresar a posición de Inicio: La pieza debe elevarse, desplazarse respecto el eje x y el eje y hasta quedar encima de la posición de inicio y bajar hasta la posición de reposo
Used in
TController.**CancelPieceSelected**, **TController**.**DoNextMovement**, **TGLPiece**.**doMovement**

[ui] movtRotateLeft = **TMovement**(4)
Rotar a la izquierda: Rotar la pieza 90º en sentido antihorario
Used in
TController.**DoNextMovement**, **TController**.**RotateLeft**, **TController**.**SetCompleteMovement**, **TGLPiece**.**doMovement**

[ui] movtRotateRight = **TMovement**(5)
Rotar a la derecha: Rotar la pieza 90º en sentido horario
Used in
TController.**DoNextMovement**, **TController**.**RotateRight**, **TController**.**SetCompleteMovement**, **TGLPiece**.**doMovement**

[ui] movtSelect = **TMovement**(0)
Seleccionar: La pieza debe desplazarse de la posición de reposo a la posición destacada
Used in
TController.**DoNextMovement**, **TController**.**SelectPiece**, **TController**.**SetCompleteMovement**, **TGLPiece**.**doMovement**

[ui] movtUnSelect = **TMovement**(1)
Des-seleccionar: La pieza vuelve desde la posición Destacada a la de reposo
Used in
TController.**DoNextMovement**, **TGLPiece**.**doMovement**

[ui] NUMBER_OF_PIECES = 24
Número de Piezas
Used in
TGLBoard.**InitializePieces**, **TGLBorders**.**Create**, **TGLPieces**.**Create**, **TGLPlayFieldSquares**.**Create**, **TPieces**.**Create**, **TSquares**.**Create**

[ui] NUMBER_OF_SQUARES = **NUMBER_OF_PIECES**
Número de Casilleros

[ui] pccBlack = **TPieceColor**(1)
Piezas Negras
Used in
TBoard.**Create**, **TController**.**InitGame**, **TGLBoard**.**InitializePlayers**, **TMainForm**.**btnStartGameClick**, **InitNextMoveMatrix**, **setPiecesColors**

[ui] pccWhite = **TPieceColor**(0)
Piezas Blancas
Used in
TBoard.**Create**, **TBoard**.**InitializePlayers**, **TController**.**Create**, **TController**.**InitGame**, **TGLBoard**.**InitializePlayers**, **TGLStartPositionSquare**.**setPieceColor**, **TGLStartPositionSquare**.**setStartPosition**, **InitNextMoveMatrix**, **setPiecesColors**

[ui] sdBottom = **TSide**(2)
Abajo
Used in
TBoard.**InitializePieces**, **TBoard**.**InitializeSquares**, **TGLBoard**.**InitializePieces**, **setInitialSquaresSideColors**, **setPiecesSideColors**

[ui] sdcBlue = **TSideColor**(0)
Lado Azul
Used in
TGLBoard.**InitializeSquares**, **TGLSide**.**SetSideColor**, **setInitialSquaresSideColors**, **setPiecesSideColors**

[ui] sdcNone = **TSideColor**(3)
No se ha asignado ningún color (para casilleros solamente)
Used in
TSquare.**CanPlayPiece**, **setInitialSquaresSideColors**
[ui] sdcRed = **TSideColor**(1)
Lado Rojo
Used in
TGLBoard.**InitializeSquares**, **TGLSide**.**SetSideColor**, **setInitialSquaresSideColors**, **setPiecesSideColors**
[ui] sdcYellow = **TSideColor**(2)
Lado Amarillo
Used in
TGLBoard.**InitializeSquares**, **TGLSide**.**SetSideColor**, **setInitialSquaresSideColors**, **setPiecesSideColors**
[ui] sdLeft = **TSide**(3)
Izquierda
Used in
TBoard.**InitializePieces**, **TBoard**.**InitializeSquares**, **TGLBoard**.**InitializePieces**, **setInitialSquaresSideColors**, **setPiecesSideColors**
[ui] sdRight = **TSide**(1)
Derecha
Used in
TBoard.**InitializePieces**, **TBoard**.**InitializeSquares**, **TGLBoard**.**InitializePieces**, **setInitialSquaresSideColors**, **setPiecesSideColors**
[ui] sdTop = **TSide**(0)
Arriba
Used in
TBoard.**InitializePieces**, **TBoard**.**InitializeSquares**, **TGLBoard**.**InitializePieces**, **setInitialSquaresSideColors**, **setPiecesSideColors**

Simple Types:

[ui] TMovement = (**movtSelect**, **movtUnSelect**, **movtReturnToHome**, **movtGoOverSquare**, **movtRotateLeft**, **movtRotateRight**, **movtAcceptPiece**)
Comandos
Enumeración de los posibles comandos que dispone el usuario para indicar el movimiento a realizar
[ui] TPieceColor = (**pccWhite**, **pccBlack**)
Color de la Pieza.
Enumeración de los posibles colores de Piezas / jugadores, es decir blancas o negras
Used in
TController.**InitGame**
[ui] TSide = (**sdTop**, **sdRight**, **sdBottom**, **sdLeft**)
Lados
Enumeración de los 4 lados posibles
[ui] TSideColor = (**sdcBlue**, **sdcRed**, **sdcYellow**, **sdcNone**)
Color de Lado.
Enumeración de los posibles colores a asignar a cada lado de los objetos de la clase TSides

Functions:

[ui] function extractNameNum(aNameStr: String): integer
Extrae el Id del objeto a partir del nombre
Parameters
aNameStr

Nombre del Objeto
Result
Id del Objeto
Called by
TController.**EventOccursPieceSelectedSt**,
TController.**EventOccursSquareSelectedSt**, **TController**.**MovePieceOverSquare**,
TController.**SelectPiece**, **TController**.**SetNewPickedObject**

[ui] function extractNameStr(aNameStr: String): String
Extrae la denominación de la clase a partir del nombre
Parameters
aNameStr
Nombre del Objeto
Result
Denominación de la clase
Called by
TController.**EventOccursBeforePieceSelectedSt**,
TController.**EventOccursPieceSelectedSt**,
TController.**EventOccursSquareSelectedSt**, **TController**.**SetNewPickedObject**,
TMainForm.**getGLObjectByName**

[ui] function getNameStr(aClassStr: String; aId: integer): String
Nombre del Objeto a partir de la denominacion de la clase y del identificador asignado.
Devuelve el nombre a partir de los parámetros
Parameters
aClassStr
Nombre de la clase
aId
Identificador dentro de la clase

Result
Devuelve el nombre armado
Called by
TController.**DoNextMovement**, **TController**.**DoOpponentMove**,
TController.**EventOccursSquareSelectedSt**,
TController.**TimeOutBeforePieceSelectedSt**,
TController.**TimeOutPieceSelectedSt**, **TController**.**TimeOutSquareSelectedSt**,
TGLBorders.**Create**, **TGLPieces**.**Create**, **TGLPlayFieldSquares**.**Create**,
TGLRotationArrows.**SetNames**

[ul] initialization

[ul] procedure setInitialSquaresSideColors
Establece el Color de cada lado de cada casillero.
Called by
initialization

[ul] procedure setPiecesColors
Asigna el color de pieza (blanco o negro) a cada una de ellas
Called by
initialization

[ul] procedure setPiecesSideColors

Asigna el color de cada lado a cada pieza
Called by
Initialization

AIII.2 Unit Controller

Used Units:

Used in interface: System, SysInit, Classes, ExtCtrls, **Model**, GLScene, Controls, **LogToFile**, **Common**, UObjectList
Used in implementation: IniFiles, Math, SysUtils, Dialogs, Forms, TypInfo, ShellApi

Used by:
In Interface of Units: FMain

Description:

Esta Unidad contiene todas las clases correspondientes al SubModelo de Diseño Controlador, que será el que interactúe entre la vista y el modelo. Su clase principal que en análisis llamamos Controlador (Controller), recibe los mensajes de interacción del usuario y los canaliza por sí mismo o con ayuda del modelo. Asimismo cubrirá las tareas de coordinación de la tutoría del sistema a través de eventos de ayuda al usuario. Los otros eventos que disparará para la vista serán el de realizar movimiento, el de manifestación de error y el de juego finalizado.

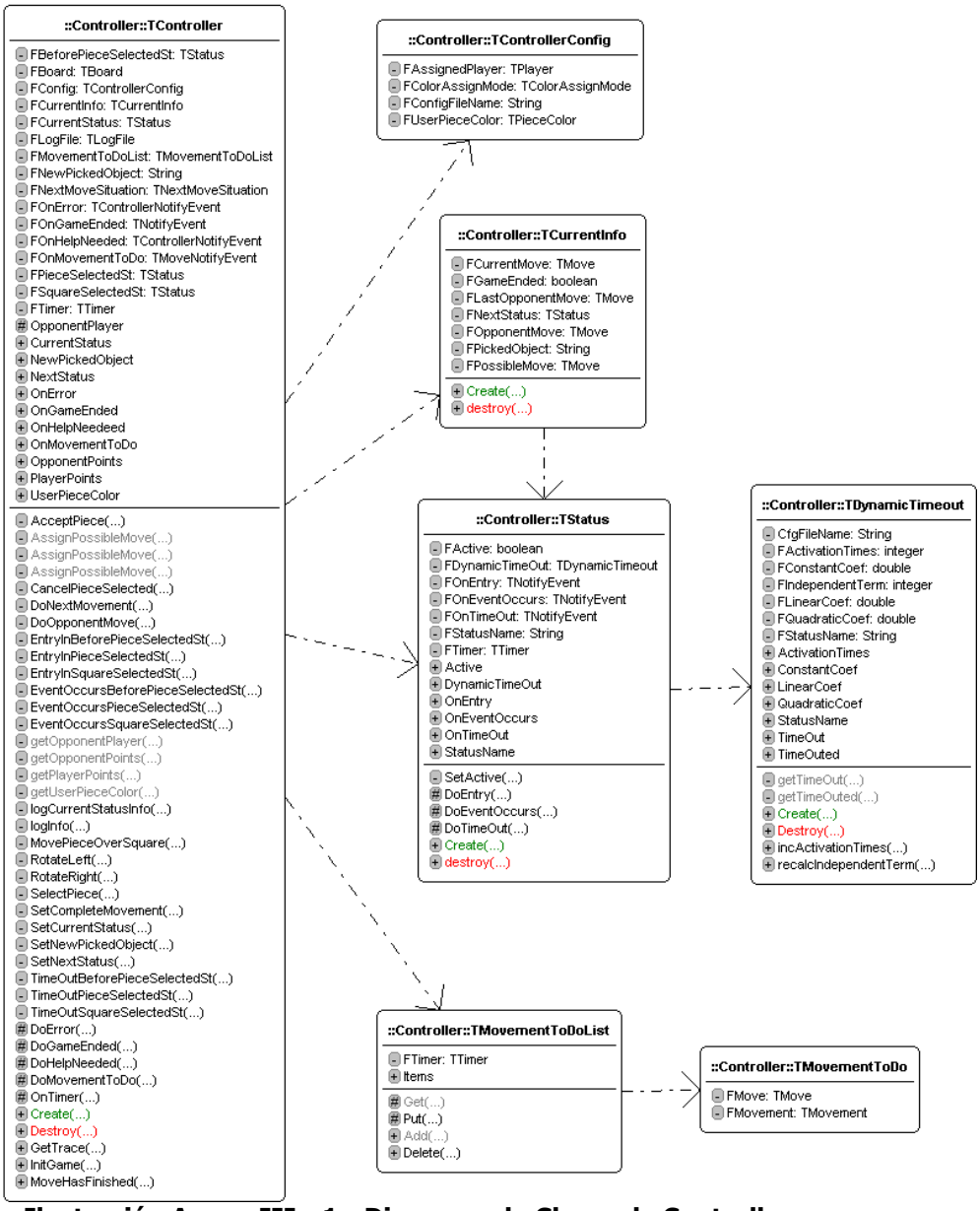


Ilustración Anexo III - 1 - Diagrama de Clases de Controller

List of Variables:

Global Variables: *None*
File-Local Variables: NextMoveMatrix

List of Constants:

Global Constants: camAllwaysBlack, camAllwaysWhite, camAlternated, camRamdonly, lgtError, lgtErrorForThisStatus, lgtGameEnded, lgtHelpNeeded, lgtLeftRotation, lgtMovementInAdvance, lgtOpponentMove, lgtPieceAcceptation, lgtPieceCancelation, lgtPieceColorAssign, lgtPieceSelection, lgtPlayModeAssign, lgtRightRotation, lgtSquareAndPieceCancelation, lgtSquareChanging, lgtSquareSelection, lgtStatusTimeOutInitParams, nmsGameContinue, nmsGameEnded,

nmsOneMoveForLoss, **nmsOneMoveForWin**
File-Local Constants: *None*

List of Simple Types:

Global Simple Types: TColorAssignMode, **TControllerNotifyEvent**, **TLogType**, **TMoveNotifyEvent**, **TNextMoveSituation**
File-Local Simple Types: *None*

List of Classes:

Global Classes: TController, **TControllerConfig**, **TCurrentInfo**, **TDynamicTimeout**, **TMovementToDo**, **TMovementToDoList**, **TStatus**
File-Local Classes: *None*

List of Functions:

Global Functions: *File-Local None*
File-Local Functions:
[ul] procedure initialization
[ul] procedure InitNextMoveMatrix
Inicializa la Matriz de Condiciones y Situaciones para usar en la próxima movida

Variables:

[ul] NextMoveMatrix: **array** [**TPieceColor**, Boolean, boolean] **of TNextMoveSituation**
Matriz de Codiciones y Situaciones a usar en la próxima movida que incluye a
- Color de la Pieza
- ¿ Usuario puede seguir jugando ? - ¿ Oponente Jugó ?

Used in
TController.**EventOccursSquareSelectedSt**, **InitNextMoveMatrix**

Constants:

[ui] camAllwaysBlack = **TColorAssignMode**(1)
Siempre Negras
Used in
TController.**InitGame**
[ui] camAllwaysWhite = **TColorAssignMode**(0)
Siempre Blancas
Used in
TController.**InitGame**
[ui] camAlternated = **TColorAssignMode**(2)
Alternadas (una partida blancas, la siguiente negras y así siguiendo)
Used in
TController.**InitGame**
[ui] camRamdonly = **TColorAssignMode**(3)
Decidir aleatóreamente el color de piezas
Used in
TController.**InitGame**
[ui] lgtError = **TLogType**(3)
El Objeto señalado inadecuado
Used in
TController.**SetNewPickedObject**

[ui] lgtErrorForThisStatus = **TLogType**(4)
El Objeto señalado es inadecuado para el estado actual (pero podría ser válido en otro estado)
Used in
TController.**EventOccursBeforePieceSelectedSt**,
TController.**EventOccursSquareSelectedSt**

[ui] lgtGameEnded = **TLogType**(16)
Finalizó la partida
Used in
TController.**EventOccursSquareSelectedSt**

[ui] lgtHelpNeeded = **TLogType**(15)
Ayuda necesaria
Used in
TController.**DoHelpNeeded**

[ui] lgtLeftRotation = **TLogType**(12)
Se indica rotar a la izquierda
Used in
TController.**EventOccursSquareSelectedSt**

[ui] lgtMovementInAdvance = **TLogType**(11)
Se adelanta en el movimiento
Used in
TController.**SetNewPickedObject**

[ui] lgtOpponentMove = **TLogType**(14)
Movida del oponente
Used in
TController.**DoOpponentMove**

[ui] lgtPieceAcceptation = **TLogType**(7)
Se acepta la pieza seleccionada en el casillero seleccionado
Used in
TController.**EventOccursSquareSelectedSt**

[ui] lgtPieceCancelation = **TLogType**(8)
Se cancela la pieza seleccionada
Used in
TController.**EventOccursPieceSelectedSt**,
TController.**EventOccursSquareSelectedSt**

[ui] lgtPieceColorAssign = **TLogType**(1)
Color de piezas asignadas al jugador
Used in
TController.**InitGame**

[ui] lgtPieceSelection = **TLogType**(5)
Se selcciona una pieza
Used in
TController.**EventOccursBeforePieceSelectedSt**,
TController.**EventOccursPieceSelectedSt**,
TController.**EventOccursSquareSelectedSt**

[ui] lgtPlayModeAssign = **TLogType**(2)
Modo de juego de usuario y oponente
Used in
TController.**Create**

[ui] lgtRightRotation = **TLogType**(13)
Se indica rotar a la derecha
Used in
TController.**EventOccursSquareSelectedSt**

[ui] lgtSquareAndPieceCancelation = **TLogType**(10)
Se cancela la selección del casillero y de la pieza
[ui] lgtSquareChanging = **TLogType**(9)
Se cambia el casillero seleccionado por otro
[ui] lgtSquareSelection = **TLogType**(6)
Se selecciona un casillero
Used in
TController.**EventOccursPieceSelectedSt**,
TController.**EventOccursSquareSelectedSt**
[ui] lgtStatusTimeOutInitParams = **TLogType**(0)
Parámetros iniciales del TimeOut correspondiente al estado referido
Used in
TController.**Create**
[ui] nmsGameContinue = **TNextMoveSituation**(0)
Puede Continuar <A.C.: PC>
Used in
TController.**Create**, **TController**.**EventOccursSquareSelectedSt**,
InitNextMoveMatrix
[ui] nmsGameEnded = **TNextMoveSituation**(1)
Finalizó <A.C.: F>
Used in
TController.**EventOccursSquareSelectedSt**, **InitNextMoveMatrix**
[ui] nmsOneMoveForLoss = **TNextMoveSituation**(3)
Ultima Chance de jugar del Oponente <A.C.: UJO>
Used in
TController.**EventOccursSquareSelectedSt**, **InitNextMoveMatrix**
[ui] nmsOneMoveForWin = **TNextMoveSituation**(2)
Ultima Jugada del Usuario <A.C.: UJU> <A.C.: Controlador :: Ultima Jugada del Usuario>
Used in
TController.**EventOccursSquareSelectedSt**, **InitNextMoveMatrix**

Simple Types:

[ui] TColorAssignMode = (**camAllwaysWhite**, **camAllwaysBlack**, **camAlternated**, **camRamdonly**)
Modo de Asignación de Color de Piezas
Enumeración de los posibles modos de asignaciones del color de piezas al usuario. Cada caso representa una posible asignación configurable por el investigador de acuerdo a su criterio.
[ui] TControllerNotifyEvent = **procedure** (Sender: TObject; **const** aObjectName: String) **of object**
Evento de notificación con parámetros dado que TNotifyEvent no contiene parámetros adicionalesal Sender
Parameters
Sender
Objeto que envía la notificación
aObjectName
Nombre del objeto al que se hace referencia con la notificación del evento
[ui] TLogType = (**lgtStatusTimeOutInitParams**, **lgtPieceColorAssign**, **lgtPlayModeAssign**, **lgtError**, **lgtErrorForThisStatus**, **lgtPieceSelection**, **lgtSquareSelection**, **lgtPieceAcceptation**, **lgtPieceCancelation**, **lgtSquareChanging**, **lgtSquareAndPieceCancelation**, **lgtMovementInAdvance**, **lgtLeftRotation**, **lgtRightRotation**, **lgtOpponentMove**, **lgtHelpNeeded**, **lgtGameEnded**)

Tipo de registro
Enumeración de los tipos de registro. Los mismos fueron obtenidos de los diagramas de secuencias realizados en el análisis.
Used in
TController.**logInfo**

[ui] TMoveNotifyEvent = **procedure** (Sender: TObject; **const** aPieceName, aSquareName: String; **const** aMovementType: **TMovement**) **of object**
Evento de notificación con parámetros dado que TNotifyEvent no contiene parámetros adicionalesal Sender
Parameters
Sender
Objeto que envía la notificación
aPieceName
Nombre de la Pieza que debe moverse
aSquareName
Nombre del Casillero de referencia
aMovementType
Tipo de movimiento a realizar por la pieza

[ui] TNextMoveSituation = (**nmsGameContinue**, **nmsGameEnded**, **nmsOneMoveForWin**, **nmsOneMoveForLoss**)
Situaciones Posibles para la próxima movida
Enumeración de los posibles Situaciones en que queda la partida luego de realizarse (o no) las movidas actuales por parte del usuario y del oponente.

Functions:

[ul] initialization

[ul] procedure InitNextMoveMatrix
Inicializa la Matriz de Condiciones y Situaciones para usar en la próxima movida
Called by
initialization

AIII.2.1 Class TController

TObject <-- **TController**
Declared in Unit **Controller**

No known Subclasses

Used by Classes:

TMainForm

Description:

Controlador <A.C.: Controlador>
Clase que tendrá una única instancia y que será la responsable de controlar el juego. Respetará el diagrama de estados analizado. El objeto creado será el responsable de conectar la Vista con el Modelo. Con la vista (por quien es creado) se comunicara a través de cuatro eventos a los cuales deberá registrarse para conocer el momento en que ocurre alguno de ellos, a saber:
- Juego finalizado (OnGameEnded) - Error (OnError) - Necesita ayuda (OnHelpNeedeed) - Movimiento a Realizar (OnMovementToDo)

Used in
TMainForm.**btnStartGameClick**, **TMainForm**

List of Constructors:

[+] constructor Create
Hereda y crea a FBoard, FCurrentInfo, FOpponentMove, FPossibleMove, FMovementToDoList, FConfig, FTimer y a cada uno de los estados.

List of Destructors:

[+] destructor Destroy
Destruye a FTimer, FBoard, FLogInfo, FCurrentInfo, FConfig, FMovementToDoList y a cada uno de los estados y hereda

List of Methods:

[-] procedure AcceptPiece
Acepta Pieza <A.C.: No Definido> Se Juega la pieza.

[-] function AssignPossibleMove
Asigna una Movida Posible para la pieza dada.

[-] function AssignPossibleMove
Asigna una movida Posible.

[-] function AssignPossibleMove
Asigna una Movida Posible para la pieza dada en un casillero dado.

[-] procedure CancelPieceSelected
Cancela la pieza seleccionada <A.C.: Controlador :: Cancelar Pieza> Se agrega a la lista de acciones la cancelacción de la pieza y se asigna el próximo estado (FBeforePieceSelectedSt)

[#] procedure DoError
Establecer evento de error Activa el evento OnError si fue asignado (deberá asignarse para poder "avisarle" al objeto visual que ejecute ShowError)

[#] procedure DoGameEnded
Establecer evento Juego Finalizado.

[#] procedure DoHelpNeeded
Establecer Evento Usuario necesita ayuda.

[#] procedure DoMovementToDo
Establecer evento Movida a realizar Activa el evento OnMovementToDo (deberá asignarse para poder "avisarle" al objeto visual que ejecute DoMovement)

[-] procedure DoNextMovement
Realiza el próximo movimiento.

[-] procedure DoOpponentMove
Realizar Jugada del Oponente <A.C.: Controlador :: Realizar Jugada del Oponente> Invoca SetCompleteMovement con el movimiento del oponente o de la última jugada del oponente de acuerdo al parámetro recibido.

[-] procedure EntryInBeforePieceSelectedSt
Implementa el evento EntryIn del Estado EntryInBeforePieceSelectedSt

[-] procedure EntryInPieceSelectedSt
Implementa el evento EntryIn del Estado FPieceSelectedSt

[-] procedure EntryInSquareSelectedSt
Implementa el evento EntryIn del Estado FSquareSelectedSt

[-] procedure EventOccursBeforePieceSelectedSt
Implementa el evento EventOccurs del Estado FBeforePieceSelectedSt.

[-] procedure EventOccursPieceSelectedSt
Implementa el evento EventOccurs del Estado FPieceSelectedSt.

[-] procedure EventOccursSquareSelectedSt
Implementa el evento EventOccurs del Estado FSquareSelectedSt.

[-] function getOpponentPlayer
Obtiene el jugador oponente.

[-] function getOpponentPoints
Devuelve los puntos del oponente.
[-] function getPlayerPoints
Devuelve los puntos del usuario.
[+] procedure GetTrace
Obtener Trazado <A.C.: Controlador :: Obtener Trazado> Abre el explorador de archivos en el directorio de las partidas.
[-] function getUserPieceColor
Devuelve el Color de piezas asigando al usuario
[+] procedure InitGame
Inicia la partida <A.C.: Controlador :: Iniciar Partida> Indica a FBoard que debe inicializar el tablero y cuales son las piezas que se asignaron al usuario, previamente obtendrá las piezas a asignar de acuerdo a lo configurado.
[-] procedure logCurrentStatusInfo
Guarda información para el trazado del evento actual para el estado actual.
[-] procedure logInfo
Guarda información para el trazado correspondiente al evento actual <A.C.: Registrar evento>.
[+] procedure MoveHasFinished
Mensaje que indica que el movimiento de piezas ordenado finalizó.
[-] procedure MovePieceOverSquare
Indica el traslado de la pieza encima del casillero seleccionado <A.C.: Controlador :: Seleccionar Casillero> Se agrega a la lista de acciones el traslado de la pieza hacie el casillero y se asigna el próximo estado (FSquareSelectedSt)
[#] procedure OnTimer
Establece evento TimeOut.
[-] procedure RotateLeft
Indica la rotación de la pieza hacia la izquierda <A.C.: No Definido> Se agrega a la lista de acciones la rotación de la pieza hacia la izquierda
[-] procedure RotateRight
Indica la rotación de la pieza hacia la derecha <A.C.: No Definido> Se agrega a la lista de acciones la rotación de la pieza hacia la derecha
[-] procedure SelectPiece
Seleccionar Pieza <A.C.: Controlador :: Seleccionar Pieza> Se agrega a la lista de acciones la selección de la pieza y se asigna el próximo estado (FPieceSelectedSt)
[-] procedure SetCompleteMovement
Realiza una jugada completa <A.C.: No Definido> Sirve para realizar la jugada del oponente o eventualmente podría utilizarse en un futuro simulador de partidas.
[-] procedure SetCurrentStatus
Establece el Estado Actual.
[-] procedure SetNewPickedObject
Establece el objeto indicado por el usuario.
[-] procedure SetNextStatus
Establece el próximo estado.
[-] procedure TimeOutBeforePieceSelectedSt
Implementa el evento TimeOut del Estado FBeforePieceSelectedSt <A.C.: Controlador :: Atender TimeOut>.
[-] procedure TimeOutPieceSelectedSt
Implementa el evento TimeOut del Estado FPieceSelectedSt <A.C.: Controlador :: Atender TimeOut>
Si se puede jugar la pieza seleccionada, se indica al casillero posible que muestre ayuda
Sino, se indicara que la pieza posible muestre ayuda
Finalmente se invocará a DoHelpNeeded con el objeto determinade según el caso

[-] procedure TimeOutSquareSelectedSt

Implementa el evento TimeOut del Estado FSquareSelectedSt <A.C.: Controlador :: Atender TimeOut>.

Fields:

[-] FBeforePieceSelectedSt: **TStatus**

Estado Antes de Seleccionar la Pieza <S.D.: Antes de Seleccionar la Pieza>

Used in

AcceptPiece, **CancelPieceSelected**, **Create**, **Destroy**, **EventOccursSquareSelectedSt**, **MoveHasFinished**

[-] FBoard: **TBoard**

Referencia al Tablero de Juego <A.C.: Controlador :: Tablero>

Used in

AcceptPiece, **AssignPossibleMove**, **AssignPossibleMove**, **Create**, **Destroy**, **EventOccursPieceSelectedSt**, **EventOccursSquareSelectedSt**, **getOpponentPlayer**, **InitGame**, **MovePieceOverSquare**, **SelectPiece**, **SetCompleteMovement**, **SetNewPickedObject**

[-] FConfig: **TControllerConfig**

Configurador del controlador <A.C.: Controlador :: No definido>

Used in

AssignPossibleMove, **AssignPossibleMove**, **Create**, **Destroy**, **EventOccursSquareSelectedSt**, **getOpponentPlayer**, **getPlayerPoints**, **getUserPieceColor**, **InitGame**, **SetNewPickedObject**, **TimeOutSquareSelectedSt**

[-] FCurrentInfo: **TCurrentInfo**

Información actual del controlador <A.C.: Controlador :: No definido>

Used in

AcceptPiece, **AssignPossibleMove**, **AssignPossibleMove**, **AssignPossibleMove**, **CancelPieceSelected**, **Create**, **Destroy**, **DoOpponentMove**, **EventOccursBeforePieceSelectedSt**, **EventOccursPieceSelectedSt**, **EventOccursSquareSelectedSt**, **InitGame**, **MoveHasFinished**, **MovePieceOverSquare**, **OnTimer**, **RotateLeft**, **RotateRight**, **SelectPiece**, **SetNewPickedObject**, **SetNextStatus**, **TimeOutBeforePieceSelectedSt**, **TimeOutPieceSelectedSt**, **TimeOutSquareSelectedSt**

[-] FCurrentStatus: **TStatus**

Estado actual <A.C.: Controlador :: Estado Actual> Referencia a uno de los tres estados posibles en que se puede encontrar la partida

Used in

logCurrentStatusInfo, **SetCurrentStatus**, **SetNewPickedObject**

[-] FLogFile: **TLogFile**

Objeto que administra el trazado de la partida <A.C. No Definida>

Used in

Create, **Destroy**, **logInfo**

[-] FMovementToDoList: **TMovementToDoList**

Lista a utilizar como cola (First In - First Out) de movimientos a realizar que se van a solicitar al modelo <A.C. No Definida>

Used in

AcceptPiece, **CancelPieceSelected**, **Create**, **Destroy**, **DoNextMovement**, **InitGame**, **MoveHasFinished**, **MovePieceOverSquare**, **RotateLeft**, **RotateRight**, **SelectPiece**, **SetCompleteMovement**, **SetNewPickedObject**, **SetNextStatus**

[-] FNewPickedObject: String
Nombre del último objeto señalado. En el evento OnEventOccurs correspondiente, se determinará si corresponde reemplazar a FPickedObject o no
Used in
AcceptPiece, **DoError**, **EventOccursBeforePieceSelectedSt**, **EventOccursPieceSelectedSt**, **EventOccursSquareSelectedSt**, **MovePieceOverSquare**, **SelectPiece**, **SetNewPickedObject**

[-] FNextMoveSituation: **TNextMoveSituation**
Situación evaluada para el próximo par de movidas
Used in
Create, **EventOccursSquareSelectedSt**

[-] FOnError: **TControllerNotifyEvent**
Referencia al evento OnError. Indica que el usuario ha señalado un objeto equivocado
Used in
DoError

[-] FOnGameEnded: TNotifyEvent
Referencia al evento OnGameEnded. <A.C.: Controlador :: No Definido - Reemplaza la funcionalidad del atributo Timer de Reinicio que pertenece al Formulario Principal (MainFrm.tmrForReinitialize
Used in
DoGameEnded

[-] FOnHelpNeeded: **TControllerNotifyEvent**
Referencia al evento OnHelpNeeded. Indica que el usuario necesita ayuda sobre un objeto dado
Used in
DoHelpNeeded

[-] FOnMovementToDo: **TMoveNotifyEvent**
Referencia al evento OnMovementToDo. Indica que deberá realizarse un movimiento de una pieza dada
Used in
DoMovementToDo

[-] FPieceSelectedSt: **TStatus**
Estado Pieza Seleccionada <S.D.: Pieza Seleccionada>
Used in
AcceptPiece, **Create**, **Destroy**, **SelectPiece**

[-] FSquareSelectedSt: **TStatus**
Estado Casillero Seleccionado <S.D.: Casillero Seleccionado>
Used in
AcceptPiece, **Create**, **Destroy**, **MovePieceOverSquare**, **RotateLeft**, **RotateRight**

[-] FTimer: TTimer

Contador del Tiempo del Controlador para mostrar ayuda al usuario que será referenciado por todos los objetos de la clase TStatus creados <A.C.: Controlador :: Timer>

Used in

Create, **Destroy**

Properties:

[+] property CurrentStatus: **TStatus read FCurrentStatus write SetCurrentStatus**

Estado actual <A.C.: Controlador :: Estado Actual> Referencia a uno de los tres estados posibles en que se puede encontrar la partida

Used in

Create, **OnTimer**, **SetNextStatus**, **TMainForm**.**GLSceneViewerMouseDown**

[+] property NewPickedObject: String **read FNewPickedObject write SetNewPickedObject**

Nombre del último objeto señalado. En el evento OnEventOccurs correspondiente, se determinará si corresponde reemplazar a FPickedObject o no

Used in

TMainForm.**GLSceneViewerMouseDown**

[+] property NextStatus: **TStatus write SetNextStatus**

Used in

CancelPieceSelected, **EventOccursSquareSelectedSt**, **MovePieceOverSquare**, **RotateLeft**, **RotateRight**, **SelectPiece**

[+] property OnError: **TControllerNotifyEvent read FOnError write FOnError**

Referencia al evento OnError. Indica que el usuario ha señalado un objeto equivocado

Used in

TMainForm.**btnStartGameClick**

[+] property OnGameEnded: TNotifyEvent **read FOnGameEnded write FOnGameEnded**

Referencia al evento OnGameEnded. <A.C.: Controlador :: No Definido - Reemplaza la funcionalidad del atributo Timer de Reinicio que pertenece al Formulario Principal (MainFrm.tmrForReinitialize

Used in

TMainForm.**btnStartGameClick**

[+] property OnHelpNeedeed: **TControllerNotifyEvent read FOnHelpNeeded write FOnHelpNeeded**

Referencia al evento OnHelpNeeded. Indica que el usuario necesita ayuda sobre un objeto dado

Used in

TMainForm.**btnStartGameClick**

[+] property OnMovementToDo: **TMoveNotifyEvent read FOnMovementToDo write FOnMovementToDo**

Referencia al evento OnMovementToDo. Indica que deberá realizarse un movimiento de una pieza dada

Used in

TMainForm.**btnStartGameClick**

[#] [ro] property OpponentPlayer: **TPlayer read getOpponentPlayer**
Obtiene el jugador oponente. Devuelve el oponente invocando la funcion FBoard.OpponentPlayer
Used in
EventOccursSquareSelectedSt, **getOpponentPoints**, **InitGame**

[+] [ro] property OpponentPoints: integer **read getOpponentPoints**
Devuelve los puntos del oponente.
Used in
EventOccursSquareSelectedSt, **TMainForm**.**GLSceneViewerMouseDown**

[+] [ro] property PlayerPoints: integer **read getPlayerPoints**
Devuelve los puntos del usuario.
Used in
EventOccursSquareSelectedSt, **TMainForm**.**GLSceneViewerMouseDown**

[+] [ro] property UserPieceColor: **TPieceColor read getUserPieceColor**
Devuelve el Color de piezas asigando al usuario
Used in
TMainForm.**btnStartGameClick**

Constructors:

[+] constructor Create(aParentOwner: TComponent)
Hereda y crea a FBoard, FCurrentInfo, FOpponentMove, FPossibleMove, FMovementToDoList, FConfig, FTimer y a cada uno de los estados. El estado inicial, no se implementa como estado.
Parameters
aParentOwner
Componente que permite crear al FTimer para poder asociarle un owner
Called by
TMainForm.**btnStartGameClick**

Destructors:

[+] destructor Destroy; **override**
Destruye a FTimer, FBoard, FLogInfo, FCurrentInfo, FConfig, FMovementToDoList y a cada uno de los estados y hereda
Overrides
Destroy

Methods:

[-] procedure AcceptPiece
Acepta Pieza <A.C.: No Definido> Se Juega la pieza. Se agrega a la lista de acciones la aceptación de la pieza y se recalculan los términos independientes de cada TimeOut de cada Estado invocando los respectivos recalcIndependentTerm
Called by
EventOccursSquareSelectedSt

[-] function AssignPossibleMove(aPiece: **TPiece**): boolean; **overload**

Asigna una Movida Posible para la pieza dada. Esta segunda implementación polimórfica da solución al caso de tener restringida la pieza en la selección de la movida posible.

Parameters

aPiece

Pieza que se desea evaluar si se puede ubicar en el tablero

Result

Devuelve true si pudo asignar una posible Movida

[-] function AssignPossibleMove: boolean; **overload**

Asigna una movida Posible. Esta es la implementación más libre de las tres existentes, ya que no recibe parámetros, con lo cual se asignará una jugada posible de acuerdo al tipo de jugador asignado a FBoard.

Result

Devuelve true si pudo asignar una posible Movida

Called by

TimeOutBeforePieceSelectedSt, **TimeOutPieceSelectedSt**, **TimeOutSquareSelectedSt**

[-] function AssignPossibleMove(aPiece: **TPiece**; aSquare: **TSquare**): boolean; **overload**

Asigna una Movida Posible para la pieza dada en un casillero dado. Esta tercera implementación polimórfica da solución al caso de tener restringida la pieza y el casillero en la selección de la movida posible.

Parameters

aPiece

Pieza que se desea evaluar si se puede ubicar en el tablero

aSquare

Casillero que se desea evaluar si se puede ubicar en el tablero

Result

Devuelve true si pudo asignar una posible Movida

[-] procedure CancelPieceSelected

Cancela la pieza seleccionada <A.C.: Controlador :: Cancelar Pieza> Se agrega a la lista de acciones la cancelacción de la pieza y se asigna el próximo estado (FBeforePieceSelectedSt)

Called by

EventOccursPieceSelectedSt, **EventOccursSquareSelectedSt**

[#] procedure DoError(**const** aObjectName: String); **dynamic**

Establecer evento de error Activa el evento OnError si fue asignado (deberá asignarse para poder "avisarle" al objeto visual que ejecute ShowError)

Parameters

aObjectName

Nombre del objeto que debe mostrar el error

Called by

EventOccursBeforePieceSelectedSt, **EventOccursSquareSelectedSt**, **SetNewPickedObject**

[#] procedure DoGameEnded; **dynamic**

Establecer evento Juego Finalizado. Activa el evento OnGameEnded si fue asignado (deberá asignarse para habilitar el botón de reinicio)

Called by

MoveHasFinished

[#] procedure DoHelpNeeded(**const** aObjectName: String); **dynamic**

Establecer Evento Usuario necesita ayuda. Activa el evento OnHelpNeeded si fue asignado (deberá asignarse para poder "avisarle" al objeto visual que ejecute ShowHelp)

Parameters

aObjectName

Nombre del objeto que debe mostrar la ayuda

Called by

TimeOutBeforePieceSelectedSt, **TimeOutPieceSelectedSt**, **TimeOutSquareSelectedSt**

[#] procedure DoMovementToDo(**const** aPieceName, aSquareName: String; aMovementType: **TMovement**); **dynamic**

Establecer evento Movida a realizar Activa el evento OnMovementToDo (deberá asignarse para poder "avisarle" al objeto visual que ejecute DoMovement)

Parameters

aPieceName

Nombre de la pieza

aSquareName

Nombre del casillero (si no es de aplicación, estará vacío)

aMovementType

Tipo de movimiento a realizar

Called by

DoNextMovement

[-] procedure DoNextMovement

Realiza el próximo movimiento. Toma el primer movimiento de la lista de movimientos a realizar e invoca a DoMovementToDo con los parámetros correspondientes en cada caso.

Called by

MoveHasFinished, **SetNextStatus**

[-] procedure DoOpponentMove(aIsLast: boolean)

Realizar Jugada del Oponente <A.C.: Controlador :: Realizar Jugada del Oponente> Invoca SetCompleteMovement con el movimiento del oponente o de la última jugada del oponente de acuerdo al parámetro recibido. Guarda la información en el registro

Parameters

aIsLast

True para el caso de ser la última movida

Called by

EventOccursSquareSelectedSt, **InitGame**

[-] procedure EntryInBeforePieceSelectedSt(Sender: TObject)

Implementa el evento EntryIn del Estado EntryInBeforePieceSelectedSt

Parameters

Sender

Objeto que envía el mensaje

Called by

Create

[-] procedure EntryInPieceSelectedSt(Sender: TObject)

Implementa el evento EntryIn del Estado FPieceSelectedSt

Parameters

Sender

Objeto que envía el mensaje
Called by
Create

[-] procedure EntryInSquareSelectedSt(Sender: TObject)
Implementa el evento EntryIn del Estado FSquareSelectedSt
Parameters
Sender
Objeto que envía el mensaje
Called by
Create

[-] procedure EventOccursBeforePieceSelectedSt(Sender: TObject)
Implementa el evento EventOccurs del Estado FBeforePieceSelectedSt. El único objeto válido es una pieza propia no utilizada. Si no fuera así, se deberá registrar el hecho y disparar el error. Caso contrario, se registra la selección de la pieza y se realiza el procedimiento correspondiente (SelectPiece)
Parameters
Sender
Objeto que envía el mensaje
Called by
Create

[-] procedure EventOccursPieceSelectedSt(Sender: TObject)
Implementa el evento EventOccurs del Estado FPieceSelectedSt. Si el objeto seleccionado es un casillero de juego, se registra y se invoca MovePieceOverSquare. Sino, se cancela y se registra. Además, si el objeto seleccionado es una pieza válida, se selecciona la misma y se registra.
Parameters
Sender
Objeto que envía el mensaje
Called by
Create

[-] procedure EventOccursSquareSelectedSt(Sender: TObject)
Implementa el evento EventOccurs del Estado FSquareSelectedSt. Si es el indicador de giro a la izquierda, se rota la pieza hacia la izquierda invocando RotateLeft y se registra
Si es el indicador de giro a la derechaa, se rota la pieza hacia la derecha invocando RotateRight y se registra
Si no es un casillero, Cancelar la movida con CancelPieceSelected y registrarlo. Si además es otra pieza, seleccionarla con SelectPiece y registrarlo.
Si es otro casillero, Mover la pieza sobre el casillero con MovePieceOverSquare y registrarlo
Si es la pieza seleccionada, si se puede jugar en ese casillero en la orientación actual, se acepta la pieza con AcceptPiece y se registra. Sino, se dispara el error y se registra. Se determina la condición de última jugada y de acuerdo a ella, se realiza la jugada del oponente una o dos según el caso y finaliza el juego o continúa esperando la seleccion de pieza del usuario.

Parameters
Sender
Objeto que envía el mensaje
Called by
Create

[-] function getOpponentPlayer: **TPlayer**

Obtiene el jugador oponente. Devuelve el oponente invocando la funcion FBoard.OpponentPlayer

Result

Referencia al objeto Jugador Oponente

[-] function getOpponentPoints: integer

Devuelve los puntos del oponente.

Result

Cantidad de piezas aceptadas por el oponente hasta el momento

[-] function getPlayerPoints: integer

Devuelve los puntos del usuario.

Result

Cantidad de piezas aceptadas por el usuario hasta el momento

[+] procedure GetTrace(aHandle: THandle)

Obtener Trazado <A.C.: Controlador :: Obtener Trazado> Abre el explorador de archivos en el directorio de las partidas.

Parameters

aHandle

Handle del form que invoca el procedimiento

Called by

TMainForm.**FormKeyDown**

[-] function getUserPieceColor: **TPieceColor**

Devuelve el Color de piezas asigando al usuario

Result

Color de piezas del usuario

[+] procedure InitGame

Inicia la partida <A.C.: Controlador :: Iniciar Partida> Indica a FBoard que debe inicializar el tablero y cuales son las piezas que se asignaron al usuario, previamente obtendrá las piezas a asignar de acuerdo a lo configurado.

Si al usuario se le asignaron las piezas negras, inmediatamente agragará la jugada correspondiente al oponente de piezas blancas.

Called by

TMainForm.**btnStartGameClick**

[-] procedure logCurrentStatusInfo(aLogType: **TLogType**; aLogMsg, aLogParams: String)

Guarda información para el trazado del evento actual para el estado actual. <A.C.: Registrar evento>. Invoca a logInfo con el estado actual como parámetro.

Parameters

aLogType

Tipo de Registro

aLogMsg

Descripción del Registro

aLogParams

Parametros del registro que ameritan indicarse en forma reconocible (cada parámetro va seguido del signo igual y su respectivo valor)

Called by

DoHelpNeeded, **EventOccursBeforePieceSelectedSt**, **EventOccursPieceSelectedSt**, **EventOccursSquareSelectedSt**, **SetNewPickedObject**

[-] procedure logInfo(aLogType: **TLogType**; aStatusName, aLogMsg, aLogParams: String)
Guarda información para el trazado correspondiente al evento actual <A.C.: Registrar evento>.
Parameters
aLogType
Tipo de Registro
aStatusName
Nombre del Estado
aLogMsg
Descripción del Registro
aLogParams
Parametros del registro que ameritan indicarse en forma reconocible (cada parámetro va seguido del signo igual y su respectivo valor)
Called by
Create, **DoOpponentMove**, **InitGame**, **logCurrentStatusInfo**

[+] procedure MoveHasFinished
Mensaje que indica que el movimiento de piezas ordenado finalizó. <A.C. Finalizó Desplazamiento y Finalizó Rotación (se unifican)> Cada vez que termine un movimiento, desde la vista se invocará este mensaje que según haya finalizado o no la partida, deberá borrar el primer movimiento de la lista de movimientos a realizar.
Called by
InitGame, **TMainForm**.**GLCadencerProgress**

[-] procedure MovePieceOverSquare
Indica el traslado de la pieza encima del casillero seleccionado <A.C.: Controlador :: Seleccionar Casillero> Se agrega a la lista de acciones el traslado de la pieza hacie el casillero y se asigna el próximo estado (FSquareSelectedSt)
Called by
EventOccursPieceSelectedSt, **EventOccursSquareSelectedSt**

[#] procedure OnTimer(Sender: TObject); **virtual**
Establece evento TimeOut. Ejecuta el evento timeOut en el caso que la partida no haya finalizado
Parameters
Sender
Objeto que envía el evento
Called by
Create

[-] procedure RotateLeft
Indica la rotación de la pieza hacia la izquierda <A.C.: No Definido> Se agrega a la lista de acciones la rotación de la pieza hacia la izquierda
Called by
EventOccursSquareSelectedSt

[-] procedure RotateRight

Indica la rotación de la pieza hacia la derecha <A.C.: No Definido> Se agrega a la lista de acciones la rotación de la pieza hacia la derecha

Called by

EventOccursSquareSelectedSt

[-] procedure SelectPiece

Seleccionar Pieza <A.C.: Controlador :: Seleccionar Pieza> Se agrega a la lista de acciones la selección de la pieza y se asigna el próximo estado (FPieceSelectedSt)

Called by

EventOccursBeforePieceSelectedSt, **EventOccursPieceSelectedSt**, **EventOccursSquareSelectedSt**

[-] procedure SetCompleteMovement(aMove: **TMove**)

Realiza una jugada completa <A.C.: No Definido> Sirve para realizar la jugada del oponente o eventualmente podría utilizarse en un futuro simulador de partidas. Consta de:
- Se determina hacia que lado rotar la pieza
- Se selecciona la pieza
- Se desplaza sobre el casillero
- Se rota, si es necesari hacia el lado determinado
- Se acepta la pieza

Parameters

aMove

Movida completa a realizar

Called by

DoOpponentMove

[-] procedure SetCurrentStatus(**const** Value: **TStatus**)

Establece el Estado Actual. Además, si está asignado el evento OnEntry, desactiva el estado anterior y activa el nuevo

Parameters

Value

Estado actual a asignar

[-] procedure SetNewPickedObject(**const** Value: String)

Establece el objeto indicado por el usuario. <A.C.: No Definido>
Además de establecer el nuevo objeto indicado por el usuario, realiza además el primer filtro de objetos a procesar en el estado actual, descartando a todos ellos que no sean:
- Piezas propias no utilizadas
- Casilleros de Juego desocupados
- Casilleros de posición inicial propios
- Indicadores de rotación.

Puede suceder que el jugador se adelante en sus movimientos. En este caso, no se dará curso a dicha indicación, se mostará como error y se registrará Finalmente, dispara el evento EventOccurs en el Estado actual (FCurrentStatus).

Parameters

Value

Es el nombre del nuevo objeto señalado por el usuario

[-] procedure SetNextStatus(**const** Value: **TStatus**)

Establece el próximo estado. Si no está en movimiento y la cantidad de movimientos a realizar es mayor a cero, se indica la realización de movimientos y se actualizará el Estado Actual CurrentStatus con el valor recibido

Parameters

Value

Nuevo próximo estado

[-] procedure TimeOutBeforePieceSelectedSt(Sender: TObject)

Implementa el evento TimeOut del Estado FBeforePieceSelectedSt <A.C.: Controlador :: Atender TimeOut>.

Indica a la pieza posible que muestre ayuda invocando a DoHelpNeeded

Parameters

Sender

Objeto que envía el mensaje

Called by

Create

[-] procedure TimeOutPieceSelectedSt(Sender: TObject)

Implementa el evento TimeOut del Estado FPieceSelectedSt <A.C.: Controlador :: Atender TimeOut>

Si se puede jugar la pieza seleccionada, se indica al casillero posible que muestre ayuda

Sino, se indicara que la pieza posible muestre ayuda

Finalmente se invocará a DoHelpNeeded con el objeto determinade según el caso

Parameters

Sender

Objeto que envía el mensaje

Called by

Create

[-] procedure TimeOutSquareSelectedSt(Sender: TObject)

Implementa el evento TimeOut del Estado FSquareSelectedSt <A.C.: Controlador :: Atender TimeOut>.

Si se puede aceptar la pieza seleccionada, se indica al casillero posible que muestre ayuda

Sino, si se pudiera aceptar la pieza previa rotación, se ayudará mediante el indicador de giro que menos cantidad de movimientos necesite realizar para dejar la pieza en condición de jugar

Sino (quiere decir que la pieza no puede jugarse en el casillero elegido de ninguna forma), si la pieza pudiera jugarse en otro casillero, se indicará ese otro casillero

Sino (quiere decir que esa pieza no puede jugar, es decir que ya quedó descartada para esta partida), si puede jugarse cualquier pieza, se indica a esa otra pieza que está en su posición de origen

Sino, deberá saltar una excepción, dado que no es una situación del juego posible

Finalmente se invocará a DoHelpNeeded con el objeto determinade según el caso

Parameters

Sender

Objeto que envía el mensaje

Called by

Create

AIII.2.2 Class TControllerConfig

TObject <-- **TControllerConfig**
Declared in Unit **Controller**

No known Subclasses

Used by Classes:

TController

Description:

Configurador de Controlador <A.C.: No definida>
Clase Auxiliar que contiene la información de configuración del controlador
Used in
TController.**Create**, **TController**

Fields:

[-] FAssignedPlayer: **TPlayer**
Referencia al Jugador de Piezas blancas o negras según haya sido asignado <A.C.: Controlador :: Jugador>
Used in
TController.**AssignPossibleMove**, **TController**.**AssignPossibleMove**, **TController**.**EventOccursSquareSelectedSt**, **TController**.**getOpponentPlayer**, **TController**.**getPlayerPoints**, **TController**.**InitGame**, **TController**.**SetNewPickedObject**, **TController**.**TimeOutSquareSelectedSt**

[-] FColorAssignMode: **TColorAssignMode**
Modo de asignación de piezas configurado <A.C. No Definida>
Used in
TController.**InitGame**

[-] FConfigFileName: String
Nombre del archivo de configuración
Used in
TController.**Create**, **TController**.**InitGame**

[-] FUserPieceColor: **TPieceColor**
Indica el color de piezas asignadas al usuario
Used in
TController.**Create**, **TController**.**EventOccursSquareSelectedSt**, **TController**.**getUserPieceColor**, **TController**.**InitGame**

AIII.2.3 Class TCurrentInfo

TObject <-- **TCurrentInfo**
Declared in Unit **Controller**

No known Subclasses

Used by Classes:

TController

Description:

Información actual del Controlador <A.C.: No definida>
Clase auxiliar que Contiene los atributos de información actual del controlador

Used in
TController.**Create**, **TController**

List of Constructors:

[+] constructor Create
Hereda y crea las movidas correspondientes

List of Destructors:

[+] destructor destroy
Destruye las movidas y hereda

Fields:

[-] FCurrentMove: **TMove**
Movida actual <A.C.: Controlador :: Pieza Seleccionada y Controlador :: Casillero Seleccionado> (Se agrega la orientación de la pieza, también incluida en el objeto de la clase TMove);
Used in
Create, **destroy**, **TController**.**AcceptPiece**, **TController**.**CancelPieceSelected**, **TController**.**Create**, **TController**.**EventOccursPieceSelectedSt**, **TController**.**EventOccursSquareSelectedSt**, **TController**.**MovePieceOverSquare**, **TController**.**RotateLeft**, **TController**.**RotateRight**, **TController**.**SelectPiece**, **TController**.**SetNewPickedObject**, **TController**.**TimeOutPieceSelectedSt**, **TController**.**TimeOutSquareSelectedSt**

[-] FGameEnded: boolean
Finalizó la partida
Used in
TController.**EventOccursSquareSelectedSt**, **TController**.**MoveHasFinished**, **TController**.**OnTimer**, **TController**.**SetNewPickedObject**

[-] FLastOpponentMove: **TMove**
Última Movida del Oponente <A.C. No Definida>
Used in
Create, **destroy**, **TController**.**DoOpponentMove**, **TController**.**EventOccursSquareSelectedSt**

[-] FNextStatus: **TStatus**
Estado al que pasará el controlador al completar el movimiento <A.C. No Definida>
Used in
TController.**MoveHasFinished**, **TController**.**SetNextStatus**

[-] FOpponentMove: **TMove**
Movida del Oponente <A.C. No Definida>
Used in
Create, **destroy**, **TController**.**DoOpponentMove**, **TController**.**EventOccursSquareSelectedSt**, **TController**.**InitGame**, **TController**.**SetNewPickedObject**

[-] FPickedObject: String

Nombre del último objeto válido indicado por el usuario <A.C.: Controlador :: Objeto Señalado>

Used in

TController.**AcceptPiece**, **TController**.**EventOccursBeforePieceSelectedSt**, **TController**.**EventOccursPieceSelectedSt**, **TController**.**EventOccursSquareSelectedSt**, **TController**.**MovePieceOverSquare**, **TController**.**SelectPiece**, **TController**.**SetNewPickedObject**

[-] FPossibleMove: **TMove**

Movida Posible <A.C. No Definida>

Used in

Create, **destroy**, **TController**.**AcceptPiece**, **TController**.**AssignPossibleMove**, **TController**.**AssignPossibleMove**, **TController**.**AssignPossibleMove**, **TController**.**CancelPieceSelected**, **TController**.**EventOccursSquareSelectedSt**, **TController**.**TimeOutBeforePieceSelectedSt**, **TController**.**TimeOutPieceSelectedSt**, **TController**.**TimeOutSquareSelectedSt**

Constructors:

[+] constructor Create

Hereda y crea las movidas correspondientes

Called by

TController.**Create**

Destructors:

[+] destructor destroy; **override**

Destruye las movidas y hereda

Overrides

destroy

AIII.2.4 Class TDynamicTimeout

TObject <-- **TDynamicTimeout**
Declared in Unit **Controller**

No known Subclasses

Used by Classes:

TStatus

Description:

Tiempo de Espera dinámico. <A.C.: No Definida>
Para cumplir con el requisito de la espera dinámica y adaptable a cada situación, es que se diseña esta clase que permite adaptar el tiempo de espera para ayudar al usuario de acuerdo a la cantidad de veces que recibió ayuda previamente. Se hace uso de una ecuación cuadrática cuyos factores son configurables. De esta manera se podrá adaptar a los requisitos del evaluador. El TimeOut se irá adaptando con los cambios del término independiente, el cual se incrementará a medida que el usuario demuestra que no hace uso de la ayuda y bajará cuando demuestra que la necesita. El valor del término independiente se persiste, de forma tal que el mismo seguirá valiendo para la próxima partida. Cada estado definido contará con su propio TDynamicTimeout con lo cual, los factores en cada caso podrá ajustarse individualmente.

No contiene el timer, por lo cual, simplemente actúa como gestor del valor a usar como TimeOut para ser usado por el objeto que lo creó.

Used in

TStatus.**Create**, **TStatus**

List of Constructors:

[+] constructor Create

Hereda e inicializa valores de FActivationTimes y FIndependetTerm desde el archivo de configuración.

List of Destructors:

[+] destructor Destroy

Guarda el último valor de FIndependentTerm en CfgFileName y Hereda

List of Methods:

[-] function getTimeOut

Calcula el TimeOut actual aplicando la fórmula

[-] function getTimeOuted

Indica si ocurrió el TimeOut en al menos una oportuniadad.

[+] procedure incActivationTimes

Incrementa en 1 (uno) la variable FActivationTimes

[+] procedure recalcIndependentTerm

Recalcula el término independiente.

Fields:

[-] CfgFileName: String

Archivo de configuración que contiene los términos de la ecuación almacenados

Used in

Create, **Destroy**

[-] FActivationTimes: integer

Cantidad de veces que se ejecutó el procedimiento IncActivationTimes después de haberse creado una instancia o ejecutado recalcTimeOutVar por última vez (antes de empezar una nueva jugada)

Used in

Create, **getTimeOuted**, **incActivationTimes**, **recalcIndependentTerm**

[-] FConstantCoef: double

Término constante de la ecuación

Used in

Create, **getTimeOut**, **TController**.**Create**

[-] FIndependentTerm: integer

Término independiente de la ecuación

Used in

Create, **Destroy**, **getTimeOut**, **recalcIndependentTerm**, **TController**.**Create**

[-] FLinearCoef: double

Término lineal de la ecuación

Used in

Create, **getTimeOut**, **TController**.**Create**

[-] FQuadraticCoef: double
Término cuadrático de la ecuación
Used in
Create, **getTimeOut**, **TController**.**Create**

[-] FStatusName: String
Nombre del Estado al que corresponde
Used in
Create, **Destroy**

Properties:

[+] [ro] property ActivationTimes: integer **read FActivationTimes**
Cantidad de veces que se ejecutó el procedimiento IncActivationTimes después de haberse creado una instancia o ejecutado recalcTimeOutVar por última vez (antes de empezar una nueva jugada)
[+] [ro] property ConstantCoef: double **read FConstantCoef**
Término constante de la ecuación
[+] [ro] property LinearCoef: double **read FLinearCoef**
Término lineal de la ecuación
[+] [ro] property QuadraticCoef: double **read FQuadraticCoef**
Término cuadrático de la ecuación
[+] [ro] property StatusName: String **read FStatusName**
Nombre del Estado al que corresponde
[+] [ro] property TimeOut: double **read getTimeOut**
Calcula el TimeOut actual aplicando la fórmula
Used in
TStatus.**DoEntry**

[+] [ro] property TimeOuted: boolean **read getTimeOuted**
Indica si ocurrió el TimeOut en al menos una oportuniadad.
Used in
recalcIndependentTerm

Constructors:

[+] constructor Create(aStatusName, aConfigFileName: String)
Hereda e inicializa valores de FActivationTimes y FIndependetTerm desde el archivo de configuración. Establece aConfigFileName y obtiene FConstantCoef, FLinearCoef, FQuadraticCoef, FActivationTimes del archivo de configuración (aConfigFileName)
Parameters
aStatusName
Nombre del Estado que lo creó
aConfigFileName
Archivo de configuración de donde se obtienen los valores de la ecuación característica y el último valor del término independiente
Called by
TStatus.**Create**

Destructors:

[+] destructor Destroy; **override**
Guarda el último valor de FIndependentTerm en CfgFileName y Hereda
Overrides
Destroy

Methods:

[-] function getTimeOut: double
Calcula el TimeOut actual aplicando la fórmula
Result
TimeOut calculado

[-] function getTimeOuted: boolean
Indica si ocurrió el TimeOut en al menos una oportuniadad.
Result
Devuelve True si "FActivationTimes > 0

[+] procedure incActivationTimes
Incrementa en 1 (uno) la variable FActivationTimes
Called by
TStatus.**DoTimeOut**

[+] procedure recalcIndependentTerm
Recalcula el término independiente. Si recibió ayuda, lo reduce, Sino, lo aumenta. Como mínimo valdrá cero.
Called by
TController.**AcceptPiece**

AIII.2.5 Class TMovementToDo

TObject <-- **TMovementToDo**
Declared in Unit **Controller**

No known Subclasses

Used by Classes:

TMovementToDoList

Description:

Movimiento a realizar: <A.C.: No definida> Simple clase auxiliar que se define por movimiento y una movida (TMove)

Used in
TMovementToDoList.**Add**, **TMovementToDoList**.**Get**, **TMovementToDoList**

Fields:

[-] FMove: **TMove**
Movida asociada al movimiento que permite saber sobre que objetos actuar
Used in
TController.**AcceptPiece**, **TController**.**CancelPieceSelected**,
TController.**DoNextMovement**, **TController**.**MovePieceOverSquare**,

TController.**RotateLeft**, **TController**.**RotateRight**, **TController**.**SelectPiece**, **TController**.**SetCompleteMovement**, **TController**.**SetNewPickedObject**

[-] FMovement: **TMovement**
Movimiento posible enumerado
Used in
TController.**AcceptPiece**, **TController**.**CancelPieceSelected**, **TController**.**DoNextMovement**, **TController**.**MovePieceOverSquare**, **TController**.**RotateLeft**, **TController**.**RotateRight**, **TController**.**SelectPiece**, **TController**.**SetCompleteMovement**

AIII.2.6 Class TMovementToDoList

TObject <-- ... <-- TObjectList <-- **TMovementToDoList**
Declared in Unit **Controller**

No known Subclasses

Used by Classes:

TController

Description:

Lista de Movimientos a realizar: <A.C.: No definida> Clase auxiliar que permite almacenar los sucesivos movimientos a realizar por una pieza y que luego serán invocados sucesivamente cada uno al terminar el anterior.
Used in
TController.**Create**, **TController**

List of Methods:

[+] function Add
Crea un objeto de la clase TMovementToDo y lo agrega al final de la lista
[+] procedure Delete
Borra el objeto y si no quedan elementos en la lista habilita el timer
[#] function Get
Devuelve el movimiento correspondiente
[#] procedure Put
Asigna un movimiento en una posición dada en la lista

Fields:

[-] FTimer: TTimer
Es la referencia al timer del Controlador que hay que detener en cada agregado de movimientos
Used in
Add, **Delete**, **TController**.**Create**

Properties:

[+] property Items [Index: integer]: **TMovementToDo read Get write Put**; **default**
Devuelve el movimiento correspondiente

Methods:

[+] function Add: **TMovementToDo**
Crea un objeto de la clase TMovementToDo y lo agrega al final de la lista
Result
Objeto creado
Called by
TController.**AcceptPiece**, **TController**.**CancelPieceSelected**, **TController**.**InitGame**, **TController**.**MovePieceOverSquare**, **TController**.**RotateLeft**, **TController**.**RotateRight**, **TController**.**SelectPiece**, **TController**.**SetCompleteMovement**

[+] procedure Delete(Index: integer); **override**
Borra el objeto y si no quedan elementos en la lista habilita el timer
Parameters
Index
Indice del movimiento en la lista
Overrides
Delete
Called by
TController.**MoveHasFinished**

[#] function Get(Index: integer): **TMovementToDo**
Devuelve el movimiento correspondiente
Parameters
Index
Número de orden del movimiento requerido
Result
Referencia al Objeto pedido

[#] procedure Put(Index: integer; aItem: **TMovementToDo**)
Asigna un movimiento en una posición dada en la lista
Parameters
Index
Número de orden del movimiento a asignar
aItem
Objeto de la clase TMovementToDo a asignar

AIII.2.7 Class TStatus

TObject <-- **TStatus**
Declared in Unit **Controller**

No known Subclasses

Used by Classes:

TCurrentInfo, **TController**

Description:

Estado. <A.C.: Controlador :: Estado Actual>
Define el Estado de una máquina de estados para dar soporte a los eventos:
- Al Entrar (OnEntry)
- Al Cumplirse el tiempo de TimeOut (OnTimeOut)
- Al ocurrir un evento del Usuario (OnEventOccurs).

Esta implementación es genérica y aplicable cualquier otro caso, al cual se podría agregar otro evento Al Salir no implementado para esta ocación.

Used in
TController.**Create**, **TController**, **TCurrentInfo**

List of Constructors:

[+] constructor Create
Crea el Objeto FDynamicTimeOut.

List of Destructors:

[+] destructor destroy
Destruye el Objeto FDynamicTimeOut y hereda

List of Methods:

[#] procedure DoEntry
Entrar en el estado.
[#] procedure DoEventOccurs
Activa el evento OnEventOccurs si fue asignado
[#] procedure DoTimeOut
Activa el evento OnTimeOut si fue asignado e incrementa las activaciones llamada invocando a IncActivationTimes del FDynamicTimeOut
[-] procedure SetActive
Activar estado.

Fields:

[-] FActive: boolean
Indica si es el estado actual
Used in
SetActive

[-] FDynamicTimeOut: **TDynamicTimeout**
Tiempo de Espera Dinámico
Used in
Create, **destroy**, **DoEntry**, **DoTimeOut**, **TController**.**AcceptPiece**, **TController**.**Create**

[-] FOnEntry: TNotifyEvent
Evento OnEntry
Used in
DoEntry, **SetActive**

[-] FOnEventOccurs: TNotifyEvent
Evento OnEventOccurs
Used in
DoEventOccurs

[-] FOnTimeOut: TNotifyEvent
Evento OnTimeOut
Used in
DoTimeOut

[-] FStatusName: String
Nombre asignado al Estado
Used in
Create, **DoEntry**, **DoEventOccurs**

[-] FTimer: TTimer
Referencia a un contador de tiempo del Sistema Operativo
Used in
DoEntry, **TController**.**Create**

Properties:

[+] property Active: boolean **read FActive write SetActive**
Indica si es el estado actual
Used in
TController.**SetCurrentStatus**

[+] property DynamicTimeOut: **TDynamicTimeout read FDynamicTimeOut write FDynamicTimeOut**
Tiempo de Espera Dinámico
[+] property OnEntry: TNotifyEvent **read FOnEntry write FOnEntry**
Evento OnEntry
Used in
TController.**Create**

[+] property OnEventOccurs: TNotifyEvent **read FOnEventOccurs write FOnEventOccurs**
Evento OnEventOccurs
Used in
TController.**Create**

[+] property OnTimeOut: TNotifyEvent **read FOnTimeOut write FOnTimeOut**
Evento OnTimeOut
Used in
TController.**Create**

[+] property StatusName: String **read FStatusName write FStatusName**
Nombre asignado al Estado
Used in
TController.**logCurrentStatusInfo**, **TMainForm**.**GLSceneViewerMouseDown**

Constructors:

[+] constructor Create(aStatusName, aConfigFileName: String)
Crea el Objeto FDynamicTimeOut. Hereda y asigna parámetros
Parameters
aStatusName
Nombre del Estado
aConfigFileName
Archivo de configuración que se pasa como parámetro de creación de FDynamicTimeOut
Called by
TController.**Create**

Destructors:

[+] destructor destroy; **override**
Destruye el Objeto FDynamicTimeOut y hereda
Overrides
destroy

Methods:

[#] procedure DoEntry; **dynamic**
Entrar en el estado. Establece el intervalo de espera al timer y activa el evento OnEntry si fue asignado
Called by
SetActive

[#] procedure DoEventOccurs; **dynamic**
Activa el evento OnEventOccurs si fue asignado
Called by
TController.**SetNewPickedObject**

[#] procedure DoTimeOut; **dynamic**
Activa el evento OnTimeOut si fue asignado e incrementa las activaciones llamada invocando a IncActivationTimes del FDynamicTimeOut
Called by
TController.**OnTimer**

[-] procedure SetActive(**const** Value: boolean)
Activar estado. Usado al establecer la propiedad Active de forma tal que si el estado está activo se dispara el evento OnEntry
Parameters
Value
Nuevo valor para asignar a FActive

AIII.3 Unit FMain

Used Units:

Used in interface: System, SysInit, Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, GLScene, GLObjects, GLWin32Viewer, GLMisc, glbehaviours, GLCadencer, glthorfx, GLVectorFileObjects, glfile3ds, StdCtrls, math, GLGeomObjects, gltexture, ExtCtrls, jpeg, GLMultiPolygon, GLExtrusion, GLPolyhedron, glfilemd3, GLWin32FullScreenViewer, GLSpaceText, GLSound, **Controller**, **View**, GLFireFX, GLParticleFX, GLPerlinPFX, GLFileMD2, **Common**, GLSkyBox, GLGui, GLWindows
Used in implementation: glutils

Description:

Esta unidad permite definir el entorno visual conteniendo todos los componenetes de la librería GLScen básicos para formar el tablero visual y poder incorporar sus correspondientes piezas, casillero, bordes e indicadores de rotación.

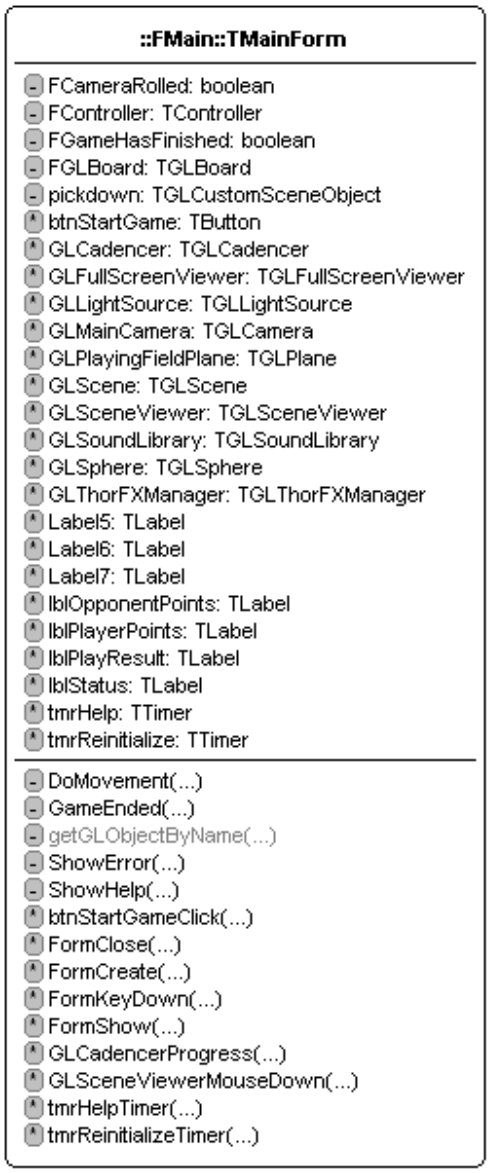


Ilustración Anexo III - 2 - Diagrama de Clases de FMain

List of Variables:

Global Variables: MainForm
File-Local Variables: *None*

List of Classes:

Global Classes: TMainForm
File-Local Classes: *None*

Variables:

[ui] MainForm: **TMainForm**

AIII.3.1 Class TMainForm

TObject <-- ... <-- TForm <-- **TMainForm**
Declared in Unit **FMain**

No known Subclasses

Not used by any known Class

Description:

List of Methods:

[[^] procedure btnStartGameClick
Procedimiento que captura el evento OnClick de btnStartGame.
[-] procedure DoMovement
Procedimiento que captura al evento OnMovementToDo de FController.
[[^] procedure FormClose
Procedimiento que captura al evento "al cerrarse" del formulario principal.
[[^] procedure FormCreate
Procedimiento que captura el evento crear formulario principal.
[[^] procedure FormKeyDown
Procedimiento que captura al evento OnKeyDown.
[[^] procedure FormShow
Procedimiento que captura el evento OnShow del formulario principal.
[-] procedure GameEnded
Procedimiento que captura al evento OnGameEnded de FController.
[-] function getGLObjectByName
Función auxiliar que permite obtener el objeto visual a partir de su nombre.
[[^] procedure GLCadencerProgress
Procedimiento que captura el evento OnProgress de GLCadencer.
[[^] procedure GLSceneViewerMouseDown
Procedimiento que captura el evento OnMouseDown de GLCadencer.
[-] procedure ShowError
Procedimiento que captura al evento OnError de FController.
[-] procedure ShowHelp
Procedimiento que captura al evento OnHelpNeeded de FController.
[[^] procedure tmrHelpTimer
Timer de tiempo de ayuda.
[[^] procedure tmrReinitializeTimer
Procedimiento que captura el evento OnTimer del Timer de Reinicialización del juego.

Fields:

[[^] btnStartGame: TButton
botón de inicio del juego
Used in
FormShow, **GameEnded**, **tmrReinitializeTimer**

[-] FCameraRolled: boolean
True, indica que la cámara fue rotada por tener que jugar el usuario con piezas negras
Used in
btnStartGameClick

[-] FController: **TController**
Private declarations Controlador de la partida
Used in
btnStartGameClick, **FormClose**, **FormKeyDown**, **GLCadencerProgress**, **GLSceneViewerMouseDown**

[-] FGameHasFinished: boolean
True, indica que la partida ha finalizado.
Used in
btnStartGameClick, **GameEnded**, **GLCadencerProgress**

[-] FGLBoard: **TGLBoard**
Tablero Visual
Used in
btnStartGameClick, **FormClose**

[[^] GLButton: TGLDodecahedron
Timer para reiniciar una nueva partida
Used in
FormShow, **GameEnded**, **getGLObjectByName**

[[^] GLCadencer: TGLCadencer
Administrador de Frames de la escena
Used in
FormClose

[[^] GLFullScreenViewer: TGLFullScreenViewer
Visualizador a pantalla completa
[[^] GLLightSource: TGLLightSource
Fuente de Luz
[[^] GLMainCamera: TGLCamera
Camara Principal
Used in
btnStartGameClick

[[^] GLPlayingFieldPlane: TGLPlane
Used in
btnStartGameClick, **getGLObjectByName**, **GLCadencerProgress**, **ShowHelp**

[[^] GLScene: TGLScene
Escena principal
[[^] GLSceneViewer: TGLSceneViewer
Visualizador de la escena principal
Used in
GLSceneViewerMouseDown

[[^] GLSoundLibrary: TGLSoundLibrary
Librería de sonidos
Used in
FormCreate, **GameEnded**, **ShowError**, **ShowHelp**

[[^] GLSphere: TGLSphere
Esfera origen de la ayuda
Used in
getGLObjectByName, **ShowHelp**

[[^] GLThorFXManager: TGLThorFXManager
Indicador luminoso
Used in
ShowHelp, **tmrHelpTimer**

[[^] Label1: TLabel

[[^] Label5: TLabel

[[^] Label6: TLabel

[[^] Label7: TLabel

[[^] lblOpponentPoints: TLabel
Used in
GLSceneViewerMouseDown

[[^] lblPlayerPoints: TLabel
Used in
GLSceneViewerMouseDown

[[^] lblPlayResult: TLabel

[[^] lblStatus: TLabel
Used in
btnStartGameClick, **GLSceneViewerMouseDown**

[-] pickdown: TGLCustomSceneObject
Referencia al último objeto visual indicado con el botón izquierdo del mouse
Used in
GLSceneViewerMouseDown

[[^] tmrHelp: TTimer
Timer para finalizar la ayuda con el indicador luminoso
Used in
ShowHelp, **tmrHelpTimer**

[[^] tmrReinitialize: TTimer
Used in
GameEnded, **tmrReinitializeTimer**

Methods:

[[^] procedure btnStartGameClick(Sender: TObject)
Procedimiento que captura el evento OnClick de btnStartGame. Crea a FController y a FGLBoard.
Restaura la orientación de la cámara si está rotada.

Establece la posición de la cámara sobre el centro del tablero.
Asigna a los cuatro eventos que publica FController (OnGameEnded, OnError, OnHelpNeeded, OnMovementToDo) los respectivos procedures implementados (GameEnded, ShowError, ShowHelp, DoMovement).
Invoca la incialización de FController (con InitGame).
Finalmente, si el color de piezas asignado al usuario es negro, rota el tablero 180º.
Parameters
Sender
Objeto que dispara el evento

[-] procedure DoMovement(Sender: TObject; **const** aPieceName, aSquareName: String; **const** aMovementType: **TMovement**)
Procedimiento que captura al evento OnMovementToDo de FController. Se obtiene la pieza visual a partir del nombre con getGLObjectByName, se indica a la pieza visual que debe realizar el movimiento indicado en el parámetro aMovementType, diferenciando el caso en que venga el parámetro aSquareName, en el cual también se lo incluirá en dicha invocación del procedimiento doMovement de la pieza con comportamiento polimórfico.
Parameters
Sender
Objeto que dispara el evento
aPieceName
Nombre de la Pieza Visual que debe realizar un movimiento
aSquareName
Nombre del Casillero Visual involucrado (solo si tiene un nombre de casillero válido.
aMovementType
Tipo de movimiento a realizar
Called by
btnStartGameClick

[[^] procedure FormClose(Sender: TObject; **var** Action: TCloseAction)
Procedimiento que captura al evento "al cerrarse" del formulario principal. Se deshabilita a GLCadencer, y se destruye a FGLBoard y a FController (si están asignados).
Parameters
Sender
Objeto que dispara el evento
Action
Indica la acción que debe realizar el formulario

[[^] procedure FormCreate(Sender: TObject)
Procedimiento que captura el evento crear formulario principal. Se cargan los sonidos para estar disponibles en cualquier momento y su ejecución no tenga demoras.
Parameters
Sender
Objeto que dispara el evento

[[^] procedure FormKeyDown(Sender: TObject; **var** Key: Word; Shift: TShiftState)
Procedimiento que captura al evento OnKeyDown. Se utiliza para reconocer si el investigador realizó la combinación de teclas [Ctrl + T]. Si fue así, llamará al procedimiento GetTrace de FController.
Parameters
Sender
Objeto que dispara el evento
Key

Tecla invocada
Shift
Estado de la tecla que permitirá saber si está presionada la tecla Ctrl

[[^] procedure FormShow(Sender: TObject)
Procedimiento que captura el evento OnShow del formulario principal. Se pone invisible el botón de inicio de partida y se invoca la misma.
Parameters
Sender
Objeto que dispara el evento

[-] procedure GameEnded(Sender: TObject)
Procedimiento que captura al evento OnGameEnded de FController. Se invoca el sonido y la escena correspondiente al festejo final Se habilita el botón de inicio y se habilita el timer de reinicialización.
Parameters
Sender
Objeto que dispara el evento
Called by
btnStartGameClick

[-] function getGLObjectByName(aObjectName: String): **TGLBasicForm**
Función auxiliar que permite obtener el objeto visual a partir de su nombre. Es usada para poder realizar el mensaje enviado por FController en forma de evento para así obtener directamente el objeto a referenciar.
Parameters
aObjectName
Nombre del objeto visual
Result
Objeto Visual requerido
Called by
DoMovement, **ShowError**, **ShowHelp**

[[^] procedure GLCadencerProgress(Sender: TObject; **const** deltaTime, newTime: Double)
Procedimiento que captura el evento OnProgress de GLCadencer. Recorre todos los componentes hijos del plano del juego y le da la oportunidad de realizar acciones (DoActions) indicándole el tiempo que transcurrió desde el último frame. Asimismo, si el objeto visual finalizó su movimimiento indicado, se lo comunica al controlador a través de MoveHasFinished.
Parameters
Sender
Objeto que dispara el evento
deltaTime
Tiempo transcurrido desde el frame anterior
newTime
Tiempo correspondiente al evento

[[^] procedure GLSceneViewerMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer)
Procedimiento que captura el evento OnMouseDown de GLCadencer. Se aplica el método sugerido en demostraciones de GLScene para identificar los objetos señalados con el mouse, utilizando GLSceneViewer.Buffer.GetPickedObject(x, y), asignándoselo a PickedObject para que se envíe el mensaje a FController.

Parameters
Sender
Objeto que dispara el evento
Button
Botón del mouse presionado
Shift
Estado de la tecla Shift
X
Coordenada X respecto de la pantalla (no de la escena)
Y
Coordenada Y respecto de la pantalla (no de la escena)

[-] procedure ShowError(Sender: TObject; **const** aObjectName: String)
Procedimiento que captura al evento OnError de FController. Se obtiene el objeto visual a partir del nombre con getGLObjectByName, se indica al objeto visual que debe mostrarse en error con ShowError con parámetro el tiempo definido para ello. Se activa el sonido de error.
Parameters
Sender
Objeto que dispara el evento
aObjectName
Nombre del Objeto Visual que debe mostrar el error
Called by
btnStartGameClick

[-] procedure ShowHelp(Sender: TObject; **const** aObjectName: String)
Procedimiento que captura al evento OnHelpNeeded de FController. Se obtiene el objeto visual a partir del nombre con getGLObjectByName, se indica al objeto visual que debe mostrarse en ayuda con ShowHelp con parámetro el tiempo definido para ello. Se habilita el efecto luminoso que señala el objeto a ayudar. Se activa el sonido de ayuda.
Parameters
Sender
Objeto que dispara el evento
aObjectName
Nombre del Objeto Visual que debe mostrar el error
Called by
btnStartGameClick

[[^] procedure tmrHelpTimer(Sender: TObject)
Timer de tiempo de ayuda. En el momento de hacer el ShowHelp se activa y acá se desactiva. El señalador de ayuda permanecerá "encendido" el tiempo indicado en la propiedad Interval, es decir, acá se "apaga".
Parameters
Sender
Objeto que dispara el evento

[[^] procedure tmrReinitializeTimer(Sender: TObject)
Procedimiento que captura el evento OnTimer del Timer de Reinicialización del juego. Desabilita el Timer e invoca el procedimiento btnStartGame.Click.
Parameters
Sender
Objeto que dispara el evento

AIII.4 Unit LogToFile

Used Units:

Used in interface: System, SysInit, classes, sysUtils
Used in implementation: *None*

Used by:
In Interface of Units: Controller

Description:

Contendrá la clase que se encargará de almacenar la información necesaria en cada momento en el archivo de trazado permitiendo así el registro cronológico de la partida.

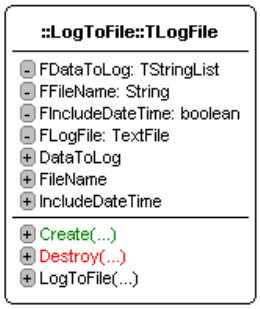


Ilustración Anexo III – 3 - Diagrama de Clases de LogToFile

List of Classes:

Global Classes: TLogFile
File-Local Classes: *None*

List of Functions:

Global Functions:
[ui] function NextName
Función auxiliar para nombrar al archivo de resguardo.
[ui] function NowInFormat
Función que da formato a la fecha-hora
File-Local Functions: *None*

Functions:

[ui] function NextName(aFileName: String): String
Función auxiliar para nombrar al archivo de resguardo.
Parameters
aFileName
Nombre del archivo a guardar
Result
Nuevo nombre que tomará el archivo de resguardo (número de cinco cifras seguido de guión y del nombre del archivo original)
Called by
TLogFile.Create

[ui] function NowInFormat: String
Función que da formato a la fecha-hora
Result
Fecha-Hora en formato yyyy-MM-dd hh:mm:ss
Called by
TLogFile.**LogToFile**

AIII.4.1 Class TLogFile

TObject <-- **TLogFile**
Declared in Unit **LogToFile**

No known Subclasses

Used by Classes:

TController

Description:

Clase que se encarga de almacenar la información en el archivo de trazado <A.C.: No Definida>
Used in
TController.**Create**, **TController**

List of Constructors:

[+] constructor Create
Crea y abre el archivo del trazado.

List of Destructors:

[+] destructor Destroy
Libera la lista, cierra el archivo y hereda el destroy

List of Methods:

[+] procedure LogToFile
Guarda el registro que se encuentra en la lista en formato CSV (Comma Separated Values).

Fields:

[-] FDataToLog: TStringList
Lista de datos a registrar
Used in
Create, **Destroy**, **LogToFile**

[-] FFileName: String
Nombre del Archivo del trazado

[-] FIncludeDateTime: boolean
¿ Incluye Fecha y Hora cada registro ?
Used in
Create

[-] FLogFile: TextFile
Archivo de texto
Used in
Create, **Destroy**, **LogToFile**

Properties:

[+] property DataToLog: TStringList **read FDataToLog write FDataToLog**
Lista de datos a registrar
Used in
TController.**logInfo**

[+] [ro] property FileName: String **read FFileName**
Nombre del Archivo del trazado
[+] property IncludeDateTime: boolean **read FIncludeDateTime write FIncludeDateTime**
¿ Incluye Fecha y Hora cada registro ?

Constructors:

[+] constructor Create(aFileName: String)
Crea y abre el archivo del trazado. Si el archivo existe, lo resguarda. Crea la lista de campos de registros a guardar.
Parameters
aFileName
Path completo y nombre del archivo a crear
Called by
TController.**Create**

Destructors:

[+] destructor Destroy; **override**
Libera la lista, cierra el archivo y hereda el destroy
Overrides
Destroy

Methods:

[+] procedure LogToFile
Guarda el registro que se encuentra en la lista en formato CSV (Comma Separated Values). Sin embargo, la extensión del archivo puede ser cualquiera
Called by
TController.**logInfo**

AIII.5 Unit Model

Used Units:

Used in interface: System, SysInit, uObjectlist, **Common**, Classes
Used in implementation: types, dialogs, sysutils, typinfo, Clipbrd

Used by:
In Interface of Units: Controller

Description:

Esta Unidad contiene todas las clases correspondientes al SubModelo de Diseño Modelo. Aquí se definen las clases del juego y sus funcionalidades, desde las reglas hasta la obtención de jugadas posibles. Desde el punto de vista del modelo, no importa si el usuario tiene asignadas piezas de un color u otro. Tampoco reconocer situaciones de juego, como por ejemplo quién juega

primero o quién debe realizar la próxima movida. Tanto de uno como de otro se encargará el controlador.

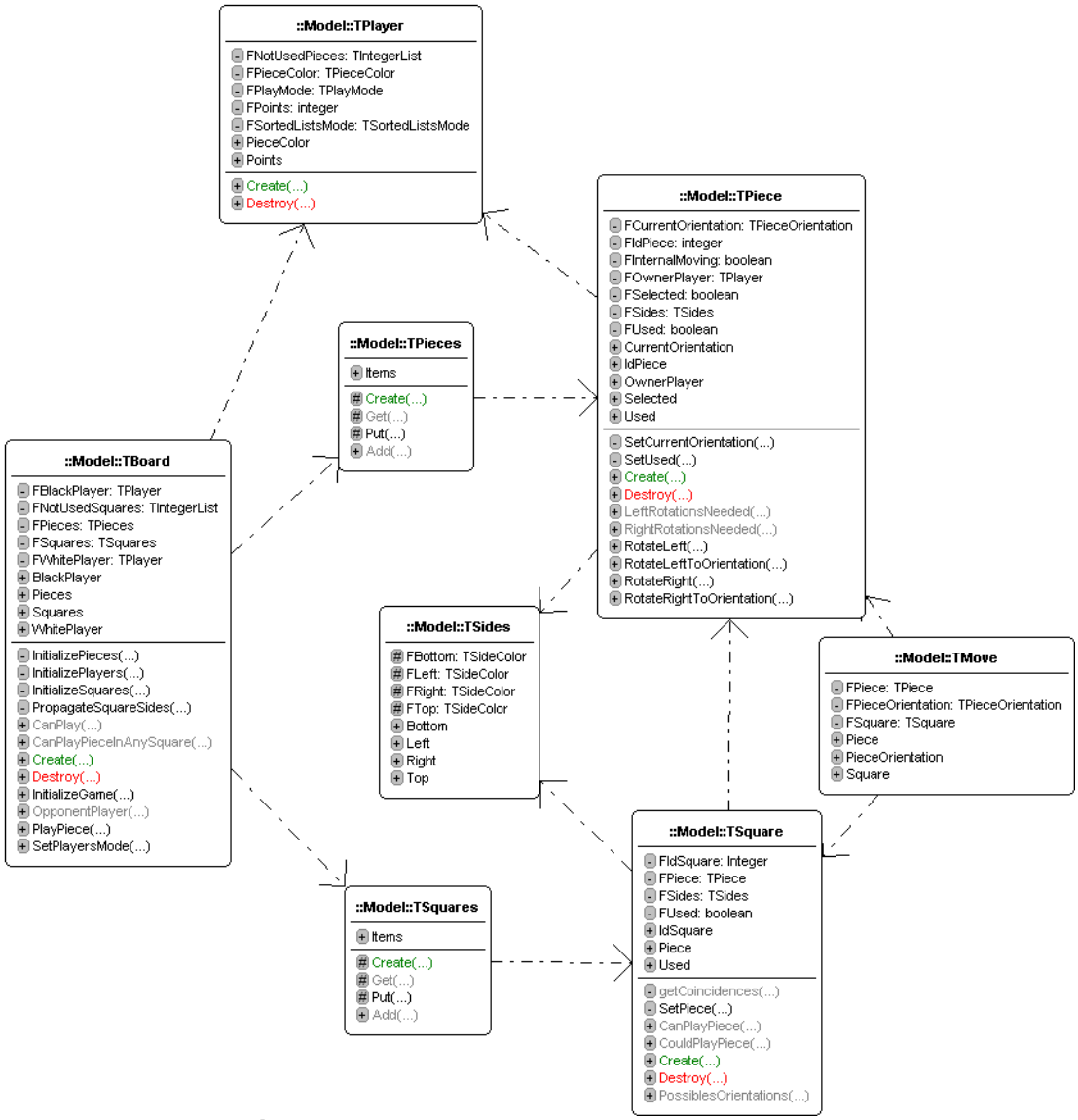


Ilustración Anexo III - 4 - Diagrama de Clases de Model

List of Constants:

Global Constants: ppstInitial, ppstRotatedRightOnce, ppstRotatedRightThreeTimes, ppstRotatedRightTwice, slmRandom, slmSequential, spmBestForBoth, spmBestForOpponent, spmBestForPlayer, spmLessEffort
File-Local Constants: *None*

List of Simple Types:

Global Simple Types: TPieceOrientation, TPlayMode, TSortedListsMode
File-Local Simple Types: *None*

List of Classes:

Global Classes: TBoard, **TMove**, **TPiece**, **TPieces**, **TPlayer**, **TSides**, **TSquare**, **TSquares**
File-Local Classes: *None*

Constants:

[ui] ppstInitial = **TPieceOrientation**(0)
Posición Inicial
Used in
TPiece.**Create**, **TPiece**.**RotateLeft**, **TPiece**.**RotateRight**
[ui] ppstRotatedRightOnce = **TPieceOrientation**(1)
Rotada una vez hacia la Derecha
Used in
TPiece.**RotateLeft**, **TPiece**.**RotateRight**
[ui] ppstRotatedRightThreeTimes = **TPieceOrientation**(3)
Rotada tres veces hacia la Derecha
Used in
TPiece.**RotateLeft**, **TPiece**.**RotateRight**
[ui] ppstRotatedRightTwice = **TPieceOrientation**(2)
Rotada dos veces hacia la Derecha
Used in
TPiece.**RotateLeft**, **TPiece**.**RotateRight**
[ui] slmRandom = **TSortedListsMode**(1)
Ordenada aleatóreamente
Used in
TBoard.**CanPlay**, **TController**.**Create**
[ui] slmSequential = **TSortedListsMode**(0)
Ordenada por el identificador único
Used in
TController.**Create**, **TPlayer**.**Create**
[ui] spmBestForBoth = **TPlayMode**(2)
Mejor jugada para ambos Jugadores (previsto)
[ui] spmBestForOpponent = **TPlayMode**(3)
Mejor jugada para el oponente (previsto)
Used in
TController.**Create**
[ui] spmBestForPlayer = **TPlayMode**(1)
Mejor jugada para el Jugador
Used in
TBoard.**CanPlay**
[ui] spmLessEffort = **TPlayMode**(0)
Implica el menor esufuerzo de cálculo
Used in
TBoard.**CanPlay**, **TController**.**Create**, **TPlayer**.**Create**

Simple Types:

[ui] TPieceOrientation = (**ppstInitial**, **ppstRotatedRightOnce**, **ppstRotatedRightTwice**, **ppstRotatedRightThreeTimes**)
Orientación de la Pieza.
Enumeración de las posibles Orientaciones que puede tener una Pieza
Used in
TBoard.**CanPlay**, **TBoard**.**CanPlayPieceInAnySquare**, **TController**.**DoOpponentMove**, **TController**.**EventOccursSquareSelectedSt**, **TSquare**.**CouldPlayPiece**

[ui] TPlayMode = (**spmLessEffort**, **spmBestForPlayer**, **spmBestForBoth**, **spmBestForOpponent**)

Modo de Juego

Enumeración de los posibles Modos de Juego que se asignará al usuario para ayudarlo y al oponente para realizar su jugada

Used in

TController.**Create**

[ui] TSortedListsMode = (**slmSequential**, **slmRandom**)

Modo de ordenamiento

Enumeración de los posibles Modos de Ordenamiento de la listas de piezas sin usar y de casilleros sin usar

Used in

TController.**Create**

AIII.5.1 Class TBoard

TObject <-- **TBoard**
Declared in Unit **Model**

No known Subclasses

Used by Classes:

TController

Description:

Tablero <A.C.: Tablero>
Clase que representa el modelo del tablero de juego que inicializa los componentes abstractos del tablero: jugadores, casilleros y piezas. Impide la realización de jugadas erróneas y asimismo "piensa" movidas.

Used in

TController.**Create**, **TController**

List of Constructors:

[+] constructor Create

Hereda y crea a FWhitePlayer, FBlackPlayer, FPieces, FSquares y FNotUsedSquares.

List of Destructors:

[+] destructor Destroy

Destruye a a FWhitePlayer, FBlackPlayer, FPieces, FSquares y FNotUsedSquares y Hereda

List of Methods:

[+] function CanPlay

¿Puede Jugar alguna pieza en algún casillero? Crea las listas de casilleros y piezas del jugador libres, las reordena si es necesario y las recorre y evalúa según el modo de juego asignado.

[+] function CanPlayPieceInAnySquare

¿Puede Jugar esta pieza en algún casillero? En este caso la búsqueda se reduce a la pieza dada.

[+] procedure InitializeGame

Inicializa la partida <A.C.

[-] procedure InitializePieces

Inicializar Piezas <A.C.: Tablero :: Inicializar Piezas> Establece los colores de los bordes de cada pieza obteniéndolo de la matriz PiecesSideColors

[-] procedure InitializePlayers
Inicilizar Jugadores <A.C.: Tablero :: Inicializar Jugadores> Asigna las piezas al jugador correspondiente y las agrega a su propia lista de piezas sin utilizar

[-] procedure InitializeSquares
Inicializar Casilleros <A.C.: Tablero :: Inicializar Casilleros> Asigna los colores a cada lado que obtiene de la matriz InitialSquaresSideColor.

[+] function OpponentPlayer
Función auxiliar que devuelve el jugador blanco si el parámetro es negro y viceversa

[+] procedure PlayPiece
Acepta la pieza indicada en el casillero inidicado.

[-] procedure PropagateSquareSides
Propagar Lados <A.C.: Tablero :: No Definido> Realiza la propagación de lados asignados una vez que se ha aceptado la pieza asignando a los casilleros contiguos los colores de lado que impone la pieza colocada

[+] procedure SetPlayersMode
Establece modo de juego de los jugadores y modo de ordenamiento de las listas.

Fields:

[-] FBlackPlayer: **TPlayer**
Jugador de Piezas Negras <A.C.: Tablero :: Jugador 2>
Used in
Create, **Destroy**, **InitializePlayers**, **OpponentPlayer**, **SetPlayersMode**

[-] FNotUsedSquares: TIntegerList
Lista de Identificadores de Casilleros no Utilizados <A.C.: No Definido>
Used in
CanPlay, **CanPlayPieceInAnySquare**, **Create**, **Destroy**, **InitializeSquares**, **PlayPiece**

[-] FPieces: **TPieces**
Lista de Piezas del Juego <A.C.: Tablero :: Piezas>
Used in
CanPlay, **Create**, **Destroy**, **InitializePieces**, **InitializePlayers**

[-] FSquares: **TSquares**
Lista de Casilleros del Juego <A.C.: Tablero :: Casilleros>
Used in
CanPlay, **CanPlayPieceInAnySquare**, **Create**, **Destroy**, **InitializeSquares**, **PropagateSquareSides**

[-] FWhitePlayer: **TPlayer**
Jugador de Piezas Blancas <A.C.: Tablero :: Jugador 1>
Used in
Create, **Destroy**, **InitializePlayers**, **OpponentPlayer**, **SetPlayersMode**

Properties:

[+] [ro] property BlackPlayer: **TPlayer read FBlackPlayer**
Jugador de Piezas Negras <A.C.: Tablero :: Jugador 2>
Used in
TController.**InitGame**

[+] [ro] property Pieces: **TPieces read FPieces**
Lista de Piezas del Juego <A.C.: Tablero :: Piezas>
Used in
TController.**AssignPossibleMove**, **TController**.**EventOccursPieceSelectedSt**, **TController**.**EventOccursSquareSelectedSt**, **TController**.**InitGame**, **TController**.**SelectPiece**, **TController**.**SetNewPickedObject**

[+] [ro] property Squares: **TSquares read FSquares**
Lista de Casilleros del Juego <A.C.: Tablero :: Casilleros>
Used in
TController.**AssignPossibleMove**, **TController**.**AssignPossibleMove**, **TController**.**EventOccursSquareSelectedSt**, **TController**.**InitGame**, **TController**.**MovePieceOverSquare**, **TController**.**SetNewPickedObject**

[+] [ro] property WhitePlayer: **TPlayer read FWhitePlayer**
Jugador de Piezas Blancas <A.C.: Tablero :: Jugador 1>
Used in
TController.**InitGame**

Constructors:

[+] constructor Create
Hereda y crea a FWhitePlayer, FBlackPlayer, FPieces, FSquares y FNotUsedSquares. Asigna a cada jugador su respectivo color de piezas (pccWhite a FWhitePlayer y pccBlack a FBlackPlayer
Called by
TController.**Create**

Destructors:

[+] destructor Destroy; **override**
Destruye a a FWhitePlayer, FBlackPlayer, FPieces, FSquares y FNotUsedSquares y Hereda
Overrides
Destroy

Methods:

[+] function CanPlay(aPlayer: **TPlayer**; **var** aPieceId: integer; **var** aPieceOrientation: **TPieceOrientation**; **var** aSquareId: Integer): boolean
¿Puede Jugar alguna pieza en algún casillero? Crea las listas de casilleros y piezas del jugador libres, las reordena si es necesario y las recorre y evalúa según el modo de juego asignado.
Parameters
aPlayer
Jugador sobre el que se va a evaluar la posibilidad de juga4r
aPieceId
Parámetro de salida que devuelve el identificador de la pieza a jugar, si es posible hacerlo
aPieceOrientation
Orientación que debe tomar la pieza para poder jugarse
aSquareId
Parámetro de salida que devuelve el identificador del casillero donde jugar la pieza, si es posible hacerlo

Result
True si es posible que el jugador realice alguna jugada
Called by
TController.**AssignPossibleMove**, **TController**.**EventOccursSquareSelectedSt**, **TController**.**InitGame**

[+] function CanPlayPieceInAnySquare(aPlayer: **TPlayer**; aPiece: **TPiece**; **var** aPieceOrientation: **TPieceOrientation**; **var** aSquareId: Integer): boolean
¿Puede Jugar esta pieza en algún casillero? En este caso la búsqueda se reduce a la pieza dada.
Parameters
aPlayer
Jugador sobre el que se va a evaluar la posibilidad de jugar
aPiece
Referencia a la pieza a jugar en algún casillero, si es posible hacerlo
aPieceOrientation
Orientación que debe tomar la pieza para poder jugarse
aSquareId
Parámetro de salida que devuelve el identificador del casillero donde jugar la pieza, si es posible hacerlo
Result
True si es posible que el jugador realice alguna jugada con la pieza elegida
Called by
TController.**AssignPossibleMove**

[+] procedure InitializeGame
Inicializa la partida <A.C. Inicializar Tablero> Invoca sucesivamente a InitializePlayers, a InitializeSquares y a InitializePieces
Called by
TController.**InitGame**

[-] procedure InitializePieces
Inicializar Piezas <A.C.: Tablero :: Inicializar Piezas> Establece los colores de los bordes de cada pieza obteniéndolo de la matriz PiecesSideColors
Called by
InitializeGame

[-] procedure InitializePlayers
Inicilizar Jugadores <A.C.: Tablero :: Inicializar Jugadores> Asigna las piezas al jugador correspondiente y las agrega a su propia lista de piezas sin utilizar
Called by
InitializeGame

[-] procedure InitializeSquares
Inicializar Casilleros <A.C.: Tablero :: Inicializar Casilleros> Asigna los colores a cada lado que obtiene de la matriz InitialSquaresSideColor. Además agrega el idSquare a la lista de casilleros no utilizados FNotUsedSquares
Called by
InitializeGame

[+] function OpponentPlayer(aPlayer: **TPlayer**): **TPlayer**
Función auxiliar que devuelve el jugador blanco si el parámetro es negro y viceversa
Parameters

aPlayer
Jugador del cual quiere conocerse el oponente
Result
Oponente de aPlayer
Called by
TController.**getOpponentPlayer**

[+] procedure PlayPiece(aPiece: **TPiece**; aSquare: **TSquare**)
Acepta la pieza indicada en el casillero inidicado. Se invoca SetPiece del casillero. Se incrementan los puntos del jugador. Se propagan los lados del casillero hacia los de sus vecinos. Se borra el casillero de la lista de casilleros no utilizados.
Parameters
aPiece
Pieza a aceptar
aSquare
Casillero a aceptar
Called by
TController.**AcceptPiece**, **TController**.**SetCompleteMovement**

[-] procedure PropagateSquareSides(aSquare: **TSquare**)
Propagar Lados <A.C.: Tablero :: No Definido> Realiza la propagación de lados asignados una vez que se ha aceptado la pieza asignando a los casilleros contiguos los colores de lado que impone la pieza colocada
Parameters
aSquare
Casillero cuyos lados se propagan a los limítrofes
Called by
PlayPiece

[+] procedure SetPlayersMode(aWhitePlayerMode, aBlackPlayerMode: **TPlayMode**; aSortedListsMode: **TSortedListsMode**)
Establece modo de juego de los jugadores y modo de ordenamiento de las listas.
Parameters
aWhitePlayerMode
Modo de juego del jugador de piezas blancas
aBlackPlayerMode
Modo de juego del jugador de piezas Negras
aSortedListsMode
Modo de ordenamiento de las listas de casilleros disponibles y piezas sin usar.
Called by
TController.**Create**

AIII.5.2 Class TMove

TObject <-- **TMove**
Declared in Unit **Model**

No known Subclasses

Used by Classes:

TMovementToDo, **TCurrentInfo**

Description:

Movida <A.C.: No Definida>
Clase que permite agrupar en una sola entidad la referencia a una movida completa.
Used in
TController.**Create**, **TCurrentInfo**.**Create**, **TCurrentInfo**, **TMovementToDo**

Fields:

[-] FPiece: **TPiece**
Referencia a la Pieza
[-] FPieceOrientation: **TPieceOrientation**
Referencia a la Orientación en que quedará la pieza una vez realizada la movida
[-] FSquare: **TSquare**
Referencia al Casillero de la movida

Properties:

[+] property Piece: **TPiece read FPiece write FPiece**
Referencia a la Pieza
Used in
TController.**AcceptPiece**, **TController**.**AssignPossibleMove**,
TController.**AssignPossibleMove**, **TController**.**AssignPossibleMove**,
TController.**CancelPieceSelected**, **TController**.**DoNextMovement**,
TController.**DoOpponentMove**, **TController**.**EventOccursPieceSelectedSt**,
TController.**EventOccursSquareSelectedSt**, **TController**.**InitGame**,
TController.**SelectPiece**, **TController**.**SetCompleteMovement**,
TController.**TimeOutBeforePieceSelectedSt**,
TController.**TimeOutPieceSelectedSt**, **TController**.**TimeOutSquareSelectedSt**

[+] property PieceOrientation: **TPieceOrientation read FPieceOrientation write FPieceOrientation**
Referencia a la Orientación en que quedará la pieza una vez realizada la movida
Used in
TController.**AssignPossibleMove**, **TController**.**AssignPossibleMove**,
TController.**AssignPossibleMove**, **TController**.**EventOccursSquareSelectedSt**,
TController.**InitGame**, **TController**.**SetCompleteMovement**,
TController.**TimeOutSquareSelectedSt**

[+] property Square: **TSquare read FSquare write FSquare**
Referencia al Casillero de la movida
Used in
TController.**AcceptPiece**, **TController**.**AssignPossibleMove**,
TController.**AssignPossibleMove**, **TController**.**AssignPossibleMove**,
TController.**DoNextMovement**, **TController**.**DoOpponentMove**,
TController.**EventOccursSquareSelectedSt**, **TController**.**InitGame**,
TController.**MovePieceOverSquare**, **TController**.**SetCompleteMovement**,
TController.**TimeOutPieceSelectedSt**, **TController**.**TimeOutSquareSelectedSt**

AIII.5.3 Class TPiece

TObject <-- **TPiece**
Declared in Unit **Model**

No known Subclasses

Used by Classes:

TPieces, **TSquare**, **TMove**

Description:

Pieza <A.C.: Pieza>
Es una abstracción de la pieza que el usuario puede mover.
Used in
TPieces.**Add**, **TPieces**.**Get**, **TMove**, **TPieces**, **TSquare**

List of Constructors:

[+] constructor Create
Hereda y crea FSides.

List of Destructors:

[+] destructor Destroy
Destruye FSides y hereda

List of Methods:

[+] function LeftRotationsNeeded
Rotaciones hacia la izquierda necesarias para llegar a la orientación de la pieza buscada
[+] function RightRotationsNeeded
Rotaciones hacia la derecha necesarias para llegar a la orientación de la pieza buscada
[+] procedure RotateLeft
Rotar hacia la Izquierda <A.C.: Pieza :: Rotar hacia la Izquierda> Rota la pieza 90º en sentido antihorario.
[+] procedure RotateLeftToOrientation
Establece la orientación rotando la pieza hacia la izquierda
[+] procedure RotateRight
Rotar hacia la Derecha <A.C.: Pieza :: Rotar hacia la Derecha> Rota la pieza 90º en sentido horario.
[+] procedure RotateRightToOrientation
Establece la orientación rotando la pieza por el camino más corto
[-] procedure SetCurrentOrientation
Establece la orientación buscada rotando la pieza hacia la derecha
[-] procedure SetUsed
Establece a la pieza como usada y borra de la lista de piezas disponibles del jugador dueño de la pieza

Fields:

[-] FCurrentOrientation: **TPieceOrientation**
Orientación actual de la pieza <A.C.: Pieza :: Orientación>
Used in
Create, **LeftRotationsNeeded**, **RightRotationsNeeded**, **RotateLeft**, **RotateLeftToOrientation**, **RotateRight**, **RotateRightToOrientation**, **SetCurrentOrientation**, **TBoard**.**CanPlay**, **TSquare**.**PossiblesOrientations**

[-] FIdPiece: integer
Identificador de la pieza numerados del 0 al 23 <A.C.: Pieza :: Jugador>
Used in
SetUsed, **TPieces**.**Create**

[-] FInternalMoving: boolean
Toma valor verdadero cuando se requiere "pensar" una movida y así no establecer los giros
Used in
Create, **TBoard**.**CanPlay**, **TSquare**.**PossiblesOrientations**

[-] FOwnerPlayer: **TPlayer**
Jugador al que está asignada la pieza <A.C.: Pieza :: Jugador>
Used in
TBoard.**PlayPiece**

[-] FSelected: boolean
Indica si ha sido seleccionada por el jugador <A.C.: Pieza :: Seleccionada>
[-] FSides: **TSides**
Lados en su orientación actual <A.C.: Pieza :: Lados>
Used in
Create, **Destroy**, **RotateLeft**, **RotateRight**, **TBoard**.**InitializePieces**,
TSquare.**CanPlayPiece**, **TSquare**.**getCoincidences**, **TSquare**.**SetPiece**

[-] FUsed: boolean
Indica si ha sido aceptada en algún Casillero de juego <A.C.: Pieza :: Usada>
Used in
SetUsed

Properties:

[+] [ro] property CurrentOrientation: **TPieceOrientation read FCurrentOrientation**
Orientación actual de la pieza <A.C.: Pieza :: Orientación>
Used in
TController.**DoOpponentMove**, **TController**.**EventOccursSquareSelectedSt**

[+] [ro] property IdPiece: integer **read FIdPiece**
Identificador de la pieza numerados del 0 al 23 <A.C.: Pieza :: Jugador>
Used in
TController.**DoNextMovement**, **TController**.**DoOpponentMove**,
TController.**EventOccursSquareSelectedSt**,
TController.**TimeOutBeforePieceSelectedSt**,
TController.**TimeOutPieceSelectedSt**, **TController**.**TimeOutSquareSelectedSt**

[+] property OwnerPlayer: **TPlayer read FOwnerPlayer write FOwnerPlayer**
Jugador al que está asignada la pieza <A.C.: Pieza :: Jugador>
Used in
SetUsed, **TBoard**.**InitializePlayers**, **TController**.**SetNewPickedObject**

[+] property Selected: boolean **read FSelected write FSelected**
Indica si ha sido seleccionada por el jugador <A.C.: Pieza :: Seleccionada>
[+] property Used: boolean **read FUsed write SetUsed**
Indica si ha sido aceptada en algún Casillero de juego <A.C.: Pieza :: Usada>
Used in
TController.**SetNewPickedObject**, **TSquare**.**SetPiece**

Constructors:

[+] constructor Create
Hereda y crea FSides. Establece FInternalMoving en false y FCurrentOrientation en ppstInitial
Called by
TPieces.**Add**

Destructors:

[+] destructor Destroy; **override**
Destruye FSides y hereda
Overrides
Destroy

Methods:

[+] function LeftRotationsNeeded(aOrientation: **TPieceOrientation**): integer
Rotaciones hacia la izquierda necesarias para llegar a la orientación de la pieza buscada
Parameters
aOrientation
Orientación buscada
Result
Cantidad de rotaciones hacia la izquierda que se necesitan para alcanzar la orientación buscada
Called by
TController.**SetCompleteMovement**, **TController**.**TimeOutSquareSelectedSt**

[+] function RightRotationsNeeded(aOrientation: **TPieceOrientation**): integer
Rotaciones hacia la derecha necesarias para llegar a la orientación de la pieza buscada
Parameters
aOrientation
Orientación buscada
Result
Cantidad de rotaciones hacia la derecha que se necesitan para alcanzar la orientación buscada
Called by
TController.**SetCompleteMovement**, **TController**.**TimeOutSquareSelectedSt**

[+] procedure RotateLeft
Rotar hacia la Izquierda <A.C.: Pieza :: Rotar hacia la Izquierda> Rota la pieza 90º en sentido antihorario. De acuerdo a la orientación actual (FCurrentOrientation) establece la siguiente. Transfiere los colores de cada lado asignado al siguiente en el sentido indicado
Called by
RotateLeftToOrientation, **TController**.**DoNextMovement**

[+] procedure RotateLeftToOrientation(aOrientation: **TPieceOrientation**)
Establece la orientación rotando la pieza hacia la izquierda
Parameters
aOrientation
Nueva Orientación a adoptar por la pieza
Called by
TBoard.**CanPlay**, **TController**.**SetCompleteMovement**, **TSquare**.**PossiblesOrientations**

[+] procedure RotateRight

Rotar hacia la Derecha <A.C.: Pieza :: Rotar hacia la Derecha> Rota la pieza 90º en sentido horario. De acuerdo a la orientación actual (FCurrentOrientation) establece la siguiente. Transfiere los colores de cada lado asignado al siguiente en el sentido indicado
Called by
RotateRightToOrientation, **TBoard**.**CanPlay**, **TController**.**DoNextMovement**, **TSquare**.**PossiblesOrientations**

[+] procedure RotateRightToOrientation(aOrientation: **TPieceOrientation**)

Establece la orientación rotando la pieza por el camino más corto
Parameters
aOrientation
Nueva Orientación a adoptar por la pieza
Called by
SetCurrentOrientation

[-] procedure SetCurrentOrientation(**const** Value: **TPieceOrientation**)

Establece la orientación buscada rotando la pieza hacia la derecha
Parameters
Value
Nueva Orientación a adoptar por la pieza
Called by
TBoard.**CanPlay**

[-] procedure SetUsed(**const** Value: boolean)

Establece a la pieza como usada y borra de la lista de piezas disponibles del jugador dueño de la pieza
Parameters
Value
Verdadero indica que la pieza se establece como usada

AIII.5.4 Class TPieces

TObject <-- ... <-- TObjectList <-- **TPieces**
Declared in Unit **Model**

No known Subclasses

Used by Classes:

TBoard

Description:

Piezas <A.C. Piezas>
Lista de Piezas formada por los 24 objetos de la clase TPiece que componen el juego.
Used in

TBoard.**Create**, **TBoard**

List of Constructors:

[#] constructor Create

Hereda y crea cada una de las 24 piezas que componen la lista asignandoles el sucesivo y respectivo FIdPiece con valores que van del 0 a 23

List of Methods:

[+] function Add
Crea un objeto de la clase TPiece y lo agrega al final de la lista
[#] function Get
Devuelva la pieza correspondiente a la posición Index
[#] procedure Put
Asigna una pieza en una posición dada

Properties:

[+] property Items [Index: integer]: **TPiece read Get write Put**; **default**
Devuelva la pieza correspondiente a la posición Index

Constructors:

[#] constructor Create
Hereda y crea cada una de las 24 piezas que componen la lista asignandoles el sucesivo y respectivo FIdPiece con valores que van del 0 a 23
Called by
TBoard.**Create**

Methods:

[+] function Add: **TPiece**
Crea un objeto de la clase TPiece y lo agrega al final de la lista
Result
Objeto creado
Called by
Create

[#] function Get(Index: integer): **TPiece**
Devuelva la pieza correspondiente a la posición Index
Parameters
Index
Número de orden de la pieza requerida
Result
Referencia al Objeto pedido

[#] procedure Put(Index: integer; aItem: **TPiece**)
Asigna una pieza en una posición dada
Parameters
Index
Número de orden de la pieza a asignar
aItem
Objeto de la clase TPiece a asignar

AIII.5.5 Class TPlayer

TObject <-- **TPlayer**
Declared in Unit **Model**

No known Subclasses

Used by Classes:

TPiece, **TBoard**, **TControllerConfig**, **TController**

Description:

Jugador <A.C.: Jugador>
Clase que define a ambos jugadores.
Used in
TBoard.**Create**, **TBoard**, **TController**, **TControllerConfig**, **TPiece**

List of Constructors:

[+] constructor Create
Hereda y crea la lista de Piezas no utilizadas y establece modo de ordenamiento de listas y modo de juego por defecto en slmSequential y spmLessEffort respectivamente.

List of Destructors:

[+] destructor Destroy
Destruye la lista de piezas no utilizadas y Hereda

Fields:

[-] FNotUsedPieces: TIntegerList
Lista de Identificadores de Piezas no Utilizadas <A.C.: Jugador :: No Definido>
Used in
Create, **Destroy**, **TBoard**.**CanPlay**, **TBoard**.**InitializePlayers**, **TPiece**.**SetUsed**

[-] FPieceColor: **TPieceColor**
Color de Piezas del jugador <A.C.: No Definido, pero reemplaza la funcionalidad del atributo "Es del usuario">

[-] FPlayMode: **TPlayMode**
Modo de Juego asignado <A.C.: Jugador :: No Definido>
Used in
Create, **TBoard**.**CanPlay**, **TBoard**.**SetPlayersMode**

[-] FPoints: integer
Puntos al momento <A.C.: Jugador :: Puntos>
Used in
TBoard.**PlayPiece**

[-] FSortedListsMode: **TSortedListsMode**
Modo de Ordenamiento de listas asignado <A.C.: Jugador :: No Definido>
Used in
Create, **TBoard**.**CanPlay**, **TBoard**.**SetPlayersMode**

Properties:

[+] property PieceColor: **TPieceColor read FPieceColor write FPieceColor**
Color de Piezas del jugador <A.C.: No Definido, pero reemplaza la funcionalidad del atributo "Es del usuario">
Used in
TBoard.**Create**, **TController**.**InitGame**, **TController**.**SetNewPickedObject**, **TController**.**TimeOutSquareSelectedSt**

[+] [ro] property Points: integer **read FPoints**
Puntos al momento <A.C.: Jugador :: Puntos>
Used in
TController.**getOpponentPoints**, **TController**.**getPlayerPoints**

Constructors:

[+] constructor Create
Hereda y crea la lista de Piezas no utilizadas y establece modo de ordenamiento de listas y modo de juego por defecto en slmSequential y spmLessEffort respectivamente.
Called by
TBoard.**Create**

Destructors:

[+] destructor Destroy; **override**
Destruye la lista de piezas no utilizadas y Hereda
Overrides
Destroy

AIII.5.6 Class TSides

TObject <-- **TSides**
Declared in Unit **Model**

No known Subclasses

Used by Classes:

TPiece, **TSquare**

Description:

Lados <A.C.: Lados>
Clase que abstrae los lados ya sea de una pieza como así también de un casillero. Las operaciones Establecer Arriba, Establecer Derecha, Establecer Abajo y Establecer Izquierda se reemplazaron publicando las propiedades tanto para la lectura como para la escritura.
Used in
TPiece.**Create**, **TSquare**.**Create**, **TPiece**, **TSquare**

Fields:

[#] FBottom: **TSideColor**
Color asignado al lado de Abajo <A.C.: Lados :: Abajo>
[#] FLeft: **TSideColor**
Color asignado al lado de la Izquierda <A.C.: Lados :: Izquierda>
[#] FRight: **TSideColor**
Color asignado al lado de la Derecha <A.C.: Lados :: Derecha>
[#] FTop: **TSideColor**
Color asignado al lado de Arriba <A.C.: Lados :: Arriba>

Properties:

[+] property Bottom: **TSideColor read FBottom write FBottom**
Valor Público de FBottom <A.C.: Lados :: Establecer Abajo>
Used in

TBoard.**InitializePieces**, **TBoard**.**InitializeSquares**,
TBoard.**PropagateSquareSides**, **TPiece**.**RotateLeft**, **TPiece**.**RotateRight**,
TSquare.**CanPlayPiece**, **TSquare**.**getCoincidences**, **TSquare**.**SetPiece**

[+] property Left: **TSideColor read FLeft write FLeft**
Valor Público de FLeft <A.C.: Lados :: Establecer Izquierda>
Used in
TBoard.**InitializePieces**, **TBoard**.**InitializeSquares**,
TBoard.**PropagateSquareSides**, **TPiece**.**RotateLeft**, **TPiece**.**RotateRight**,
TSquare.**CanPlayPiece**, **TSquare**.**getCoincidences**, **TSquare**.**SetPiece**

[+] property Right: **TSideColor read FRight write FRight**
Valor Público de FRight <A.C.: Lados :: Establecer Derecha>
Used in
TBoard.**InitializePieces**, **TBoard**.**InitializeSquares**,
TBoard.**PropagateSquareSides**, **TPiece**.**RotateLeft**, **TPiece**.**RotateRight**,
TSquare.**CanPlayPiece**, **TSquare**.**getCoincidences**, **TSquare**.**SetPiece**

[+] property Top: **TSideColor read FTop write FTop**
Valor Público de FTop <A.C.: Lados :: Establecer Arriba>
Used in
TBoard.**InitializePieces**, **TBoard**.**InitializeSquares**,
TBoard.**PropagateSquareSides**, **TPiece**.**RotateLeft**, **TPiece**.**RotateRight**,
TSquare.**CanPlayPiece**, **TSquare**.**getCoincidences**, **TSquare**.**SetPiece**

AIII.5.7 Class TSquare

TObject <-- **TSquare**
Declared in Unit **Model**

No known Subclasses

Used by Classes:

TSquares, **TMove**

Description:

Casillero de Juego <A.C.: Casillero>
Es la abstracción que para el modelo del juego tiene cada una de los casilleros posibles dónde se colocan las piezas. Deberá conocer su estado (si ha sido utilizado o no), saber si puede jugarse una pieza dada o si podría jugarse previa rotación como así también, aceptar una pieza.
Used in
TSquares.**Add**, **TSquares**.**Get**, **TMove**, **TSquares**

List of Constructors:

[+] constructor Create
Hereda y crea a sus respectivos lados (FSides)

List of Destructors:

[+] destructor Destroy
Destruye los lados y Hereda

List of Methods:

[+] function CanPlayPiece
¿Puede aceptarse la pieza en este casillero? <A.C.: Casillero :: Puede Aceptarse Pieza>

[+] function CouldPlayPiece
¿Podría aceptarse la pieza en este casillero, aunque para ello fuera necesario una rotación previa de la misma? <A.C.: Casillero :: Podría Aceptarse Pieza (previa rotación si fuera necesaria)>

[-] function getCoincidences
Devuelve la cantidad de lados coincidentes que tiene el casillero con la la pieza indicada

[+] function PossiblesOrientations
Posibles orientaciones <A.C.: Casillero :: No definida> Devuelve a aPosList con cada una de las orientaciones posibles que puede tomar la pieza sobre el casillero para poder ser aceptada en él

[-] procedure SetPiece
Establece la pieza jugada en el casillero asignando los colores de cada uno de sus lados con los de la pieza jugada.

Fields:

[-] FIdSquare: Integer
Identificador único del Casillero <A.C.: Casillero :: Identificador>
Used in
TBoard.**PropagateSquareSides**, **TSquares**.**Create**

[-] FPiece: **TPiece**
Pieza <A.C.: Casillero :: Pieza> Es la pieza que ha sido que ha sido aceptada en el casillero.
Used in
SetPiece

[-] FSides: **TSides**
Lados <A.C.: Casillero :: Lados> Lados que inicialmente tienen la información de los bordes perimetrales. Luego, al colocarse cada pieza, agregarán las restricciones que las mismas imponenen.
Used in
CanPlayPiece, **Create**, **Destroy**, **getCoincidences**, **SetPiece**, **TBoard**.**InitializeSquares**, **TBoard**.**PropagateSquareSides**

[-] FUsed: boolean
Usado <A.C.: Casillero :: Usado> Indica si una pieza ha sido aceptada en él
Used in
CanPlayPiece, **getCoincidences**, **PossiblesOrientations**, **SetPiece**

Properties:

[+] [ro] property IdSquare: integer **read FIdSquare**
Identificador único del Casillero <A.C.: Casillero :: Identificador>
Used in
TBoard.**PlayPiece**, **TController**.**DoNextMovement**, **TController**.**DoOpponentMove**, **TController**.**EventOccursSquareSelectedSt**, **TController**.**TimeOutPieceSelectedSt**, **TController**.**TimeOutSquareSelectedSt**

[+] [ro] property Piece: **TPiece read FPiece**
Pieza <A.C.: Casillero :: Pieza> Es la pieza que ha sido que ha sido aceptada en el casillero.

[+] [ro] property Used: boolean **read FUsed**
Usado <A.C.: Casillero :: Usado> Indica si una pieza ha sido aceptada en él
Used in
TBoard.**PropagateSquareSides**, **TController**.**SetNewPickedObject**

Constructors:

[+] constructor Create
Hereda y crea a sus respectivos lados (FSides)
Called by
TSquares.**Add**

Destructors:

[+] destructor Destroy; **override**
Destruye los lados y Hereda
Overrides
Destroy

Methods:

[+] function CanPlayPiece(aPiece: **TPiece**): boolean
¿Puede aceptarse la pieza en este casillero? <A.C.: Casillero :: Puede Aceptarse Pieza>
Parameters
aPiece
Pieza a evaluar
Result
True si puede jugarse la pieza en la orientación actual
Called by
PossiblesOrientations, **TBoard**.**CanPlay**,
TController.**EventOccursSquareSelectedSt**,
TController.**TimeOutSquareSelectedSt**

[+] function CouldPlayPiece(aPiece: **TPiece**; **var** aPossibleOrientation: **TPieceOrientation**): boolean
¿Podría aceptarse la pieza en este casillero, aunque para ello fuera necesario una rotación previa de la misma? <A.C.: Casillero :: Podría Aceptarse Pieza (previa rotación si fuera necesaria)>
Parameters
aPiece
Pieza a evaluar
aPossibleOrientation
Parámetro de salida que indica la orientación que podría asignarse a la pieza para que pueda jugarse
Result
True, si podría jugarse la pieza en alguna de las 4 orientación
Called by
TController.**AssignPossibleMove**, **TController**.**EventOccursSquareSelectedSt**

[-] function getCoincidences(aPiece: **TPiece**): integer
Devuelve la cantidad de lados coincidentes que tiene el casillero con la la pieza indicada
Parameters
aPiece

Pieza a evaluar
Result
Cantidad de Lados Coincidentes
Called by
TBoard.**CanPlay**

[+] function PossiblesOrientations(aPiece: **TPiece**; **var** aPosList: TStringList): boolean
Posibles orientaciones <A.C.: Casillero :: No definida> Devuelve a aPosList con cada una de las orientaciones posibles que puede tomar la pieza sobre el casillero para poder ser aceptada en él
Parameters
aPiece
Pieza a evaluar
aPosList
Lista de Orientaciones posibles
Result
True, si al menos es posible jugar la pieza en una orientación cualquiera
Called by
CouldPlayPiece, **TBoard**.**CanPlay**, **TBoard**.**CanPlayPieceInAnySquare**

[-] procedure SetPiece(**const** Value: **TPiece**)
Establece la pieza jugada en el casillero asignando los colores de cada uno de sus lados con los de la pieza jugada. Tanto la pieza como el casillero quedan con su propiedad Used en true.
Parameters
Value
Pieza aceptada en el casillero
Called by
TBoard.**PlayPiece**

AIII.5.8 Class TSquares

TObject <-- ... <-- TObjectList <-- **TSquares**
Declared in Unit **Model**

No known Subclasses

Used by Classes:

TBoard

Description:

Casilleros <A.C. Casilleros>
Lista de Casilleros formada por objetos de la clase TSquare
Used in
TBoard.**Create**, **TBoard**

List of Constructors:

[#] constructor Create
Hereda y crea cada uno de las 24 casilleros que componen la lista asignandoles el sucesivo y respectivo FIdSquare con valores que van del 0 a 23

List of Methods:

[+] function Add
TSquares Crea un objeto de la clase TSquare y lo agrega al final de la lista
[#] function Get
Devuelva el casillero correspondiente a la posición Index
[#] procedure Put
Asigna un casillero en una posición dada

Properties:

[+] property Items [Index: integer]: **TSquare read Get write Put**; **default**
Devuelva el casillero correspondiente a la posición Index

Constructors:

[#] constructor Create
Hereda y crea cada uno de las 24 casilleros que componen la lista asignandoles el sucesivo y respectivo FIdSquare con valores que van del 0 a 23
Called by
TBoard.**Create**

Methods:

[+] function Add: **TSquare**
TSquares Crea un objeto de la clase TSquare y lo agrega al final de la lista
Result
Objeto creado
Called by
Create

[#] function Get(Index: integer): **TSquare**
Devuelva el casillero correspondiente a la posición Index
Parameters
Index
Número de orden del casillero requerida
Result
Referencia al Objeto pedido

[#] procedure Put(Index: integer; aItem: **TSquare**)
Asigna un casillero en una posición dada
Parameters
Index
Número de orden del casillero a asignar
aItem
Objeto de la clase TSquare a asignar

AIII.6 Unit View

Used Units:

Used in interface: System, SysInit, GLScene, GLVectorFileObjects, Types, GLMisc, GLTexture, **Common**, UObjectList
Used in implementation: SysUtils, GLUtils, Forms, Dialogs, VectorGeometry, Graphics

Used by:
In Interface of Units: FMain

Description:

En esta unidad están definidas todas las clases visuales. A partir de una clase básica general (TGLBasicForm) se heredan cada una de las clases que componen el juego. En el caso particular de la pieza visual, su herencia no es directa, dado que se define previamente una clase que contiene todos los movimientos posibles llamada TGLDynamicForm. Otro caso particular es el de la clase indicadores de giro, que se forma a partir de ambos indicadores. Esta conjunción facilitará el traslado de los mismos de una pieza a otra. También se definen las constantes del nombre del archivo 3ds que se asigna a cada objeto visual.

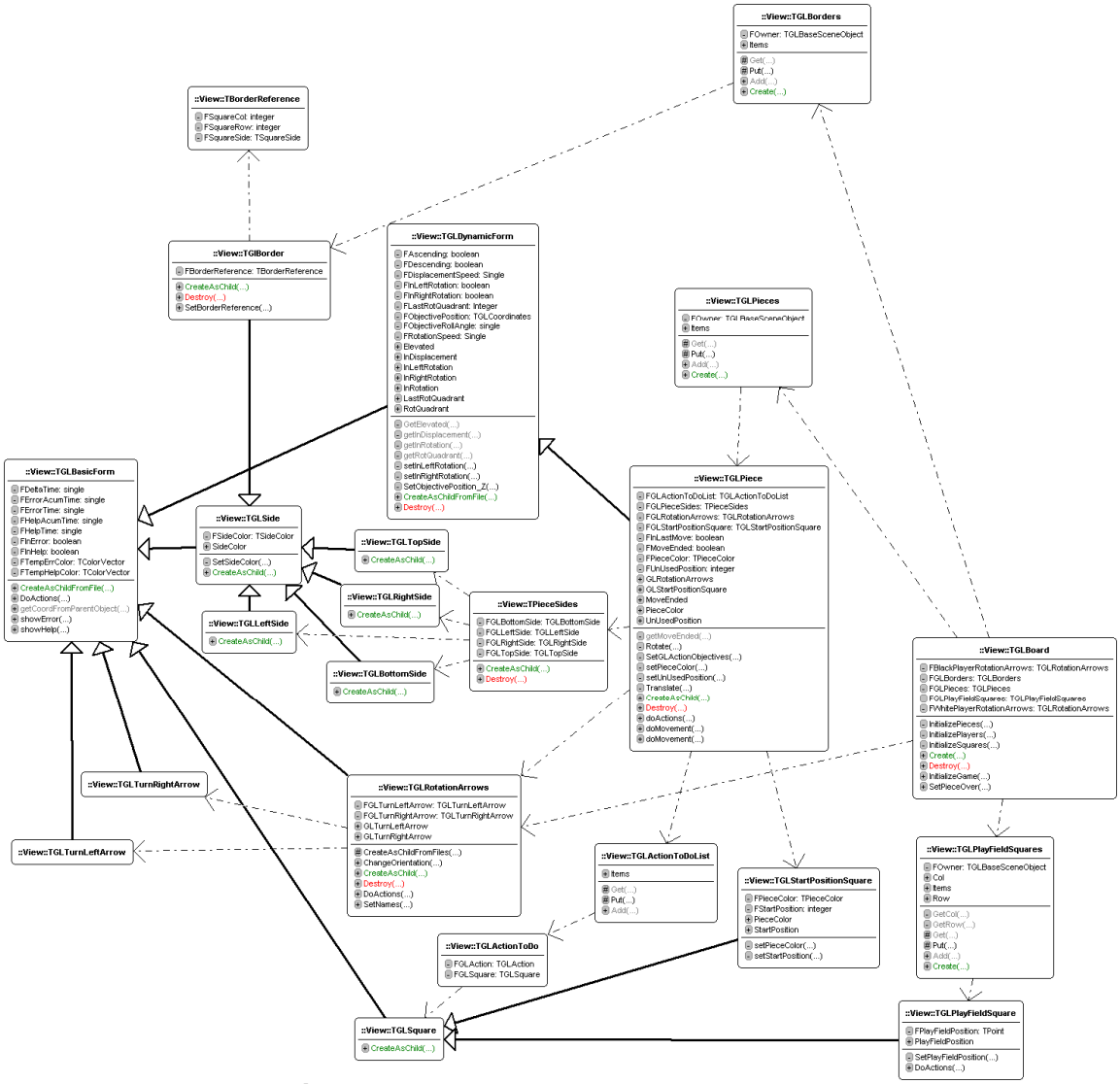


Ilustración Anexo III - 5 - Diagrama de Clases de View

List of Constants:

Global Constants: ARROWSCENTER_FILE_NAME, **glactGoDown**, **glactGoOverSquare**, **glactGoUp**, **glactHideArrows**, **glactRotateLeft**, **glactRotateRight**, **glactShowArrows**, **GL_PIECE_BASE_ELEV**, **GL_PIECE_ELEVATED_ELEV**, **LEFTARROW_FILE_NAME**, **PIECE_FILE_NAME**, **POINTER_FILE_NAME**, **RIGHTARROW_FILE_NAME**, **SIDE_FILE_NAME**, **sqsBottom**, **sqsLeft**, **sqsRight**, **sqsTop**, **SQUARE_FILE_NAME**
File-Local Constants: *None*

List of Simple Types:

Global Simple Types: TGLAction, **TSquareSide**
File-Local Simple Types: *None*

List of Classes:

Global Classes: TBorderReference, **TGLActionToDo**, **TGLActionToDoList**, **TGLBasicForm**, **TGLBoard**, **TGlBorder**, **TGLBorders**, **TGLBottomSide**, **TGLDynamicForm**, **TGLLeftSide**, **TGLPiece**, **TGLPieces**, **TGLPlayFieldSquare**, **TGLPlayFieldSquares**, **TGLRightSide**, **TGLRotationArrows**, **TGLSide**, **TGLSquare**, **TGLStartPositionSquare**, **TGLTopSide**, **TGLTurnLeftArrow**, **TGLTurnRightArrow**, **TPieceSides**
File-Local Classes: *None*

List of Functions:

Global Functions:
[ui] function getGLClassName
Función auxiliar para determinar la clase a la que pertence un objecto
[ui] function getQuadrant
Función auxiliar para determinar a que cuadrante pertenece un ángulo dado
File-Local Functions: *None*

Constants:

[ui] ARROWSCENTER_FILE_NAME = 'center.3ds'
Used in
TGLRotationArrows.**CreateAsChild**
[ui] glactGoDown = **TGLAction**(4)
Desplazarse verticalmente hasta la posición indicada por GL_PIECE_BASE_ELEV
Used in
TGLPiece.**doMovement**, **TGLPiece**.**SetGLActionObjectives**
[ui] glactGoOverSquare = **TGLAction**(0)
Desplazarse horizontalmente hasta arriba de un casillero
Used in
TGLPiece.**doMovement**, **TGLPiece**.**SetGLActionObjectives**, **TGLPiece**.**Translate**
[ui] glactGoUp = **TGLAction**(3)
Desplazarse verticalmente hasta la posición indicada por GL_PIECE_ELEVATED_ELEV
Used in
TGLPiece.**doMovement**, **TGLPiece**.**SetGLActionObjectives**
[ui] glactHideArrows = **TGLAction**(6)
Ocultar Indicadores de Rotación
Used in
TGLPiece.**doMovement**, **TGLPiece**.**SetGLActionObjectives**
[ui] glactRotateLeft = **TGLAction**(1)
Rotar hacia la Izquierda 90º
Used in
TGLPiece.**doActions**, **TGLPiece**.**doMovement**, **TGLPiece**.**SetGLActionObjectives**

[ui] glactRotateRight = **TGLAction**(2)
Rotar hacia la Derecha 90º
Used in
TGLPiece.**doActions**, **TGLPiece**.**doMovement**, **TGLPiece**.**SetGLActionObjectives**
[ui] glactShowArrows = **TGLAction**(5)
Mostrar Indicadores de Rotación
Used in
TGLPiece.**SetGLActionObjectives**
[ui] GL_PIECE_BASE_ELEV = 0.25
Used in
TGLDynamicForm.**GetElevated**, **TGLPiece**.**SetGLActionObjectives**,
TGLPiece.**setUnUsedPosition**, **TGLSquare**.**CreateAsChild**
[ui] GL_PIECE_ELEVATED_ELEV = 0.75
Used in
TGLPiece.**SetGLActionObjectives**, **TGLRotationArrows**.**CreateAsChildFromFiles**
[ui] LEFTARROW_FILE_NAME = 'ArrowTurnLeft.3ds'
Used in
TGLRotationArrows.**CreateAsChild**
[ui] PIECE_FILE_NAME = 'Piece.3ds'
Constantes de nombres de Archivos 3D
Used in
TGLPiece.**CreateAsChild**
[ui] POINTER_FILE_NAME = 'Pointer.3ds'
[ui] RIGHTARROW_FILE_NAME = 'ArrowTurnRight.3ds'
Used in
TGLRotationArrows.**CreateAsChild**
[ui] SIDE_FILE_NAME = 'Side.3ds'
Used in
TGlBorder.**CreateAsChild**, **TGLSide**.**CreateAsChild**
[ui] sqsBottom = **TSquareSide**(2)
Lado Inferior
Used in
TGLBoard.**InitializeSquares**, **TGlBorder**.**SetBorderReference**
[ui] sqsLeft = **TSquareSide**(3)
Lado Izquierdo
Used in
TGLBoard.**InitializeSquares**, **TGlBorder**.**SetBorderReference**
[ui] sqsRight = **TSquareSide**(1)
Lado Derecho
Used in
TGLBoard.**InitializeSquares**, **TGlBorder**.**SetBorderReference**
[ui] sqsTop = **TSquareSide**(0)
Lado Superior
Used in
TGLBoard.**InitializeSquares**, **TGlBorder**.**SetBorderReference**
[ui] SQUARE_FILE_NAME = 'Square.3ds'
Used in
TGLSquare.**CreateAsChild**

Simple Types:

[ui] TGLAction = (**glactGoOverSquare**, **glactRotateLeft**, **glactRotateRight**, **glactGoUp**, **glactGoDown**, **glactShowArrows**, **glactHideArrows**)

Acciones
Enumeración de las posibles acciones a realizar por la pieza que permitirán realizar las órdenes dadas por elmodelo a través de los comandos

[ui] TSquareSide = (**sqsTop**, **sqsRight**, **sqsBottom**, **sqsLeft**)

Lado de casillero
Enumeración de los 4 lados del casillero

Functions:

[ui] function getGLClassName(aNameStr: String): TCLass

Función auxiliar para determinar la clase a la que pertence un objecto
Parameters
aNameStr
Nombre del Objeto
Result
Clase asociada al nombre del objeto

[ui] function getQuadrant(aRollAngle: single): integer

Función auxiliar para determinar a que cuadrante pertenece un ángulo dado
Parameters
aRollAngle
Ángulo del cual se quiere conocer el cuadrante
Result
1: Superior Derecho, 2: Inferior Derecho, 3: Inferior Izquierdo, 4: Superior Izquierdo
Called by
TGLDynamicForm.**getRotQuadrant**, **TGLPiece**.**Rotate**

AIII.6.1 Class TBorderReference

TObject <-- **TBorderReference**
Declared in Unit **View**

No known Subclasses

Used by Classes:

TGlBorder

Description:

Referencia del Borde <A.C.: No Definida>
Permite identificar cómo se va a ubicar el borde en el tablero de acuerdo a la fila, la columna del casillero y el lado de casillero contiguo.
Used in

TGlBorder.**CreateAsChild**, **TGlBorder**

Fields:

[-] FSquareCol: integer

Columna del casillero
Used in
TGlBorder.**SetBorderReference**

[-] FSquareRow: integer
Fila del casillero
Used in
TGlBorder.**SetBorderReference**

[-] FSquareSide: **TSquareSide**
Lado del casillero
Used in
TGlBorder.**SetBorderReference**

AIII.6.2 Class TGLActionToDo

TObject <-- **TGLActionToDo**
Declared in Unit **View**

No known Subclasses

Used by Classes:

TGLActionToDoList

Description:

Acción a realizar: <A.C.: No definida> Simple clase auxiliar que se define por una acción y un casillero que puede o no asignarse
Used in
TGLActionToDoList.**Add**, **TGLActionToDoList**.**Get**, **TGLActionToDoList**

Fields:

[-] FGLAction: **TGLAction**
Acción
Used in
TGLPiece.**doActions**, **TGLPiece**.**doMovement**, **TGLPiece**.**doMovement**, **TGLPiece**.**SetGLActionObjectives**, **TGLPiece**.**Translate**

[-] FGLSquare: **TGLSquare**
Casillero asociado a la acción
Used in
TGLPiece.**doMovement**, **TGLPiece**.**SetGLActionObjectives**, **TGLPiece**.**Translate**

AIII.6.3 Class TGLActionToDoList

TObject <-- ... <-- TObjectList <-- **TGLActionToDoList**
Declared in Unit **View**

No known Subclasses

Used by Classes:

TGLPiece

Description:

Lista de Acciones a realizar: <A.C.: No definida> Clase auxiliar que define permite almacenar los sucesivas acciones a realizar por una pieza

Used in

TGLPiece.CreateAsChild, **TGLPiece**

List of Methods:

[+] function Add

Crea un objeto de la clase TGLActionToDo y lo agrega al final de la lista

[#] function Get

Devuelva la acción correspondiente

[#] procedure Put

Asigna una acción en una posición dada

Properties:

[+] property Items [Index: integer]: **TGLActionToDo read Get write Put**; **default**

Devuelva la acción correspondiente

Methods:

[+] function Add: **TGLActionToDo**

Crea un objeto de la clase TGLActionToDo y lo agrega al final de la lista

Result

Objeto creado

Called by

TGLPiece.doMovement, **TGLPiece.doMovement**

[#] function Get(Index: integer): **TGLActionToDo**

Devuelva la acción correspondiente

Parameters

Index

Número de orden de la acción requerida

Result

Referencia al Objeto pedido

[#] procedure Put(Index: integer; aItem: **TGLActionToDo**)

Asigna una acción en una posición dada

Parameters

Index

Número de orden de la acción a asignar

aItem

Objeto de la clase TGLAction a asignar

AIII.6.4 Class TGLBasicForm

TObject <-- ... <-- TGLActor <-- **TGLBasicForm**
Declared in Unit **View**

Direct known Subclasses:

TGLDynamicForm, **TGLRotationArrows**, **TGLSide**, **TGLSquare**, **TGLTurnLeftArrow**, **TGLTurnRightArrow**

Not used by any known Class

Description:

Forma Estática Básica <A.C.: No Definida>
Clase que hereda de la clase TGLActor de la biblioteca GLScene que permite incluir en las escenas objetos 3D en formato de archivos 3DS.
De esta clase, van a heredar las distintas clases visuales especializadas. Incluye la transformación del objeto visual tanto para el caso de tener que mostrar que ha sido indicado erróneamente como para el caso de tener que mostrar la ayuda
Used in

TGLPlayFieldSquare.**DoActions**, **TMainForm**.**getGLObjectByName**, **TMainForm**.**GLCadencerProgress**

List of Constructors:

[+] constructor CreateAsChildFromFile
Crea al Objeto a partir de un archivo 3DS como hijo de otro objeto del tipo TGLBaseSceneObject

List of Methods:

[+] procedure DoActions
Actualiza FDeltaTime.
[+] function getCoordFromParentObject
Devuelve las coordenadas relativas a un objeto superior en el árbol de dependencias
[+] procedure showError
Procedimiento virtual que muestra un error sobre el objeto.
[+] procedure showHelp
Procedimiento virtual que muestra la ayuda que brinda el objeto.

Fields:

[-] FDeltaTime: single
Tiempo transcurrido desde la última acción tomada que se usa para definir las translaciones y rotaciones a realizar de acuerdo a las respectivas velocidades consideradas
Used in
DoActions, **TGLPiece**.**Rotate**, **TGLPiece**.**Translate**

[-] FErrorAcumTime: single
Tiempo acumulado para contabilizar el tiempo durante el que se muestra el error (en segundos)
Used in
DoActions, **showError**

[-] FErrorTime: single
Tiempo durante el cual se muestra el error (en segundos)
Used in
DoActions, **showError**

[-] FHelpAcumTime: single
Tiempo acumulado para contabilizar el tiempo durante el que se muestra la ayuda (en segundos)
Used in
DoActions, **showHelp**

[-] FHelpTime: single
Tiempo durante el cual se muestra la ayuda (en segundos)
Used in
DoActions, **showHelp**

[-] FInError: boolean
En estado de error
Used in
DoActions, **showError**

[-] FInHelp: boolean
En estado de ayuda
Used in
DoActions, **showHelp**

[-] FTempErrColor: TColorVector
Color temporario donde se guarda el color original de la forma básica que se encuentra en estado de error
Used in
DoActions, **showError**

[-] FTempHelpColor: TColorVector
Color temporario donde se guarda el color original de la forma básica que se encuentra en estado de ayuda
Used in
DoActions, **showHelp**

Constructors:

[+] constructor CreateAsChildFromFile(aParentOwner: TGLBaseSceneObject; aFileName: String); **virtual**
Crea al Objeto a partir de un archivo 3DS como hijo de otro objeto del tipo TGLBaseSceneObject
Parameters
aParentOwner
Objeto Padre Visual (las posiciones se tomarán sobre él. Los movimientos del Padre Visual, arrastran al de los hijos).
aFileName
Nombre y Path completo del archivo 3DS
Overridden by
TGLDynamicForm.**CreateAsChildFromFile**
Called by
TGlBorder.**CreateAsChild**, **TGLDynamicForm**.**CreateAsChildFromFile**, **TGLRotationArrows**.**CreateAsChildFromFiles**, **TGLSide**.**CreateAsChild**, **TGLSquare**.**CreateAsChild**

Methods:

[+] procedure DoActions(aDeltaTime: Single); **virtual**
Actualiza FDeltaTime. Las clases que heredan agregarán sus acciones propias. Incrementa los valores de FErrorAcumTime (si está en error) y FHelpAcumTime (si está en ayuda). En caso de haberse superado el tiempo respectivo, se vuelve a la situación original).
Parameters

aDeltaTime
Variación del tiempo entre la escena inmediata anterior y la actual
Overridden by
TGLPiece.**doActions**, **TGLRotationArrows**.**DoActions**,
TGLPlayFieldSquare.**DoActions**
Called by
TGLPiece.**doActions**, **TGLPlayFieldSquare**.**DoActions**,
TGLRotationArrows.**DoActions**, **TMainForm**.**GLCadencerProgress**

[+] function getCoordFromParentObject(aGLObj: TGLSceneObject; aCoord: String): single
Devuelve las coordenadas relativas a un objeto superior en el árbol de dependencias
Parameters
aGLObj
Objeto padre referencial
aCoord
X, Y o Z según lo que se requiera
Result
Valor de la coordenada resultante
Called by
TMainForm.**ShowHelp**

[+] procedure showError(aTime: single); **virtual**
Procedimiento virtual que muestra un error sobre el objeto.
Parameters
aTime
Duración del estado de ayuda
Called by
TMainForm.**ShowError**

[+] procedure showHelp(aTime: single); **virtual**
Procedimiento virtual que muestra la ayuda que brinda el objeto.
Parameters
aTime
Duración del estado de error
Called by
TMainForm.**ShowHelp**

AIII.6.5 Class TGLBoard

TObject <-- **TGLBoard**
Declared in Unit **View**

No known Subclasses

Used by Classes:

TMainForm

Description:

Used in
TMainForm.**btnStartGameClick**, **TMainForm**

List of Constructors:

[+] constructor Create
Hereda y crea a FWhitePlayer, FBlackPlayer, FPieces, FSquares, FBorders y FNotUsedSquares

List of Destructors:

[+] destructor Destroy

List of Methods:

[+] procedure InitializeGame
Inicializa la partida <A.C.

[-] procedure InitializePieces
Inicializar Piezas <A.C.: Tablero :: Inicializar Piezas> Establece los colores de los bordes de cada pieza, a los de posición inicial y agrega cada uno a la lista de piezas no utilizadas

[-] procedure InitializePlayers
Inicilizar Jugadores <A.C.: Tablero :: Inicializar Jugadores> Asigna las piezas al jugador correspondiente y establece la posición inicial relativa (del 1 al 12 para cada jugador)

[-] procedure InitializeSquares
Inicializar Casilleros <A.C.: Tablero :: Inicializar Casilleros> Asigna los colores a cada lado y ubica al casillero visual de acuerdo a la posición de la fila / columna asignada

[+] procedure SetPieceOver
Mensaje que indica que se coloque la pieza dada sobre el casillero dado

Fields:

[-] FBlackPlayerRotationArrows: **TGLRotationArrows**
Flechas de rotación del Jugador de Piezas Negras <A.C.: Tablero :: Jugador 2>
Used in
Create, **Destroy**, **InitializePlayers**

[-] FGLBorders: **TGLBorders**
Lista de Bordes Visuales del Juego
Used in
Create, **Destroy**, **InitializeSquares**

[-] FGLPieces: **TGLPieces**
Lista de Piezas Visuales del Juego
Used in
Create, **Destroy**, **InitializePieces**, **InitializePlayers**

[-] FGLPlayFieldSquares: **TGLPlayFieldSquares**
Lista de Casilleros Visuales del Juego
Used in
Create, **Destroy**, **InitializeSquares**

[-] FWhitePlayerRotationArrows: **TGLRotationArrows**
Flechas de rotación del jugador de Piezas Blancas <A.C.: Tablero :: Jugador 1>
Used in
Create, **Destroy**, **InitializePlayers**

Constructors:

[+] constructor Create(aParentOwner: TGLBaseSceneObject)
Hereda y crea a FWhitePlayer, FBlackPlayer, FPieces, FSquares, FBorders y FNotUsedSquares
Parameters
aParentOwner
Objeto Padre Visual de Piezas y Casilleros (Tablero Visual)
Called by
TMainForm.btnStartGameClick

Destructors:

[+] destructor Destroy; **override**
Overrides
Destroy

Methods:

[+] procedure InitializeGame
Inicializa la partida <A.C. Inicializar Tablero Visual>
Called by
TMainForm.btnStartGameClick

[-] procedure InitializePieces
Inicializar Piezas <A.C.: Tablero :: Inicializar Piezas> Establece los colores de los bordes de cada pieza, a los de posición inicial y agrega cada uno a la lista de piezas no utilizadas
Called by
InitializeGame

[-] procedure InitializePlayers
Inicilizar Jugadores <A.C.: Tablero :: Inicializar Jugadores> Asigna las piezas al jugador correspondiente y establece la posición inicial relativa (del 1 al 12 para cada jugador)
Called by
InitializeGame

[-] procedure InitializeSquares
Inicializar Casilleros <A.C.: Tablero :: Inicializar Casilleros> Asigna los colores a cada lado y ubica al casillero visual de acuerdo a la posición de la fila / columna asignada
Called by
InitializeGame

[+] procedure SetPieceOver(aPiece: **TGLPiece**; aSquare: **TGLSquare**)
Mensaje que indica que se coloque la pieza dada sobre el casillero dado
Parameters
aPiece
Pieza a desplazar
aSquare
Casillero hasta donde debe desplazarse encima la pieza indicada

AIII.6.6 Class TGlBorder

TObject <-- ... <-- TGLActor <-- **TGLBasicForm** <-- **TGLSide** <-- **TGlBorder**
Declared in Unit **View**

No known Subclasses

Used by Classes:

TGLBorders

Description:

Borde de Tablero Visual <A.C.: Borde de Tablero Visual>
Brinda el marco visual que necesita el usuario para saber que el lado de la pieza que va a colocar en forma contigua, debe tener el mismo color.
Used in
TGLBorders.**Add**, **TGLBorders**.**Get**, **getGLClassName**, **TGLBorders**

List of Constructors:

[+] constructor CreateAsChild
Invoca a CreateAsChildFromFiles

List of Destructors:

[+] destructor Destroy

List of Methods:

[+] procedure SetBorderReference
Posiciona el borde visual

Fields:

[-] FBorderReference: **TBorderReference**
Referencia para situar el borde de acuerdo al casillero
Used in
CreateAsChild, **Destroy**, **SetBorderReference**

Inherited Fields:

Fields in TGLSide:
[-] FSideColor
Fields in TGLBasicForm:
[-] FDeltaTime, **[-] FErrorAcumTime**, **[-] FErrorTime**, **[-] FHelpAcumTime**, **[-] FHelpTime**, **[-] FInError**, **[-] FInHelp**, **[-] FTempErrColor**, **[-] FTempHelpColor**

Inherited Fields by Name:

[-] TGLBasicForm.**FDeltaTime**, **[-] TGLBasicForm**.**FErrorAcumTime**, **[-] TGLBasicForm**.**FErrorTime**, **[-] TGLBasicForm**.**FHelpAcumTime**, **[-] TGLBasicForm**.**FHelpTime**, **[-] TGLBasicForm**.**FInError**, **[-] TGLBasicForm**.**FInHelp**, **[-] TGLSide**.**FSideColor**, **[-] TGLBasicForm**.**FTempErrColor**, **[-] TGLBasicForm**.**FTempHelpColor**

Properties:

Inherited Properties:

Properties in TGLSide:
[+] SideColor

Inherited Properties by Name:

[+] TGLSide.**SideColor**

Constructors:

[+] constructor CreateAsChild(aParentOwner: TGLBaseSceneObject)
Invoca a CreateAsChildFromFiles
Parameters
aParentOwner
Objeto Padre Visual
Called by
TGLBorders.**Add**

Inherited Constructors:

Constructors in TGLSide:
[+] CreateAsChild
Constructors in TGLBasicForm:
[+] CreateAsChildFromFile

Inherited Constructors by Name:

[+] TGLSide.**CreateAsChild**, **[+] TGLBasicForm**.**CreateAsChildFromFile**

Destructors:

[+] destructor Destroy; **override**
Overrides
Destroy

Methods:

[+] procedure SetBorderReference(aSquareCol, aSquareRow: integer; aSquareSide: **TSquareSide**)
Posiciona el borde visual
Parameters
aSquareCol
Columna a ocupar
aSquareRow
Fila a ocupar
aSquareSide
Ubicación respecto del casillero
Called by
TGLBoard.**InitializeSquares**

Inherited Methods:

Methods in TGLSide:
[-] SetSideColor
Methods in TGLBasicForm:
[+] DoActions, **[+] getCoordFromParentObject**, **[+] showError**, **[+] showHelp**

Inherited Methods by Name:

[+] TGLBasicForm.**DoActions**, **[+] TGLBasicForm**.**getCoordFromParentObject**, **[-] TGLSide**.**SetSideColor**, **[+] TGLBasicForm**.**showError**, **[+] TGLBasicForm**.**showHelp**

AIII.6.7 Class TGLBorders

TObject <-- ... <-- TObjectList <-- **TGLBorders**
Declared in Unit **View**

No known Subclasses

Used by Classes:

TGLBoard

Description:

Bordes <A.C.: Bordes de Tablero Visual>
Es el contenedor de cada uno de los 24 Bordes de Tablero Visual
Used in
TGLBoard.**Create**, **TGLBoard**

List of Constructors:

[+] constructor Create
Hereda y asigna el FOwner de los bordes Visuales

List of Methods:

[+] function Add
Crea un objeto de la clase TGLBorder y lo agrega al final de la lista
[#] function Get
Devuelva el borde visual correspondiente
[#] procedure Put
Asigna un borde visual en una posición dada

Fields:

[-] FOwner: TGLBaseSceneObject
Used in
Add, **Create**

Properties:

[+] property Items [Index: integer]: **TGlBorder read Get write Put**; **default**
Devuelva el borde visual correspondiente

Constructors:

[+] constructor Create(aParentOwner: TGLBaseSceneObject)
Hereda y asigna el FOwner de los bordes Visuales
Parameters
aParentOwner
Objeto Padre Visual de los bordes (Tablero Visual)
Called by
TGLBoard.**Create**

Methods:

[+] function Add: **TGlBorder**
Crea un objeto de la clase TGLBorder y lo agrega al final de la lista
Result
Objeto creado

Called by
Create

[#] function Get(Index: integer): **TGlBorder**
Devuelva el borde visual correspondiente
Parameters
Index
Número de orden del borde visual requerido
Result
Referencia al Objeto pedido

[#] procedure Put(Index: integer; aItem: **TGlBorder**)
Asigna un borde visual en una posición dada
Parameters
Index
Número de orden del borde visual a asignar
aItem
Objeto de la clase TGLBorder a asignar

AIII.6.8 Class TGLBottomSide

TObject <-- ... <-- TGLActor <-- **TGLBasicForm** <-- **TGLSide** <-- **TGLBottomSide**
Declared in Unit **View**

No known Subclasses

Used by Classes:

TPieceSides

Description:

Lado Visual Inferior <A.C.: No Definida>
Forma Visual correspondiente al Lados inferiores y Bordes superiores.
Used in
TMainForm.**GLSceneViewerMouseDown**, **TPieceSides**.**CreateAsChild**, **TPieceSides**

List of Constructors:

[+] constructor CreateAsChild
Hereda y rota el lado 270°

Fields:

Inherited Fields:

Fields in TGLSide:
[-] FSideColor
Fields in TGLBasicForm:
[-] FDeltaTime, **[-] FErrorAcumTime**, **[-] FErrorTime**, **[-] FHelpAcumTime**, **[-] FHelpTime**, **[-] FInError**, **[-] FInHelp**, **[-] FTempErrColor**, **[-] FTempHelpColor**

Inherited Fields by Name:

[-] TGLBasicForm.**FDeltaTime**, **[-] TGLBasicForm**.**FErrorAcumTime**, **[-] TGLBasicForm**.**FErrorTime**, **[-] TGLBasicForm**.**FHelpAcumTime**, **[-] TGLBasicForm**.**FHelpTime**, **[-] TGLBasicForm**.**FInError**, **[-] TGLBasicForm**.**FInHelp**, **[-**

**] TGLSide.FSideColor, [-] TGLBasicForm.FTempErrColor, [-
] TGLBasicForm.FTempHelpColor**

Properties:

Inherited Properties:

Properties in TGLSide:
[+] SideColor

Inherited Properties by Name:

[+] TGLSide.SideColor

Constructors:

[+] constructor CreateAsChild(aParentOwner: TGLBaseSceneObject)
Hereda y rota el lado 270°
Parameters
aParentOwner
Objeto Padre Visual
Called by
TPieceSides.CreateAsChild

Inherited Constructors:

Constructors in TGLSide:
[+] CreateAsChild
Constructors in TGLBasicForm:
[+] CreateAsChildFromFile

Inherited Constructors by Name:

[+] TGLSide.CreateAsChild, [+] TGLBasicForm.CreateAsChildFromFile

Methods:

Inherited Methods:

Methods in TGLSide:
[-] SetSideColor
Methods in TGLBasicForm:
[+] DoActions, [+] getCoordFromParentObject, [+] showError, [+] showHelp

Inherited Methods by Name:

**[+] TGLBasicForm.DoActions, [+] TGLBasicForm.getCoordFromParentObject, [-
] TGLSide.SetSideColor, [+] TGLBasicForm.showError, [+] TGLBasicForm.showHelp**

AIII.6.9 Class TGLDynamicForm

TObject <-- ... <-- TGLActor <-- **TGLBasicForm** <-- **TGLDynamicForm**
Declared in Unit **View**

Direct known Subclasses:

TGLPiece

Not used by any known Class

Description:

Objetos Visuales Dinámicos <A.C.: No Definida> Clase Visual Virtual que define a los objetos con desplazamiento en la pantalla tanto de rotación como de traslación.

List of Constructors:

[+] constructor CreateAsChildFromFile
Hereda, crea el FObjectivePosition e inicializa valores

List of Destructors:

[+] destructor Destroy
Destruye a FObjectivePosition y Hereda.

List of Methods:

[-] function GetElevated
Verifica si está elevado
[-] function getInDisplacement
Compara las coordenadas de la FObjectivePosition con las actuales
[-] function getInRotation
Verifica si FRigthRotation o FLeftRotation están en true
[-] function getRotQuadrant
Verifica cual es el cuadrante actual de acuerdo al valor de la propiedad heredada RollAngle.
[-] procedure setInLeftRotation
Establece FLeftRotation y actualiza FLastQuadrant con el valor actual
[-] procedure setInRightRotation
Establece FRightRotation y actualiza FLastQuadrant con el valor actual
[-] procedure SetObjectivePosition_Z
Establece el nuevo valor de elevación de la pieza

Fields:

[-] FAscending: boolean
Indica si el objeto está en movimiento ascendente
Used in
SetObjectivePosition_Z, **TGLPiece**.**Translate**

[-] FDescending: boolean
Indica si el objeto está en movimiento descendente
Used in
SetObjectivePosition_Z, **TGLPiece**.**Translate**

[-] FDisplacementSpeed: Single
Velocidad de desplazamiento en unidades gráficas por segundo
Used in
CreateAsChildFromFile, **TGLPiece**.**Translate**

[-] FInLeftRotation: boolean
La Rotación que está haciendo es hacia la Izquierda
Used in
getInRotation, **setInLeftRotation**, **TGLPiece**.**Rotate**

[-] FInRightRotation: boolean
La Rotación que está haciendo es hacia la Derecha
Used in
getInRotation, **setInRightRotation**, **TGLPiece**.**Rotate**

[-] FLastRotQuadrant: Integer
Último cuadrante ocupado
Used in
setInLeftRotation, **setInRightRotation**, **TGLPiece**.**Rotate**

[-] FObjectivePosition: TGLCoordinates
Posición Objetivo a dónde deberá desplazarse el objeto
Used in
CreateAsChildFromFile, **Destroy**, **getInDisplacement**, **SetObjectivePosition_Z**, **TGLPiece**.**SetGLActionObjectives**, **TGLPiece**.**setUnUsedPosition**, **TGLPiece**.**Translate**

[-] FObjectiveRollAngle: single
Ángulo Objetivo hasta el cual deberá girar el objeto
Used in
CreateAsChildFromFile, **TGLPiece**.**Rotate**, **TGLPiece**.**SetGLActionObjectives**

[-] FRotationSpeed: Single
Velocidad de rotación en Grados Sexagesimales por segundo
Used in
CreateAsChildFromFile, **TGLPiece**.**Rotate**

Inherited Fields:

Fields in TGLBasicForm:
[-] FDeltaTime, **[-] FErrorAcumTime**, **[-] FErrorTime**, **[-] FHelpAcumTime**, **[-] FHelpTime**, **[-] FInError**, **[-] FInHelp**, **[-] FTempErrColor**, **[-] FTempHelpColor**

Inherited Fields by Name:

[-] TGLBasicForm.**FDeltaTime**, **[-] TGLBasicForm**.**FErrorAcumTime**, **[-] TGLBasicForm**.**FErrorTime**, **[-] TGLBasicForm**.**FHelpAcumTime**, **[-] TGLBasicForm**.**FHelpTime**, **[-] TGLBasicForm**.**FInError**, **[-] TGLBasicForm**.**FInHelp**, **[-] TGLBasicForm**.**FTempErrColor**, **[-] TGLBasicForm**.**FTempHelpColor**

Properties:

[+] [ro] property Elevated: boolean **read GetElevated**
Verifica si está elevado
[+] [ro] property InDisplacement: boolean **read getInDisplacement**
Compara las coordenadas de la FObjectivePosition con las actuales
Used in
TGLPiece.**doActions**

[+] property InLeftRotation: boolean **read FInLeftRotation write setInLeftRotation**
La Rotación que está haciendo es hacia la Izquierda
Used in
CreateAsChildFromFile, **TGLPiece**.**SetGLActionObjectives**

[+] property InRightRotation: boolean **read FInRightRotation write setInRightRotation**
La Rotación que está haciendo es hacia la Derecha
Used in
CreateAsChildFromFile, **TGLPiece**.**SetGLActionObjectives**

[+] [ro] property InRotation: boolean **read getInRotation**
Verifica si FRigthRotation o FLeftRotation están en true
Used in
TGLPiece.doActions

[+] property LastRotQuadrant: Integer **read FLastRotQuadrant write FLastRotQuadrant**
Último cuadrante ocupado
[+] [ro] property RotQuadrant: Integer **read getRotQuadrant**
Verifica cual es el cuadrante actual de acuerdo al valor de la propiedad heredada RollAngle.
Used in
setInLeftRotation, **setInRightRotation**

Constructors:

[+] constructor CreateAsChildFromFile(aParentOwner: TGLBaseSceneObject; aFileName: String); **override**
Hereda, crea el FObjectivePosition e inicializa valores
Parameters
aParentOwner
Objeto Padre Visual.
aFileName
Nombre y Path completo del archivo 3DS
Overrides
TGLBasicForm.CreateAsChildFromFile
Called by
TGLPiece.CreateAsChild

Inherited Constructors:

Constructors in TGLBasicForm:
[+] CreateAsChildFromFile

Inherited Constructors by Name:

[+] TGLBasicForm.CreateAsChildFromFile

Destructors:

[+] destructor Destroy; **override**
Destruye a FObjectivePosition y Hereda.
Overrides
Destroy
Overridden by
TGLPiece.Destroy
Called by
TGLPiece.Destroy

Methods:

[-] function GetElevated: boolean
Verifica si está elevado
Result
Si está elevado devuelve true; caso contrario, false

[-] function getInDisplacement: boolean
Compara las coordenadas de la FObjectivePosition con las actuales
Result
Devuelve true si las tres coordenadas de FObjectivePosition son iguales a la posición actual del Objeto.

[-] function getInRotation: boolean
Verifica si FRigthRotation o FLeftRotation están en true
Result
Devuelve true si cualquiera de FRigthRotation o FLeftRotation está en true.

[-] function getRotQuadrant: Integer
Verifica cual es el cuadrante actual de acuerdo al valor de la propiedad heredada RollAngle.
Result
Devuelve el valor del cuadrante que ocupa actualmente
Called by
TGLPiece.**Rotate**

[-] procedure setInLeftRotation(**const** Value: boolean)
Establece FLeftRotation y actualiza FLastQuadrant con el valor actual
Parameters
Value
Nuevo valor para FLeftRotation

[-] procedure setInRightRotation(**const** Value: boolean)
Establece FRightRotation y actualiza FLastQuadrant con el valor actual
Parameters
Value
Nuevo valor para FRightRotation

[-] procedure SetObjectivePosition_Z(**const** Value: single)
Establece el nuevo valor de elevación de la pieza
Parameters
Value
Nuevo valor de elevación de pieza
Called by
TGLPiece.**SetGLActionObjectives**

Inherited Methods:

Methods in TGLBasicForm:
[+] DoActions, **[+] getCoordFromParentObject**, **[+] showError**, **[+] showHelp**

Inherited Methods by Name:

[+] TGLBasicForm.**DoActions**, **[+] TGLBasicForm**.**getCoordFromParentObject**,
[+] TGLBasicForm.**showError**, **[+] TGLBasicForm**.**showHelp**

AIII.6.10 Class TGLLeftSide

TObject <-- ... <-- TGLActor <-- **TGLBasicForm** <-- **TGLSide** <-- **TGLLeftSide**
Declared in Unit **View**

No known Subclasses

Used by Classes:

TPieceSides

Description:

Lado Visual Izquierdo <A.C.: No Definida>
Forma Visual correspondiente al Lados izquierdos y Bordes derechos.
Used in
TMainForm.**GLSceneViewerMouseDown**, **TPieceSides**.**CreateAsChild**, **TPieceSides**

List of Constructors:

[+] constructor CreateAsChild
Hereda y rota el lado 90°

Fields:

Inherited Fields:

Fields in TGLSide:
[-] FSideColor
Fields in TGLBasicForm:
[-] FDeltaTime, **[-] FErrorAcumTime**, **[-] FErrorTime**, **[-] FHelpAcumTime**, **[-] FHelpTime**, **[-] FInError**, **[-] FInHelp**, **[-] FTempErrColor**, **[-] FTempHelpColor**

Inherited Fields by Name:

[-] TGLBasicForm.**FDeltaTime**, **[-] TGLBasicForm**.**FErrorAcumTime**, **[-] TGLBasicForm**.**FErrorTime**, **[-] TGLBasicForm**.**FHelpAcumTime**, **[-] TGLBasicForm**.**FHelpTime**, **[-] TGLBasicForm**.**FInError**, **[-] TGLBasicForm**.**FInHelp**, **[-] TGLSide**.**FSideColor**, **[-] TGLBasicForm**.**FTempErrColor**, **[-] TGLBasicForm**.**FTempHelpColor**

Properties:

Inherited Properties:

Properties in TGLSide:
[+] SideColor

Inherited Properties by Name:

[+] TGLSide.**SideColor**

Constructors:

[+] constructor CreateAsChild(aParentOwner: TGLBaseSceneObject)
Hereda y rota el lado 90°
Parameters
aParentOwner
Objeto Padre Visual
Called by
TPieceSides.**CreateAsChild**

Inherited Constructors:

Constructors in TGLSide:
[+] CreateAsChild

Constructors in TGLBasicForm:
[+] CreateAsChildFromFile

Inherited Constructors by Name:

[+] TGLSide.CreateAsChild, [+] TGLBasicForm.CreateAsChildFromFile

Methods:

Inherited Methods:

Methods in TGLSide:
[-] SetSideColor
Methods in TGLBasicForm:
[+] DoActions, [+] getCoordFromParentObject, [+] showError, [+] showHelp

Inherited Methods by Name:

[+] TGLBasicForm.DoActions, [+] TGLBasicForm.getCoordFromParentObject, [-] TGLSide.SetSideColor, [+] TGLBasicForm.showError, [+] TGLBasicForm.showHelp

AIII.6.11 Class TGLPiece

TObject <-- ... <-- TGLActor <-- **TGLBasicForm** <-- **TGLDynamicForm** <-- **TGLPiece**
Declared in Unit **View**

No known Subclasses

Used by Classes:

TGLPieces

Description:

Pieza Visual <A.C.: Pieza Visual>
Hereda de TGLDynamicForm todo el comportamiento referido a desplazamientos, y solo agrega atributos y operaciones que tengan que ver con su condición de Pieza Visual.
Used in

TGLPieces.Add, TGLPieces.Get, TMainForm.DoMovement, TMainForm.GLCadencerProgress, getGLClassName, TGLPieces

List of Constructors:

[+] constructor CreateAsChild
Invoca a CreateAsChildFromFile y Ubica la pieza visual en posición inicial

List of Destructors:

[+] destructor Destroy
Destruye a FGLStartPositionSquare y hereda

List of Methods:

[+] procedure doActions
Hereda y define la prosecución de los movimientos de acuerdo a la primera acción de la lista de acciones
[+] procedure doMovement
Permite agregar un movimiento a la lista de movimientos posibles
[+] procedure doMovement
Permite agregar un movimiento a la lista de movimientos posibles

[-] function getMoveEnded
Define que el movimiento ha finalizado.
[-] procedure Rotate
Realiza las rotaciones
[-] procedure SetGLActionObjectives
Establece los objetivos de cada acción
[-] procedure setPieceColor
Establece el color de la pieza el del casillero de posición inicial y asigna el material correspondiente.
[-] procedure setUnUsedPosition
Establece la posición inicial del casillero asociado y toma sus coordenadas como propias.
[-] procedure Translate
Realiza los traslados en ejes x, y y z

Fields:

[-] FGLActionToDoList: **TGLActionToDoList**
Used in
CreateAsChild, **Destroy**, **doActions**, **doMovement**, **doMovement**, **SetGLActionObjectives**, **Translate**

[-] FGLPieceSides: **TPieceSides**
Used in
CreateAsChild, **Destroy**, **TGLBoard**.**InitializePieces**

[-] FGLRotationArrows: **TGLRotationArrows**
Used in
doMovement, **SetGLActionObjectives**, **Translate**

[-] FGLStartPositionSquare: **TGLStartPositionSquare**
Indica sobre qué Casillero Visual de la zona del Jugador tomará su posición inicial. <A.C.: Pieza Visual :: Posición Inicial>
Used in
CreateAsChild, **Destroy**, **doMovement**, **setPieceColor**, **setUnUsedPosition**, **TGLPieces**.**Create**

[-] FInLastMove: boolean
Used in
CreateAsChild, **doMovement**

[-] FMoveEnded: boolean
Used in
CreateAsChild, **doActions**, **getMoveEnded**

[-] FPieceColor: **TPieceColor**
Se corresponde con el del jugador al que pertenece <A.C.: Pieza Visual :: Color>
Used in
setPieceColor

[-] FUnUsedPosition: integer
Número de orden de la pieza para el jugador dueño<A.C.: Pieza Visual :: No definido>
Used in
setUnUsedPosition

Inherited Fields:

Fields in TGLDynamicForm:
[-] FAscending, **[-] FDescending**, **[-] FDisplacementSpeed**, **[-] FInLeftRotation**, **[-] FInRightRotation**, **[-] FLastRotQuadrant**, **[-] FObjectivePosition**, **[-] FObjectiveRollAngle**, **[-] FRotationSpeed**
Fields in TGLBasicForm:
[-] FDeltaTime, **[-] FErrorAcumTime**, **[-] FErrorTime**, **[-] FHelpAcumTime**, **[-] FHelpTime**, **[-] FInError**, **[-] FInHelp**, **[-] FTempErrColor**, **[-] FTempHelpColor**

Inherited Fields by Name:

[-] TGLDynamicForm.FAscending, **[-] TGLBasicForm.FDeltaTime**, **[-] TGLDynamicForm.FDescending**, **[-] TGLDynamicForm.FDisplacementSpeed**, **[-] TGLBasicForm.FErrorAcumTime**, **[-] TGLBasicForm.FErrorTime**, **[-] TGLBasicForm.FHelpAcumTime**, **[-] TGLBasicForm.FHelpTime**, **[-] TGLBasicForm.FInError**, **[-] TGLBasicForm.FInHelp**, **[-] TGLDynamicForm.FInLeftRotation**, **[-] TGLDynamicForm.FInRightRotation**, **[-] TGLDynamicForm.FLastRotQuadrant**, **[-] TGLDynamicForm.FObjectivePosition**, **[-] TGLDynamicForm.FObjectiveRollAngle**, **[-] TGLDynamicForm.FRotationSpeed**, **[-] TGLBasicForm.FTempErrColor**, **[-] TGLBasicForm.FTempHelpColor**

Properties:

[+] property GLRotationArrows: **TGLRotationArrows read FGLRotationArrows write FGLRotationArrows**
Used in
TGLBoard.InitializePlayers

[+] property GLStartPositionSquare: **TGLStartPositionSquare read FGLStartPositionSquare write FGLStartPositionSquare**
Indica sobre qué Casillero Visual de la zona del Jugador tomará su posición inicial. <A.C.: Pieza Visual :: Posición Inicial>
Used in
TGLBoard.InitializePlayers

[+] [ro] property MoveEnded: boolean **read getMoveEnded**
Define que el movimiento ha finalizado. Al ser leida, se resetea (se pone el valor en false)
Used in
TMainForm.GLCadencerProgress

[+] property PieceColor: **TPieceColor read FPieceColor write setPieceColor**
Se corresponde con el del jugador al que pertenece <A.C.: Pieza Visual :: Color>
Used in
TGLBoard.InitializePlayers

[+] property UnUsedPosition: integer **read FUnUsedPosition write setUnUsedPosition**
Número de orden de la pieza para el jugador dueño<A.C.: Pieza Visual :: No definido>
Used in
TGLBoard.InitializePlayers

Inherited Properties:

Properties in TGLDynamicForm:
[+][ro] Elevated, **[+][ro] InDisplacement**, **[+] InLeftRotation**, **[+] InRightRotation**, **[+][ro] InRotation**, **[+] LastRotQuadrant**, **[+][ro] RotQuadrant**

Inherited Properties by Name:

[+][ro] TGLDynamicForm.Elevated, **[+][ro] TGLDynamicForm.InDisplacement**,
[+] TGLDynamicForm.InLeftRotation, **[+] TGLDynamicForm.InRightRotation**,
[+][ro] TGLDynamicForm.InRotation, **[+] TGLDynamicForm.LastRotQuadrant**,
[+][ro] TGLDynamicForm.RotQuadrant

Constructors:

[+] constructor CreateAsChild(aParentOwner: TGLBaseSceneObject)

Invoca a CreateAsChildFromFile y Ubica la pieza visual en posición inicial

Parameters

aParentOwner

Objeto Padre Visual

Called by

TGLPieces.Add

Inherited Constructors:

Constructors in TGLDynamicForm:
[+] CreateAsChildFromFile
Constructors in TGLBasicForm:
[+] CreateAsChildFromFile

Inherited Constructors by Name:

[+] TGLBasicForm.CreateAsChildFromFile, **[+] TGLDynamicForm.CreateAsChildFromFile**

Destructors:

[+] destructor Destroy; **override**

Destruye a FGLStartPositionSquare y hereda

Overrides

TGLDynamicForm.Destroy

Inherited Destructors:

Destructors in TGLDynamicForm:
[+] Destroy

Inherited Destructors by Name:

[+] TGLDynamicForm.Destroy

Methods:

[+] procedure doActions(aDeltaTime: single); **override**

Hereda y define la prosecución de los movimientos de acuerdo a la primera acción de la lista de acciones

Parameters

aDeltaTime

Variación del tiempo entre la escena inmediata anterior y la actual que proviene de OpenGL.

Overrides

TGLBasicForm.DoActions

[+] procedure doMovement(aMovement: **TMovement**); **overload**
Permite agregar un movimiento a la lista de movimientos posibles
Parameters
aMovement
Movimiento a realizar
Called by
doMovement, **TMainForm**.**DoMovement**

[+] procedure doMovement(aMovement: **TMovement**; aSquare: **TGLSquare**); **overload**
Permite agregar un movimiento a la lista de movimientos posibles
Parameters
aMovement
Movimiento a realizar
aSquare
Casillero sobre el cual se indica realizar el movimiento

[-] function getMoveEnded: boolean
Define que el movimiento ha finalizado. Al ser leida, se resetea (se pone el valor en false)
Result
Devuelve true si el movimiento finalizó

[-] procedure Rotate
Realiza las rotaciones
Called by
doActions

[-] procedure SetGLActionObjectives
Establece los objetivos de cada acción
Called by
doActions, **doMovement**, **doMovement**

[-] procedure setPieceColor(**const** Value: **TPieceColor**)
Establece el color de la pieza el del casillero de posición inicial y asigna el material correspondiente.
Parameters
Value
Color de pieza asignado

[-] procedure setUnUsedPosition(**const** Value: integer)
Establece la posición inicial del casillero asociado y toma sus coordenadas como propias.
Parameters
Value
Orden relativo de la pieza y el casillero visual de posición inicial asociado.

[-] procedure Translate
Realiza los traslados en ejes x, y y z
Called by
doActions

Inherited Methods:

Methods in TGLDynamicForm:
[-] GetElevated, **[-] getInDisplacement**, **[-] getInRotation**, **[-] getRotQuadrant**, **[-**

] setInLeftRotation, **[-] setInRightRotation**, **[-] SetObjectivePosition_Z**
Methods in TGLBasicForm:
[+] DoActions, **[+] getCoordFromParentObject**, **[+] showError**, **[+] showHelp**

Inherited Methods by Name:

[+] TGLBasicForm.DoActions, **[+] TGLBasicForm.getCoordFromParentObject**, **[-] TGLDynamicForm.GetElevated**, **[-] TGLDynamicForm.getInDisplacement**, **[-] TGLDynamicForm.getInRotation**, **[-] TGLDynamicForm.getRotQuadrant**, **[-] TGLDynamicForm.setInLeftRotation**, **[-] TGLDynamicForm.setInRightRotation**, **[-] TGLDynamicForm.SetObjectivePosition_Z**, **[+] TGLBasicForm.showError**, **[+] TGLBasicForm.showHelp**

AIII.6.12 Class TGLPieces

TObject <-- ... <-- TObjectList <-- **TGLPieces**
Declared in Unit **View**

No known Subclasses

Used by Classes:

TGLBoard

Description:

Piezas Visuales <A.C.: Piezas Visual>
Es el contenedor de cada uno de las 24 Piezas Visuales de Tablero Visual
Used in
TGLBoard.Create, **TGLBoard**

List of Constructors:

[+] constructor Create
Hereda y asigna el FOwner de las piezas Visuales

List of Methods:

[+] function Add
Crea un objeto de la clase TGLPiece y lo agrega al final de la lista
[#] function Get
Devuelva la pieza visual correspondiente
[#] procedure Put
Asigna una pieza visual en una posición dada

Fields:

[-] FOwner: TGLBaseSceneObject
Used in
Add, **Create**

Properties:

[+] property Items [Index: integer]: **TGLPiece read Get write Put**; **default**
Devuelva la pieza visual correspondiente

Constructors:

[+] constructor Create(aParentOwner: TGLBaseSceneObject)
Hereda y asigna el FOwner de las piezas Visuales

Parameters
aParentOwner
Objeto Padre Visual de los piezas visuales (Tablero Visual)
Called by
TGLBoard.**Create**

Methods:

[+] function Add: **TGLPiece**
Crea un objeto de la clase TGLPiece y lo agrega al final de la lista
Result
Objeto creado
Called by
Create

[#] function Get(Index: integer): **TGLPiece**
Devuelva la pieza visual correspondiente
Parameters
Index
Número de orden de la pieza visual requerido
Result
Referencia al Objeto pedido

[#] procedure Put(Index: integer; aItem: **TGLPiece**)
Asigna una pieza visual en una posición dada
Parameters
Index
Número de orden de la pieza visual a asignar
aItem
Objeto de la clase TGLPiece a asignar

AIII.6.13 Class TGLPlayFieldSquare

TObject <-- ... <-- TGLActor <-- **TGLBasicForm** <-- **TGLSquare** <-- **TGLPlayFieldSquare**
Declared in Unit **View**

No known Subclasses

Used by Classes:

TGLPlayFieldSquares

Description:

Casillero Visual de Juego <A.C.: Casillero Visual>
Casillero visual donde podrán jugarse piezas.
Used in
TGLPiece.**Translate**, **TGLPlayFieldSquares**.**Add**, **TGLPlayFieldSquares**.**Get**, **getGLClassName**, **TGLPlayFieldSquares**

List of Methods:

[+] procedure DoActions
Reimplementa DoActions.

[-] procedure SetPlayFieldPosition
Establece la posición y la adecúa al tablero <A.C.: Casillero Visual :: Establecer Posición>

Fields:

[-] FPlayFieldPosition: TPoint
Coordenadas del Casillero <A.C.: Casillero Visual :: Posición>
Used in
SetPlayFieldPosition

Inherited Fields:

Fields in TGLBasicForm:
[-] FDeltaTime, **[-] FErrorAcumTime**, **[-] FErrorTime**, **[-] FHelpAcumTime**, **[-] FHelpTime**, **[-] FInError**, **[-] FInHelp**, **[-] FTempErrColor**, **[-] FTempHelpColor**

Inherited Fields by Name:

[-] TGLBasicForm.FDeltaTime, **[-] TGLBasicForm.FErrorAcumTime**, **[-] TGLBasicForm.FErrorTime**, **[-] TGLBasicForm.FHelpAcumTime**, **[-] TGLBasicForm.FHelpTime**, **[-] TGLBasicForm.FInError**, **[-] TGLBasicForm.FInHelp**, **[-] TGLBasicForm.FTempErrColor**, **[-] TGLBasicForm.FTempHelpColor**

Properties:

[+] property PlayFieldPosition: TPoint **read FPlayFieldPosition write SetPlayFieldPosition**
Coordenadas del Casillero <A.C.: Casillero Visual :: Posición>
Used in
TGLBoard.InitializeSquares

Constructors:

Inherited Constructors:

Constructors in TGLSquare:
[+] CreateAsChild
Constructors in TGLBasicForm:
[+] CreateAsChildFromFile

Inherited Constructors by Name:

[+] TGLSquare.CreateAsChild, **[+] TGLBasicForm.CreateAsChildFromFile**

Methods:

[+] procedure DoActions(aDeltaTime: single); **override**
Reimplementa DoActions. Hereda y transmite la acción a los hijos (en particular a los indicadores de rotación).
Parameters
aDeltaTime
Variación del tiempo entre la escena inmediata anterior y la actual
Overrides
TGLBasicForm.DoActions

[-] procedure SetPlayFieldPosition(**const** Value: TPoint)
Establece la posición y la adecúa al tablero <A.C.: Casillero Visual :: Establecer Posición>
Parameters

Value
Posición en fila y columna

Inherited Methods:

Methods in TGLBasicForm:
[+] DoActions, **[+] getCoordFromParentObject**, **[+] showError**, **[+] showHelp**

Inherited Methods by Name:

[+] TGLBasicForm.DoActions, **[+] TGLBasicForm.getCoordFromParentObject**,
[+] TGLBasicForm.showError, **[+] TGLBasicForm.showHelp**

AIII.6.14 Class TGLPlayFieldSquares

TObject <-- ... <-- TObjectList <-- **TGLPlayFieldSquares**
Declared in Unit **View**

No known Subclasses

Used by Classes:

TGLBoard

Description:

Casilleros Visuales de Juego <A.C.: Casilleros Visual>
Es el contenedor de cada uno de los 24 Casilleros Visuales del Tablero Visual
Used in
TGLBoard.Create, **TGLBoard**

List of Constructors:

[+] constructor Create
Hereda y asigna el FOwner de los casilleros Visuales

List of Methods:

[+] function Add
Crea un objeto de la clase TGLSquare y lo agrega al final de la lista
[#] function Get
Devuelva el casillero visual correspondiente
[-] function GetCol
Devuelve la columna de un casillero
[-] function GetRow
Devuelve la fila de un casillero
[#] procedure Put
Asigna una pieza visual en una posición dada

Fields:

[-] FOwner: TGLBaseSceneObject
Used in
Add, **Create**

Properties:

[+] [ro] property Col [Index: integer]: integer **read GetCol**
Devuelve la columna de un casillero

Used in
TGLBoard.InitializeSquares

[+] property Items [Index: integer]: **TGLPlayFieldSquare read Get write Put**; **default**
Devuelva el casillero visual correspondiente
[+] [ro] property Row [Index: integer]: integer **read GetRow**
Devuelve la fila de un casillero
Used in
TGLBoard.InitializeSquares

Constructors:

[+] constructor Create(aParentOwner: TGLBaseSceneObject)
Hereda y asigna el FOwner de los casilleros Visuales
Parameters
aParentOwner
Objeto Padre Visual de los casilleros (Tablero Visual)
Called by
TGLBoard.Create

Methods:

[+] function Add: **TGLPlayFieldSquare**
Crea un objeto de la clase TGLSquare y lo agrega al final de la lista
Result
Objeto creado
Called by
Create

[#] function Get(Index: integer): **TGLPlayFieldSquare**
Devuelva el casillero visual correspondiente
Parameters
Index
Número de orden del casillero visual requerido
Result
Referencia al Objeto pedido

[-] function GetCol(Index: integer): integer
Devuelve la columna de un casillero
Parameters
Index
Índice del Casillero
Result
Columna del casillero

[-] function GetRow(Index: integer): integer
Devuelve la fila de un casillero
Parameters
Index
Índice del Casillero
Result
Fila del casillero

[#] procedure Put(Index: integer; aItem: **TGLPlayFieldSquare**)
Asigna una pieza visual en una posición dada
Parameters
Index
Número de orden del borde visual a asignar
aItem
Objeto de la clase TGLPiece a asignar

AIII.6.15 Class TGLRightSide

TObject <-- ... <-- TGLActor <-- **TGLBasicForm** <-- **TGLSide** <-- **TGLRightSide**
Declared in Unit **View**

No known Subclasses

Used by Classes:

TPieceSides

Description:

Lado Visual Derecho <A.C.: No Definida>
Forma Visual correspondiente al Lados derechos y Bordes izquierdos.
Used in
TMainForm.**GLSceneViewerMouseDown**, **TPieceSides**.**CreateAsChild**, **TPieceSides**

List of Constructors:

[+] constructor CreateAsChild
Hereda y rota el lado 180°

Fields:

Inherited Fields:

Fields in TGLSide:
[-] FSideColor
Fields in TGLBasicForm:
[-] FDeltaTime, **[-] FErrorAcumTime**, **[-] FErrorTime**, **[-] FHelpAcumTime**, **[-] FHelpTime**, **[-] FInError**, **[-] FInHelp**, **[-] FTempErrColor**, **[-] FTempHelpColor**

Inherited Fields by Name:

[-] TGLBasicForm.**FDeltaTime**, **[-] TGLBasicForm**.**FErrorAcumTime**, **[-] TGLBasicForm**.**FErrorTime**, **[-] TGLBasicForm**.**FHelpAcumTime**, **[-] TGLBasicForm**.**FHelpTime**, **[-] TGLBasicForm**.**FInError**, **[-] TGLBasicForm**.**FInHelp**, **[-] TGLSide**.**FSideColor**, **[-] TGLBasicForm**.**FTempErrColor**, **[-] TGLBasicForm**.**FTempHelpColor**

Properties:

Inherited Properties:

Properties in TGLSide:
[+] SideColor

Inherited Properties by Name:

[+] TGLSide.**SideColor**

Constructors:

[+] constructor CreateAsChild(aParentOwner: TGLBaseSceneObject)
Hereda y rota el lado 180°
Parameters
aParentOwner
Objeto Padre Visual
Called by
TPieceSides.**CreateAsChild**

Inherited Constructors:

Constructors in TGLSide:
[+] CreateAsChild
Constructors in TGLBasicForm:
[+] CreateAsChildFromFile

Inherited Constructors by Name:

[+] TGLSide.**CreateAsChild**, **[+] TGLBasicForm**.**CreateAsChildFromFile**

Methods:

Inherited Methods:

Methods in TGLSide:
[-] SetSideColor
Methods in TGLBasicForm:
[+] DoActions, **[+] getCoordFromParentObject**, **[+] showError**, **[+] showHelp**

Inherited Methods by Name:

[+] TGLBasicForm.**DoActions**, **[+] TGLBasicForm**.**getCoordFromParentObject**, **[-] TGLSide**.**SetSideColor**, **[+] TGLBasicForm**.**showError**, **[+] TGLBasicForm**.**showHelp**

AIII.6.16 Class TGLRotationArrows

TObject <-- ... <-- TGLActor <-- **TGLBasicForm** <-- **TGLRotationArrows**
Declared in Unit **View**

No known Subclasses

Used by Classes:

TGLPiece, **TGLBoard**

Description:

Flechas de Rotación <A.C. No Definida> Agrupa en una sola clase las dos flechas indicadoras de rotación.
Used in
TGLBoard.**Create**, **TMainForm**.**getGLObjectByName**, **TGLBoard**, **TGLPiece**

List of Constructors:

[+] constructor CreateAsChild
Invoca a CreateAsChildFromFiles

List of Destructors:

[+] destructor Destroy
Destruye ambas flechas y hereda

List of Methods:

[+] procedure ChangeOrientation
Rota las flechas y reasigna nombres a cada una de ellas.
[#] procedure CreateAsChildFromFiles
Hereda de CreateAsChildFromFiles para crear el centro, crea ambas flechas con Owner el Centro y establece posiciones iniciales
[+] procedure DoActions
hereda y transmite el DoActions a las flechas
[+] procedure SetNames
Establece el nombre

Fields:

[-] FGLTurnLeftArrow: **TGLTurnLeftArrow**
Flecha Indicadora de rotación hacia la Izquierda
Used in
ChangeOrientation, **CreateAsChildFromFiles**, **Destroy**, **DoActions**, **SetNames**

[-] FGLTurnRightArrow: **TGLTurnRightArrow**
Flecha Indicadora de rotación hacia la Derecha
Used in
ChangeOrientation, **CreateAsChildFromFiles**, **Destroy**, **DoActions**, **SetNames**

Inherited Fields:

Fields in TGLBasicForm:
[-] FDeltaTime, **[-] FErrorAcumTime**, **[-] FErrorTime**, **[-] FHelpAcumTime**, **[-] FHelpTime**, **[-] FInError**, **[-] FInHelp**, **[-] FTempErrColor**, **[-] FTempHelpColor**

Inherited Fields by Name:

[-] TGLBasicForm.FDeltaTime, **[-] TGLBasicForm.FErrorAcumTime**, **[-] TGLBasicForm.FErrorTime**, **[-] TGLBasicForm.FHelpAcumTime**, **[-] TGLBasicForm.FHelpTime**, **[-] TGLBasicForm.FInError**, **[-] TGLBasicForm.FInHelp**, **[-] TGLBasicForm.FTempErrColor**, **[-] TGLBasicForm.FTempHelpColor**

Properties:

[+] [ro] property GLTurnLeftArrow: **TGLTurnLeftArrow read FGLTurnLeftArrow**
Flecha Indicadora de rotación hacia la Izquierda
[+] [ro] property GLTurnRightArrow: **TGLTurnRightArrow read FGLTurnRightArrow**
Flecha Indicadora de rotación hacia la Derecha

Constructors:

[+] constructor CreateAsChild(aParentOwner: TGLBaseSceneObject)
Invoca a CreateAsChildFromFiles
Parameters
aParentOwner
Objeto Padre Visual (las posiciones se tomarán sobre él. Los movimientos del Padre Visual, arrastran al de los hijos).
Called by
TGLBoard.Create

Inherited Constructors:

Constructors in TGLBasicForm:
[+] CreateAsChildFromFile

Inherited Constructors by Name:

[+] TGLBasicForm.CreateAsChildFromFile

Destructors:

[+] destructor Destroy; **override**
Destruye ambas flechas y hereda
Overrides
Destroy

Methods:

[+] procedure ChangeOrientation
Rota las flechas y reasigna nombres a cada una de ellas.
Called by
TGLBoard.Create

[#] procedure CreateAsChildFromFiles(aParentOwner: TGLBaseSceneObject; aCenterFileName, aTurnRightArrowFileName, aTurnLeftArrowFileName: String)
Hereda de CreateAsChildFromFiles para crear el centro, crea ambas flechas con Owner el Centro y establece posiciones iniciales
Parameters
aParentOwner
Objeto Padre Visual (las posiciones se tomarán sobre él. Los movimientos del Padre Visual, arrastran al de los hijos)
aCenterFileName
Nombre y Path completo del archivo 3DS del centro (una mínima esfera)
aTurnRightArrowFileName
Nombre y Path completo del archivo 3DS de la flecha para rotar hacia la derecha
aTurnLeftArrowFileName
Nombre y Path completo del archivo 3DS de la flecha para rotar hacia la izquierda
Called by
CreateAsChild

[+] procedure DoActions(aDeltaTime: Single); **override**
hereda y transmite el DoActions a las flechas
Parameters
aDeltaTime
Variación del tiempo entre la escena inmediata anterior y la actual
Overrides
TGLBasicForm.DoActions

[+] procedure SetNames(id: integer)
Establece el nombre
Parameters
id
vale 1 si son los indicadores de rotación de las piezas negras y 0 si lo son de las blancas
Called by
TGLBoard.Create

Inherited Methods:

Methods in TGLBasicForm:
[+] DoActions, **[+] getCoordFromParentObject**, **[+] showError**, **[+] showHelp**

Inherited Methods by Name:

[+] TGLBasicForm.DoActions, **[+] TGLBasicForm.getCoordFromParentObject**,
[+] TGLBasicForm.showError, **[+] TGLBasicForm.showHelp**

AIII.6.17 Class TGLSide

TObject <-- ... <-- TGLActor <-- **TGLBasicForm** <-- **TGLSide**
Declared in Unit **View**

Direct known Subclasses:

TGlBorder, **TGLBottomSide**, **TGLLeftSide**, **TGLRightSide**, **TGLTopSide**

Not used by any known Class

Description:

Lado Visual Genérico <A.C.: No Definida>
Forma Visual correspondiente al Lado y al Borde.

List of Constructors:

[+] constructor CreateAsChild
Crea al Objeto a partir del archivo 3DS asociado a los los lados y bordes cualquiera sea su posición.

List of Methods:

[-] procedure SetSideColor
Establece el color y el resto de las propiedades del material

Fields:

[-] FSideColor: **TSideColor**
Color asociado al lado o al Borde, según corresponda
Used in
SetSideColor

Inherited Fields:

Fields in TGLBasicForm:
[-] FDeltaTime, **[-] FErrorAcumTime**, **[-] FErrorTime**, **[-] FHelpAcumTime**, **[-] FHelpTime**, **[-] FInError**, **[-] FInHelp**, **[-] FTempErrColor**, **[-] FTempHelpColor**

Inherited Fields by Name:

[-] TGLBasicForm.FDeltaTime, **[-] TGLBasicForm.FErrorAcumTime**, **[-] TGLBasicForm.FErrorTime**, **[-] TGLBasicForm.FHelpAcumTime**, **[-] TGLBasicForm.FHelpTime**, **[-] TGLBasicForm.FInError**, **[-] TGLBasicForm.FInHelp**, **[-] TGLBasicForm.FTempErrColor**, **[-] TGLBasicForm.FTempHelpColor**

Properties:

[+] property SideColor: **TSideColor read FSideColor write SetSideColor**
Color asociado al lado o al Borde, según corresponda

Used in
TGLBoard.**InitializePieces**, **TGLBoard**.**InitializeSquares**

Constructors:

[+] constructor CreateAsChild(aParentOwner: TGLBaseSceneObject)
Crea al Objeto a partir del archivo 3DS asociado a los los lados y bordes cualquiera sea su posición.
Parameters
aParentOwner
Objeto Padre Visual
Called by
TGLBottomSide.**CreateAsChild**, **TGLLeftSide**.**CreateAsChild**, **TGLRightSide**.**CreateAsChild**, **TGLTopSide**.**CreateAsChild**

Inherited Constructors:

Constructors in TGLBasicForm:
[+] CreateAsChildFromFile

Inherited Constructors by Name:

[+] TGLBasicForm.**CreateAsChildFromFile**

Methods:

[-] procedure SetSideColor(**const** Value: **TSideColor**)
Establece el color y el resto de las propiedades del material
Parameters
Value
Color Asignado

Inherited Methods:

Methods in TGLBasicForm:
[+] DoActions, **[+] getCoordFromParentObject**, **[+] showError**, **[+] showHelp**

Inherited Methods by Name:

[+] TGLBasicForm.**DoActions**, **[+] TGLBasicForm**.**getCoordFromParentObject**, **[+] TGLBasicForm**.**showError**, **[+] TGLBasicForm**.**showHelp**

AIII.6.18 Class TGLSquare

TObject <-- ... <-- TGLActor <-- **TGLBasicForm** <-- **TGLSquare**
Declared in Unit **View**

Direct known Subclasses:

TGLPlayFieldSquare, **TGLStartPositionSquare**

Used by Classes:

TGLActionToDo

Description:

Casillero Visual <A.C.: No Definida>
Clase Virtual que define al casillero visual que será heredado por el Casillero Visual de Juego (TGLPlayingFieldSquare) y por el Casillero de Posición Inicial (TGLStartPositionSquare)
Used in

TMainForm.**DoMovement**, **TGLActionToDo**

List of Constructors:

[+] constructor CreateAsChild

Hereda y crea el Casillero básico a partir del archivo 3DS.

Fields:

Inherited Fields:

Fields in TGLBasicForm:
[-] FDeltaTime, **[-] FErrorAcumTime**, **[-] FErrorTime**, **[-] FHelpAcumTime**, **[-] FHelpTime**, **[-] FInError**, **[-] FInHelp**, **[-] FTempErrColor**, **[-] FTempHelpColor**

Inherited Fields by Name:

[-] TGLBasicForm.**FDeltaTime**, **[-] TGLBasicForm**.**FErrorAcumTime**, **[-] TGLBasicForm**.**FErrorTime**, **[-] TGLBasicForm**.**FHelpAcumTime**, **[-] TGLBasicForm**.**FHelpTime**, **[-] TGLBasicForm**.**FInError**, **[-] TGLBasicForm**.**FInHelp**, **[-] TGLBasicForm**.**FTempErrColor**, **[-] TGLBasicForm**.**FTempHelpColor**

Constructors:

[+] constructor CreateAsChild(aParentOwner: TGLBaseSceneObject)

Hereda y crea el Casillero básico a partir del archivo 3DS. Establece las propiedades visuales del casillero
Parameters
aParentOwner
Objeto Padre Visual
Called by
TGLPiece.**CreateAsChild**, **TGLPlayFieldSquares**.**Add**

Inherited Constructors:

Constructors in TGLBasicForm:
[+] CreateAsChildFromFile

Inherited Constructors by Name:

[+] TGLBasicForm.**CreateAsChildFromFile**

Methods:

Inherited Methods:

Methods in TGLBasicForm:
[+] DoActions, **[+] getCoordFromParentObject**, **[+] showError**, **[+] showHelp**

Inherited Methods by Name:

[+] TGLBasicForm.**DoActions**, **[+] TGLBasicForm**.**getCoordFromParentObject**, **[+] TGLBasicForm**.**showError**, **[+] TGLBasicForm**.**showHelp**

AIII.6.19 Class TGLStartPositionSquare

TObject <-- ... <-- TGLActor <-- **TGLBasicForm** <-- **TGLSquare** <-- **TGLStartPositionSquare**
Declared in Unit **View**

No known Subclasses

Used by Classes:

TGLPiece

Description:

Casillero Visual de Posición Inicial <A.C.: Casillero Visual de Posición Inicial>
Clase que define al casillero visual que a uno y otro lado del tablero permitirán que cada piezas se muestren apoyadas sobre cada uno de ellos.
Used in

TGLPiece.**CreateAsChild**, **getGLClassName**, **TGLPiece**

List of Methods:

[-] procedure setPieceColor
Establece el color del casillero de acuerdo al color de pieza correspondiente
[-] procedure setStartPosition
Establece la posición relativa para luego poder establecerse las coordenadas visuales correspondientes

Fields:

[-] FPieceColor: **TPieceColor**
Color de la Pieza. El color del casillero inicial será acorde al mismo. <A.C.: Casillero Visual de Posición Inicial :: Color>
Used in
setPieceColor, **setStartPosition**

[-] FStartPosition: integer
Posición relativa (numerada del 1 al 12 para cada color de piezas) <A.C.: Casillero Visual de Posición Inicial :: Posición>

Inherited Fields:

Fields in TGLBasicForm:
[-] FDeltaTime, **[-] FErrorAcumTime**, **[-] FErrorTime**, **[-] FHelpAcumTime**, **[-] FHelpTime**, **[-] FInError**, **[-] FInHelp**, **[-] FTempErrColor**, **[-] FTempHelpColor**

Inherited Fields by Name:

[-] TGLBasicForm.**FDeltaTime**, **[-] TGLBasicForm**.**FErrorAcumTime**, **[-] TGLBasicForm**.**FErrorTime**, **[-] TGLBasicForm**.**FHelpAcumTime**, **[-] TGLBasicForm**.**FHelpTime**, **[-] TGLBasicForm**.**FInError**, **[-] TGLBasicForm**.**FInHelp**, **[-] TGLBasicForm**.**FTempErrColor**, **[-] TGLBasicForm**.**FTempHelpColor**

Properties:

[+] property PieceColor: **TPieceColor read FPieceColor write setPieceColor**
Color de la Pieza. El color del casillero inicial será acorde al mismo. <A.C.: Casillero Visual de Posición Inicial :: Color>
Used in
TGLPiece.**setPieceColor**

[+] property StartPosition: integer **read FStartPosition write setStartPosition**

Posición relativa (numerada del 1 al 12 para cada color de piezas) <A.C.: Casillero Visual de Posición Inicial :: Posición>

Used in

TGLBoard.**InitializePlayers**, **TGLPiece**.**setUnUsedPosition**

Constructors:

Inherited Constructors:

Constructors in TGLSquare:
[+] CreateAsChild
Constructors in TGLBasicForm:
[+] CreateAsChildFromFile

Inherited Constructors by Name:

[+] TGLSquare.**CreateAsChild**, **[+] TGLBasicForm**.**CreateAsChildFromFile**

Methods:

[-] procedure setPieceColor(**const** Value: **TPieceColor**)

Establece el color del casillero de acuerdo al color de pieza correspondiente

Parameters

Value

Color de la Pieza asociada

[-] procedure setStartPosition(**const** Value: integer)

Establece la posición relativa para luego poder establecerse las coordenadas visuales correspondientes

Parameters

Value

Posición Relativa de Posición Inicial

Inherited Methods:

Methods in TGLBasicForm:
[+] DoActions, **[+] getCoordFromParentObject**, **[+] showError**, **[+] showHelp**

Inherited Methods by Name:

[+] TGLBasicForm.**DoActions**, **[+] TGLBasicForm**.**getCoordFromParentObject**, **[+] TGLBasicForm**.**showError**, **[+] TGLBasicForm**.**showHelp**

AIII.6.20 Class TGLTopSide

TObject <-- ... <-- TGLActor <-- **TGLBasicForm** <-- **TGLSide** <-- **TGLTopSide**
Declared in Unit **View**

No known Subclasses

Used by Classes:

TPieceSides

Description:

Lado Visual Superior <A.C.: No Definida>
Forma Visual correspondiente a Lados superiores y a Bordes inferiores.
Used in
TMainForm.**GLSceneViewerMouseDown**, **TPieceSides**.**CreateAsChild**, **TPieceSides**

List of Constructors:

[+] constructor CreateAsChild
TGLTopSide Hereda y rota el lado 90°

Fields:

Inherited Fields:

Fields in TGLSide:
[-] FSideColor
Fields in TGLBasicForm:
[-] FDeltaTime, **[-] FErrorAcumTime**, **[-] FErrorTime**, **[-] FHelpAcumTime**, **[-] FHelpTime**, **[-] FInError**, **[-] FInHelp**, **[-] FTempErrColor**, **[-] FTempHelpColor**

Inherited Fields by Name:

[-] TGLBasicForm.**FDeltaTime**, **[-] TGLBasicForm**.**FErrorAcumTime**, **[-] TGLBasicForm**.**FErrorTime**, **[-] TGLBasicForm**.**FHelpAcumTime**, **[-] TGLBasicForm**.**FHelpTime**, **[-] TGLBasicForm**.**FInError**, **[-] TGLBasicForm**.**FInHelp**, **[-] TGLSide**.**FSideColor**, **[-] TGLBasicForm**.**FTempErrColor**, **[-] TGLBasicForm**.**FTempHelpColor**

Properties:

Inherited Properties:

Properties in TGLSide:
[+] SideColor

Inherited Properties by Name:

[+] TGLSide.**SideColor**

Constructors:

[+] constructor CreateAsChild(aParentOwner: TGLBaseSceneObject)
TGLTopSide Hereda y rota el lado 90°
Parameters
aParentOwner
Objeto Padre Visual
Called by
TPieceSides.**CreateAsChild**

Inherited Constructors:

Constructors in TGLSide:
[+] CreateAsChild
Constructors in TGLBasicForm:
[+] CreateAsChildFromFile

Inherited Constructors by Name:

[+] TGLSide.CreateAsChild, [+] TGLBasicForm.CreateAsChildFromFile

Methods:

Inherited Methods:

Methods in TGLSide:
[-] SetSideColor
Methods in TGLBasicForm:
[+] DoActions, [+] getCoordFromParentObject, [+] showError, [+] showHelp

Inherited Methods by Name:

[+] TGLBasicForm.DoActions, [+] TGLBasicForm.getCoordFromParentObject, [-] TGLSide.SetSideColor, [+] TGLBasicForm.showError, [+] TGLBasicForm.showHelp

AIII.6.21 Class TGLTurnLeftArrow

TObject <-- ... <-- TGLActor <-- **TGLBasicForm** <-- **TGLTurnLeftArrow**
Declared in Unit **View**

No known Subclasses

Used by Classes:

TGLRotationArrows

Description:

Flecha Indicadora de rotación hacia la Izquierda <A.C. Indicador de rotación hacia la izquierda>
Permite al usuario rotar la pieza seleccionada en el sentido contrario al de las agujas del reloj.
Used in

TGLRotationArrows.CreateAsChildFromFiles, TMainForm.ShowHelp, getGLClassName, TGLRotationArrows

Fields:

Inherited Fields:

Fields in TGLBasicForm:
[-] FDeltaTime, [-] FErrorAcumTime, [-] FErrorTime, [-] FHelpAcumTime, [-] FHelpTime, [-] FInError, [-] FInHelp, [-] FTempErrColor, [-] FTempHelpColor

Inherited Fields by Name:

[-] TGLBasicForm.FDeltaTime, [-] TGLBasicForm.FErrorAcumTime, [-] TGLBasicForm.FErrorTime, [-] TGLBasicForm.FHelpAcumTime, [-] TGLBasicForm.FHelpTime, [-] TGLBasicForm.FInError, [-] TGLBasicForm.FInHelp, [-] TGLBasicForm.FTempErrColor, [-] TGLBasicForm.FTempHelpColor

Constructors:

Inherited Constructors:

Constructors in TGLBasicForm:
[+] CreateAsChildFromFile

Inherited Constructors by Name:

[+] TGLBasicForm.CreateAsChildFromFile

Methods:

Inherited Methods:

Methods in TGLBasicForm:
[+] DoActions, **[+] getCoordFromParentObject**, **[+] showError**, **[+] showHelp**

Inherited Methods by Name:

[+] TGLBasicForm.DoActions, **[+] TGLBasicForm.getCoordFromParentObject**,
[+] TGLBasicForm.showError, **[+] TGLBasicForm.showHelp**

AIII.6.22 Class TGLTurnRightArrow

TObject <-- ... <-- TGLActor <-- **TGLBasicForm** <-- **TGLTurnRightArrow**
Declared in Unit **View**

No known Subclasses

Used by Classes:

TGLRotationArrows

Description:

Flecha Indicadora de rotación hacia la Derecha <A.C. Indicador de rotación hacia la derecha>
Permite al usuario rotar la pieza seleccionada en el sentido de las agujas del reloj.
Used in

TGLRotationArrows.CreateAsChildFromFiles, **TMainForm.ShowHelp**, **getGLClassName**, **TGLRotationArrows**

Fields:

Inherited Fields:

Fields in TGLBasicForm:
[-] FDeltaTime, **[-] FErrorAcumTime**, **[-] FErrorTime**, **[-] FHelpAcumTime**, **[-] FHelpTime**, **[-] FInError**, **[-] FInHelp**, **[-] FTempErrColor**, **[-] FTempHelpColor**

Inherited Fields by Name:

[-] TGLBasicForm.FDeltaTime, **[-] TGLBasicForm.FErrorAcumTime**, **[-] TGLBasicForm.FErrorTime**, **[-] TGLBasicForm.FHelpAcumTime**, **[-] TGLBasicForm.FHelpTime**, **[-] TGLBasicForm.FInError**, **[-] TGLBasicForm.FInHelp**, **[-] TGLBasicForm.FTempErrColor**, **[-] TGLBasicForm.FTempHelpColor**

Constructors:

Inherited Constructors:

Constructors in TGLBasicForm:
[+] CreateAsChildFromFile

Inherited Constructors by Name:

[+] TGLBasicForm.CreateAsChildFromFile

Methods:

Inherited Methods:

Methods in TGLBasicForm:
[+] DoActions, **[+] getCoordFromParentObject**, **[+] showError**, **[+] showHelp**

Inherited Methods by Name:

[+] TGLBasicForm.DoActions, **[+] TGLBasicForm.getCoordFromParentObject**,
[+] TGLBasicForm.showError, **[+] TGLBasicForm.showHelp**

AIII.6.23 Class TPieceSides

TObject <-- **TPieceSides**
Declared in Unit **View**

No known Subclasses

Used by Classes:

TGLPiece

Description:

Lados de Pieza Visual <A.C.: Lados de Pieza Visual>
Clase que hereda de TSides y que agrega los respectivos lados visuales que contendrá la pieza. Los valores asignados a cada lado, se van actualizando con cada rotación.
Used in
TGLPiece.CreateAsChild, **TGLPiece**

List of Constructors:

[+] constructor CreateAsChild
Hereda y crea a cada uno de los cuatro TGlSide

List of Destructors:

[+] destructor Destroy

Fields:

[-] FGLBottomSide: **TGLBottomSide**
Lado de Pieza Visual correspondiente a la posición FBottom de una pieza en su orientación actual. Arriba Visual <A.C.: Lados de Pieza Visual :: Abajo Visual>
Used in
CreateAsChild, **Destroy**, **TGLBoard.InitializePieces**

[-] FGLLeftSide: **TGLLeftSide**
Lado de Pieza Visual correspondiente a la posición FLeft de una pieza en su orientación actual. Arriba Visual <A.C.: Lados de Pieza Visual :: Izquierda Visual>
Used in
CreateAsChild, **Destroy**, **TGLBoard.InitializePieces**

[-] FGLRightSide: **TGLRightSide**
Lado de Pieza Visual correspondiente a la posición FRight de una pieza en su orientación actual. Arriba Visual <A.C.: Lados de Pieza Visual :: Derecha Visual>
Used in
CreateAsChild, **Destroy**, **TGLBoard.InitializePieces**

[-] FGLTopSide: **TGLTopSide**

Lado de Pieza Visual correspondiente a la posición FTop de una pieza en su orientación actual. Arriba Visual <A.C.: Lados de Pieza Visual :: Arriba Visual>

Used in

CreateAsChild, **Destroy**, **TGLBoard**.**InitializePieces**

Constructors:

[+] constructor CreateAsChild(aParentOwner: TGLBaseSceneObject)

Hereda y crea a cada uno de los cuatro TGlSide

Parameters

aParentOwner

Objeto Padre Visual (Pieza Visual)

Called by

TGLPiece.**CreateAsChild**

Destructors:

[+] destructor Destroy; **override**

Overrides

Destroy

Anexo IV – Bibliografía y Software

Bibliografía

- Cana López, D., 1996, SOALE, Sistema para el aprendizaje de la lectura – escritura para niños con SD (http://www.geocities.com/Athens/Atrium/5189/)

- Chapman, R., Hesketh. L, 2000, Fenotipo Conductal de las personas con síndrome de Down. Universidad de Wisconsin-Madison – Traducción Fundación Síndrome de Down de Cantabria (http://empresas.mundivia.es /downcan/fenotipo.html)

- García Escamilla, Sylvia, 1983, El niño con síndrome de Down. (Ed. Diana)

- García Trujillo, C, 2001, Programación con Delphi y OpenGl (www.elrinconcito.com/delphi)

- Gardner H., 1996, Inteligencia – Múltiples perspectivas, traducción al español 2000 (Ed. Aique)

- González C, 2000, Sistema tutorial inteligente para la enseñanza en niños con dificultades intelectuales y cognitivas, Univ. La Laguna (http://sid.usal.es/mostrarficha.asp_Q_ID_E_4786_A_fichero_E_8.4.3)

- Gorriz, B., 2001, Inteligencias Múltiples (http://www.ilustrados.com/publicaciones /EpZlypuFAlEZqBJhxP.php)

- IEEE Std 830-1993 IEEE Recommended Practice for Software Requirements Specifications (http://standards.ieee.org/reading/ieee/std_public/description/se/830-1993_desc.html)

- J. Schmuller, 2001, Aprendiendo UML en 24 horas. Ed. Prentice Hall

- Métrica V3, 2000., Metodología de Planificación, Desarrollo y Mantenimiento de Sistemas de información. Ministerio de Administraciones Públicas Español. http://www.csi.map.es/csi/metrica3/index.html. 2000.

- Mosston, Muska, and Sara Ashworth. 1990.*The spectrum of teaching styles from command to discovery.* White Plains, NY: Longman. 327 pages. 0801303508. Location: Dallas SIL Library 371.1 M914. Interest level: specialist. http://www.uned.ac.cr/servicios/global/ensenanza/instruccion/articulos/sistemas.htm)

- NewComb, R., Enseñanza del lenguaje asistida por computador, web del Laboratorio de Comunicación Oral de la Facultad de Informática de la UPM

- Project EPITOME. Engineering Platform for Intelligent Tutors and Multimedia Experiences. Nelson C. Baker, Ph.D., Project Director. Descripción del proyecto

dirigido a crear y evaluar software educacional para las áreas de ingeniería y ciencias http://www.ce.gatech.edu/~nbaker /abstracts/epitome.html. 2000.

- Toledo Ma., Kirschning I., 2001, Icatiani: Un Sistema de Apoyo para la Adquisición del Lenguaje, Universidad de las Americas, Puebla, Mexico (UDLAP) http://mailweb.udlap.mx/~ingrid/ingrid/Icatiani1.pdf.

- Troncoso, Ma. V., Del Cerro, M. M., 1998, Síndrome de Down: Lectura y Escritura. Ed. Mason – Barcelona.

Software Utilizado y Referido

- Biblioteca OpenGL para Delphi: GLScene. http://glscene.sourceforge.net /index.php. 06/01/2005.

- Entorno de desarrollo evaluado: Lazarus FreePascal Project (http://www.lazarus.freepascal.org)

- Entorno de desarrollo utilizado: Borland Delphi (http://www.borland.com)

- Herramienta para el diseño 3D evaluada: MilkShape (http://www.milkshape3d.com)

- Herramienta para Realización de Diagramas en el Proceso de Análisis y Diseño de la Arquitectura: Enterprise Architect (http://sparxsystems.com.au).

- Herramienta utilizada en el diseño de objetos 3D: Wings. (http://www.wings3d.com)

- Herramienta utilizada para Documentación del Diseño de Clases: Delphi Doc (JADD - Just Another DelphiDoc) (http://delphidoc.sourceforge.net)

- Herramienta Utilizada para la obtención de Métricas del código Fuente: CodeLens (http://sourceforge.net/projects/codelens)

- Instalación de Herramientas de fuentes abiertas que incluye Lazarus y GLScene: SkinHat (http://www.skinhat.com/3dpack).