PROYECTO FINAL DE CARRERA

Tecnología virtual interactiva de simulación de un estudio de broncoscopía

---

Autoras:

Paulina DAGNINO Victoria ROCCA

Tutores:

Santiago BESTANI Santiago RACCA

Departamento de Bioingeniería
Escuela de Ingeniería y Tecnología

20 de mayo de 2019

INSTITUTO TECNOLÓGICO DE BUENOS AIRES

Escuela de Ingeniería y Tecnología

Departamento de Bioingeniería

Tecnología virtual interactiva de simulación de un estudio de broncoscopía

por Paulina DAGNINO y Victoria ROCCA

El siguiente Proyecto Final de Grado se circunscribe en la temática del entrenamiento mediante simulación virtual del estudio de broncoscopía. Fue realizado en conjunto con la empresa mendocina Sensus3D, siendo los tutores del siguiente trabajo Santiago Bestani y Santiago Racca. La broncoscopía es un procedimiento terapéutico y diagnóstico que consiste en la visualización y tratamiento de las vías respiratorias utilizando un broncoscopio. La simulación es una herramienta de aprendizaje y práctica del procedimiento en ausencia de paciente real, y se diseña para que el usuario entrene habilidades técnicas en cuanto al uso de un broncoscopio, y también incorpore conocimientos de la anatomía de la vía aérea. Un simulador se compone de un software y un dispositivo hardware el cual equivale al que es utilizado en un equipo médico real, junto con un sensor que transduce el movimiento del tubo de inserción al movimiento virtual. En el presente proyecto se desarrolla una tecnología virtual interactiva, la cual consiste en la construcción de un escenario virtual enmarcado en un software que permite al usuario comprender la anatomía del sistema respiratorio, el aspecto característico de cierta patología y el comportamiento de las estructuras anatómicas en reacción a la interacción con el broncoscopio y sus funcionalidades. El manejo del movimiento del instrumento se desarrolla, en primera instancia, en control por parte del usuario mediante el teclado de una computadora, con el fin de permitir a futuro que el software se integre a un hardware con devolución háptica. Es por ello que el propósito inicial es exploratorio e instructivo, al recorrer e interactuar con la anatomía de la vía aérea en diversos escenarios, mediante la gamificación (la aplicación de juegos en contextos que no lo son). Si a futuro se complementa con un hardware, ofrecerá la posibilidad de desarrollar destrezas, coordinación y una adecuada manipulación del dispositivo.

Agradecimientos

Muchas personas contribuyeron con su experiencia y conocimiento, a que el siguiente Trabajo Final de Grado se lleve a cabo:

- **Dra. Elena Almar** (Hospital General de Agudos Bernardino Rivadavia, Av. Gral. Las Heras 2670, CABA, Argentina): Nos introdujo en el mundo de la endoscopía y tuvo la gentileza de enseñarnos diversos procedimientos que realiza en su labor diaria.
- **SIMMER - Centro de Simulación Médica Roemmers** (Fray Justo Sarmiento 2350, Olivos, Buenos Aires, Argentina): Nos abrió sus puertas para conocer el centro de simulación y poder manipular la tecnología con la que está equipado.
- **AARBA - Asociación de Anestesia, Analgesia y Reanimación de Buenos Aires** (Aranguren 1323, Capital Federal, Argentina) y **Dr. Ing. Matías Madorno**: Nos ilustraron en el uso de la simulación en la Asociación y cómo la misma colabora en el entrenamiento y matriculación de anestesistas.
- **Dr. Roberto Duré y equipo** (Hospital de Infecciosas “Dr. Francisco Javier Muñiz”, Uspallata 2272, CABA, Argentina): Nos permitieron presenciar un procedimiento de broncoscopía complejo.
- **Dra. María Paz Menendez y Dr. Gonzalo** (Castaño- Hospital Alemán, Pueyrredón 1640, CABA, Argentina): Estudiaron la usabilidad del software de simulación desarrollado.
- **Dr. Ing. Federico Milano**: Facilitó las imágenes tomográficas en conjunto con su procesamiento para el segmentado del modelo anatómico y respondió nuestras consultas con su experiencia en materia de simulación.
- **Dr. Daniel Alejandro Samolski**: Estudió la usabilidad del software.

No queremos dejar de mencionar el apoyo constante de Sensus3D y su equipo compuesto por **Santiago Bestani**, **Santiago Racca** y **Alberto Gentile**. A "los Santiagos", por la paciencia infinita, por enseñarnos las entrañas de su trabajo, y por la presencia permanente pese a la distancia. A Teo, porque sin sus diseños, el resultado nunca habría sido tan bueno, logrando plasmar cada uno de los pedidos a la perfección. Este trabajo es producto de la sinergia de todos sus miembros. Nos llevamos aprendizaje, experiencia y una gran amistad.

Índice general

Resumen III

Agradecimientos V

1. Presentación 1
1.1. Introducción . . . 1
1.2. Objetivos . . . 2
1.2.1. Objetivos de mínima . . . 2
1.2.2. Objetivos de máxima . . . 3

2. Marco teórico 5
2.1. Anatomía de la vía aérea . . . 5
2.1.1. Vía aérea superior . . . 5
2.1.2. Vía aérea inferior . . . 9
2.2. Patologías . . . 12
2.2.1. Patologías obstructivas intrínsecas . . . 12
2.2.2. Patologías obstructivas extrínsecas . . . 13
2.2.3. Otras patologías . . . 13
2.3. Broncoscopía . . . 14
2.3.1. Reseña histórica . . . 14
2.3.2. El broncoscopio, clasificación y características . . . 18
Broncoscopio rígido . . . 18
Broncoscopio flexible . . . 19
Caracterización de los movimientos de un broncoscopio flexible . . . 22
2.3.3. Procedimientos . . . 25
Procedimientos diagnósticos . . . 25
Procedimientos terapéuticos . . . 26
2.3.4. Visita al Hospital de enfermedades infecciosas "Dr. Francisco Javier Muñiz" 29
2.4. Simulación y educación . . . 32
2.4.1. Tipos de simuladores en el mercado . . . 33
EndoVR bronchoscopy (CAE Healthcare, Montreal, Quebec, Canadá) . . . 34
GI-BRONCH Mentor (3D Systems-Symbionix) . . . 35
ORSIM (Airway Limited, South Yorkshire, UK) . . . 35
Thoracic Anesthesia Bronchoscopy Simulator (University Health Network) 36
BronchoGuide (Olympus Europa SE & Co. KG) . . . 36

3. Marco metodológico 37
3.1. Introducción . . . 37
3.2. Segmentación de modelos en 3DSlicer . . . 38
3.2.1. Proceso . . . 38
3.2.2. Diseño . . . 42
3.2.3. Escena principal . . . 43

3.3. El simulador . . . 43
3.3.1. Unity 3D . . . 43
Editor . . . 44
Juego . . . 44
Script . . . 44
Asset . . . 44
Escena . . . 45
GameObject . . . 46
Inspector . . . 46
Componentes . . . 46
3.4. Software . . . 48
3.4.1. Escena de bienvenida . . . 51
3.4.2. Escena de selección de nivel . . . 51
3.4.3. Escena de exploración . . . 52
3.4.4. Escena principal . . . 53
Objetos . . . 55
Caracterización de los movimientos del cuerpo del broncoscopio . . . 60
Caracterización de los movimientos de la pinza . . . 65
Colisiones . . . 65
Biopsia . . . 66
3.4.5. Escena de Resultados . . . 70
3.4.6. *Scripts* . . . 72
3.5. Mejoras a futuro . . . 72

4. Conclusiones **75**

A. *Scripts* del programa en Unity3D **81**

Índice de figuras

1.1. EVIS EXERA III, de Olympus. Torre de endoscopía que permite visualizar en alta definición la imagen obtenida por un videobroncoscopio. 2

2.1. Sistema Respiratorio 6
2.2. Vía aérea superior 7
2.3. Visualización de los cornetes superior, medio e inferior en la cavidad nasal [1] . . 7
2.4. Cartílagos de la laringe 8
2.5. Subdivisión progresiva del árbol bronquial y sus correspondientes generaciones [2] 10
2.6. Visualización de la tráquea y su contexto anatómico [3] 11
2.7. Difusión de gases a través de la membrana alvéolo capilar [4] 12
2.8. Sistema endoscópico moderno completo [5] 15
2.9. Conductor lumínico de Philip Bozzini [6] 15
2.10. Gustav Killian, padre de la broncoscopía [7] 16
2.11. Dr. Shigeto Ikeda, [7] 16
2.12. Línea de hitos en la historia de la broncoscopía [7] 18
2.13. Broncoscopio rígido de la marca Lymol[8]). 19
2.14. Tubos bronquiales y traqueales de diferentes tamaños (Lymol) [8]. 19
2.15. Principio de diseño de un fibrobroncoscopio [5] 21
2.16. Partes de un fibrobroncoscopio: (1) Pieza ocular, (2) Anillo de dioptría, (3) Palanca de control, (4) Orificio para el canal de trabajo, (5) Cuerpo, (6) Fibra de inserción, (7) Fuente lumínica y (8) Válvula de succión [9] 21
2.17. Partes de un videobroncoscopio: (3) Palanca de control, (4) Orificio para el canal de trabajo, (5) Cuerpo, (6) Fibra de inserción, (7) Fuente lumínica y (8) Válvula de succión [9] 21
2.18. Largo de un broncoscopio [9] 22
2.19. Movimiento de angulación [9] 22
2.20. FOV de un broncoscopio [9] 22
2.21. Porción distal 23
2.22. Evolución temporal de los parámetros de un broncoscopio flexible) [8]). 23
2.23. Modelo de caracterización del movimiento de la fibra de inserción de un broncoscopio [10] 24
2.24. Tipos de fórceps para biopsia: (a y b) pinza cocodrilo; (c) copa oval con extremo cortante; (d y e) copa cocodrilo con extremo cortante; (f) con una aguja entre copas [11] 26
2.25. Sonda para ultrasonido endobronquial en conjunto con aguja de punción [12] . . 26
2.26. Carina de color verde, cuyo epitelio es normal, con regiones en color rosado que implican tejido en etapa previa al carcinoma [13] 27
2.27. Tipos de intervenciones y sus técnicas [14] 27
2.28. Características de los principales lásers [11] 28
2.29. Crioterapia 29

2.30. Modelo Dumon de *stent* y sus posibles variantes: radiotransparente, con forma de reloj de arena y radioopaco [11] . . . 29
2.31. Colocación y remoción de un stent . . . 30
2.32. Válvula endobronquial Zephyr® y su colocación con un broncoscopio flexible en la región a aislar [15] . . . 30
2.33. Fibrobroncoscopio Pentax FB-15V del Hospital Muñiz . . . 31
2.34. Mediciones realizadas al Pentax FB-15V. La porción distal del tubo de endoscopio la cual es flexible y se controla con el cuerpo se corresponde a 40 mm. . . . 32
2.35. Simulador de baja fidelidad [11] . . . 34
2.36. Simulador EndoVR, de la compaña CAE Healthcare [16] . . . 34
2.37. Hardware y UI del simulador GI-BRONCH Mentor [17] . . . 35
2.38. Simulador portable ORSiM [18] . . . 35
2.39. Thoracic Anesthesia Bronchoscopy Simulator [19] . . . 36
2.40. Display del software de simulación BronchoGuide, de Olympus [20] . . . 36

3.1. Pantalla de inicio del 3D Slicer, con la máscara abierta en el programa . . . 39
3.2. Pantalla principal del módulo de segmentación de 3DSlicer, que permite visualizar las diversas herramientas de segmentado que el programa ofrece . . . 40
3.3. Herramienta de Threshold con todo el rango de grises establecido (de 0 a 2.0). El valor 2.0 representa los tonos de gris más claros, y el 0 los más oscuros. . . . 40
3.4. Herramienta de Threshold con un umbral establecido entre 1.95 y 2.00 . . . 41
3.5. Visualización de la estructura tridimensional segmentada con umbrales . . . 41
3.6. Comparación de las estructuras tridimensionales obtenidas variando el ancho de la herramienta *Hollow* . . . 42
3.7. Imágenes de Broncoscopía . . . 43
3.8. Diseño de la lesión endobronquial utilizada . . . 43
3.9. Interfaz del Editor de Unity3D . . . 44
3.10. Malla poligonal de un objeto en Unity3D . . . 45
3.11. Sistema de coordenadas global y ejes locales de distintos objetos [21] . . . 45
3.12. Cuatro tipos de GameObject en Unity3D . . . 46
3.13. Tipos de *Colliders* . . . 47
3.14. *Site Map* del Software . . . 48
3.15. *User Flow* del Software . . . 49
3.16. Primer cartel explicativo del *User Flow* . . . 50
3.17. Segundo cartel explicativo del *User Flow* . . . 50
3.18. Escena de bienvenida . . . 51
3.19. Escena de selección de nivel . . . 52
3.20. Planos de corte [22] . . . 53
3.21. Corte de Tomografías Computadas . . . 53
3.22. Controles para navegar el modelo . . . 54
3.23. Escena principal . . . 54
3.24. Mapa de calor y visualización broncoscopio de la Escena principal . . . 55
3.25. Anatomía de la vía respiratoria . . . 56
3.26. Comando del movimiento del broncoscopio, expulsión de agua y pinza de biopsia 56
3.27. Modelos bronquiales segmentados en 3DSlicer importados a Unity3D como archivos .obj . . . 57
3.28. Sección distal del broncoscopio modelada como una cadena de eslabones con limitación angular, generando un arco de curvatura correspondiente a la angulación distal [23] . . . 58
3.29. Eslabones . . . 59
3.30. Ubicación broncoscopio y efecto de luz . . . 60

3.31. Expulsión de agua por parte del broncoscopio . . . 60
3.32. Modelo de Pinza . . . 61
3.33. Movimientos distales del broncoscopio . . . 62
3.34. Diferencia entre el sistema de coordenadas global, y el local al broncoscopio. . . . 62
3.35. Cilindro transparente con borde anaranjado en la entrada de la vía aérea . . . 64
3.36. Apertura de las puntas de la pinza . . . 66
3.37. *CapsuleCollider* de los eslabones que forman broncoscopio . . . 66
3.38. *CapsuleCollider* central de la pinza . . . 66
3.39. *MeshCollider* de los bronquios . . . 67
3.40. Diagrama explicativo del *script collisionTip* . . . 68
3.41. Colisión de la pinza . . . 69
3.42. Pasos de una biopsia . . . 70
3.43. Escena de Resultados . . . 71
3.44. Clases de cada objeto . . . 72
3.45. Esquema de los movimientos que controla *KinematicManager* . . . 73

A.54.*DestroyLevel* . . . 83
A.1. *BiteableMesh* . . . 84
A.2. *BoxFriction* . . . 85
A.3. *BronchiPainterBehaviour* . . . 86
A.4. *CameraPainter* . . . 87
A.5. *CameraPainter* . . . 88
A.6. *CameraPainter* . . . 89
A.7. *CameraPainter* . . . 90
A.8. *collisionTip* . . . 91
A.9. *collisionTip* . . . 92
A.10.*collisionTip* . . . 93
A.11.*collisionTip* . . . 94
A.12.*collisionTip* . . . 95
A.13.*collisionTip* . . . 96
A.14.*explorationManager* . . . 97
A.15.*explorationManager* . . . 98
A.16.*explorationManager* . . . 99
A.17.*explorationManager* . . . 100
A.18.*FluidShooter* . . . 101
A.19.*GranulomaBehaviour* . . . 102
A.20.*KinematicManager* . . . 103
A.21.*KinematicManager* . . . 104
A.22.*KinematicManager* . . . 105
A.23.*KinematicManager* . . . 106
A.24.*KinematicManager* . . . 107
A.25.*KinematicManager* . . . 108
A.26.*KinematicManager* . . . 109
A.27.*KinematicManager* . . . 110
A.28.*KinematicManager* . . . 111
A.29.*ExtendedFlyCam* . . . 112
A.30.*ExtendedFlyCam* . . . 113
A.31.*Limitation* . . . 114
A.32.*Limitation* . . . 115
A.33.*MovementDistal* . . . 116
A.34.*MovementDistal* . . . 117

A.35.*MovementDistal* . . . 118
A.36.*MovementDistal* . . . 119
A.37.*MovementForwardTop* . . . 120
A.38.*MovementForwardTop* . . . 121
A.39.*nipperCombined* . . . 122
A.40.*nipperCombined* . . . 123
A.41.*nipperCombined* . . . 124
A.42.*nipperCombined* . . . 125
A.43.*Rotate* . . . 126
A.44.*ScoreManager* . . . 127
A.45.*ScoreManager* . . . 128
A.46.*ScoreManager* . . . 129
A.47.*ScoreManager* . . . 130
A.48.*ScoreManager* . . . 131
A.49.*SetGravity* . . . 132
A.50.*SetGravity* . . . 133
A.51.*ChangeScene* . . . 134
A.52.*DestroyLevel* . . . 135
A.55.*GetLevel* . . . 136
A.56.*SelectCase* . . . 137
A.57.*SelectCase* . . . 138
A.58.*SelectCase* . . . 139
A.59.*SelectCase* . . . 140
A.60.*Timer* . . . 141
A.61.*SetDifficulty* . . . 142
A.62.*DropdownChoosing* . . . 143
A.63.*DropdownChoosing* . . . 144

Índice de cuadros

3.1. Resultado de interacción de *Collider* con *Rigidbody* 48
3.2. Modelos utilizados y características según nivel de dificultad. Las máscaras son las proporcionadas por la cátedra Procesamiento de Imágenes Biomédicas. 52
3.3. Resumen de los *scripts* que controlan los movimientos del broncoscopio, a qué eslabón se asocian y en qué eje actúan. 63
3.4. Caracterización de los movimientos del broncoscopio en *KinematicManager* . . . 63
3.5. Resumen de lo que debe tener activo cada opción que devenga del *input* de un usuario 74

A.1. *Scripts* del programa 82

Capítulo 1

Presentación

1.1. Introducción

La asignatura "Simulación y Planificación Quirúrgica" impartida por la Universidad Politécnica de Madrid, la cual propone diseñar un escenario médico virtual que simule uno real, motivó a adentrarse en la búsqueda de tecnologías de simulación en la región de Argentina. En esta búsqueda, se halla la empresa mendocina Sensus3D que "fue fundada con el propósito de introducir al mercado simuladores más innovadores que combinen tecnologías modernas aplicadas al aprendizaje, a un precio accesible para las instituciones" [24]. La empresa ofrece escenarios de entrenamiento especializado, con el fin de educar y perfeccionar técnicas médicas.

Sin embargo, durante la búsqueda se encontró que la simulación no se corresponde con la técnica educativa más utilizada en las instituciones. Uno de los motivos es su costo: el dispositivo háptico utilizado por la empresa Sensus3D, por ejemplo, es el Geomagic Touch Haptic System que ronda los 2800 dólares. En consecuencia, en la Argentina, los residentes aprenden en la práctica cotidiana sin enfrentarse a situaciones artificiales previas. Estas situaciones, según el instructor de SIMMER (Simulación Médica Roemmers) Pablo Pollini, inducen a la educación en un contexto de presión en el cual no es posible analizar las falencias a lo realizado.

Ante este contexto, es motivante realizar un escenario de simulación que permita ofrecer un entorno virtual que se asemeje a las situaciones reales que un practicante debe afrontar, con las bondades de tratarse de una situación artificial.

La broncoscopía es una técnica terapéutica y diagnóstica que permite la exploración de la vía aérea. En el siguiente trabajo se desarrolla un software en el cual se busca recrear escenarios de exploración virtuales, interactivos al usuario, imitando la imagen expuesta en un videobroncoscopio (Figura 1.1), con el fin de que el usuario se familiarice con la visión que ofrece una cámara endoscópica. El mismo está pensado para que, además del entrenamiento visual, se le incorpore a futuro un broncoscopio, y de este modo se entrenen técnicas de manejo del instrumental. En el siguiente trabajo, entonces, se plantea el uso del teclado como interface entre el usuario y el software, sin perder de vista que el mismo pueda ser incorporado a un hardware que involucre tanto un broncoscopio, como posible devolución háptica.

Asimismo, teniendo en cuenta que un simulador carece del *feedback* de un paciente real, se plantea el software con lineamientos de gamificación, en el cual el rendimiento de la práctica realizada se vea expuesto en los resultados de la simulación, enmarcada en un juego. De este

modo, es posible objetivar el rendimiento y además, ofrecer un entorno amigable y desafiante al usuario.

FIGURA 1.1: EVIS EXERA III, de Olympus. Torre de endoscopía que permite visualizar en alta definición la imagen obtenida por un videobroncoscopio.

1.2. Objetivos

1.2.1. Objetivos de mínima

Como objetivo de mínima se busca armar tres casos de extracción de muestra. Para los mismos se deben cumplimentar los siguientes requerimientos:

1. Ubicación de modelo respiratorio en un ambiente tridimensional, procurando semejanza a la realidad.
2. Estudio y simulación del comportamiento físico de un endoscopio.
3. Estudio y simulación del comportamiento físico de las pinzas de extracción de muestra
4. Análisis físico de las colisiones de los elementos participantes (modelo respiratorio e instrumental).
5. Jugabilidad:
    - Mapeo de movimientos a teclas de instrumento y pinzas

- Manejo de las cámaras: generales y cámara incorporada al broncoscopio
- Manejo de colisiones entre instrumento y paredes de las vías respiratorias
- Adición de *gadgets:* mapa de ubicación de endoscopio en el sistema respiratorio, como método de ayuda

6. *Scoring:*
   - Obtención de muestras en el lugar indicado, destacando el hecho de que las mismas no deben realizarse en las paredes sanas
   - Obtención de muestras sin dañar las paredes de las vías respiratorias
   - Medición del tiempo de desarrollo del estudio
   - Incorporación de un mapa de calor que indique las zonas más inspeccionadas

1.2.2. Objetivos de máxima

En cuanto objetivos deseables a alcanzar por encima de los de mínima se encuentran los siguientes:

1. Adición de más casos de extracción de muestra
2. Adición de la instancia de retirar un objeto foráneo de los pulmones
3. Integración del sistema con Realidad Virtual
4. Integración del control de endoscopio con un hardware que ofrezca sensaciones hápticas, provisto por la empresa.

Capítulo 2

Marco teórico

2.1. Anatomía de la vía aérea

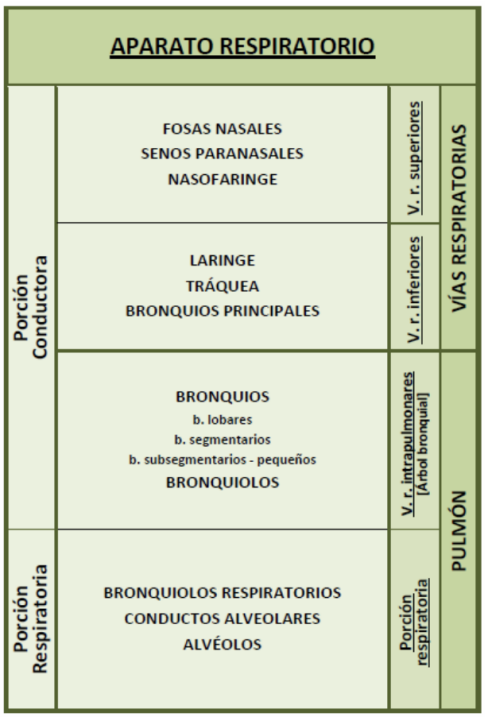
Con el motivo de ilustrar el contexto anatómico en el cual se desarrolla una broncoscopía, se procede a describir las distintas secciones de la vía aérea. La relevancia de su comprensión se atribuye a que un buen conocimiento de la vía aérea puede prevenir complicaciones durante una intervención. Un ejemplo a considerar es la publicación del primer algoritmo de intubación a cargo de la Sociedad Americana de Anestesiología, que trajo como consecuencia una disminución en las complicaciones de la intubación durante la anestesia "de un 62 % entre los años 1985 y 1992, a un 35 % entre 1993 y 1999" [25].

Los pulmones se encuentran ubicados en la cavidad torácica, y como órgano macizo se compone del estroma (la pleura visceral, los tabiques conectivos, el árbol de tejido conectivo y el tejido intersticial) y el parénquima, que a su vez está dividido en la porción conductora (bronquios y bronquíolos) y la porción respiratoria (bronquíolos respiratorios, conductos alveolares, sacos alveolares y alvéolos), como resume la Figura 2.1.

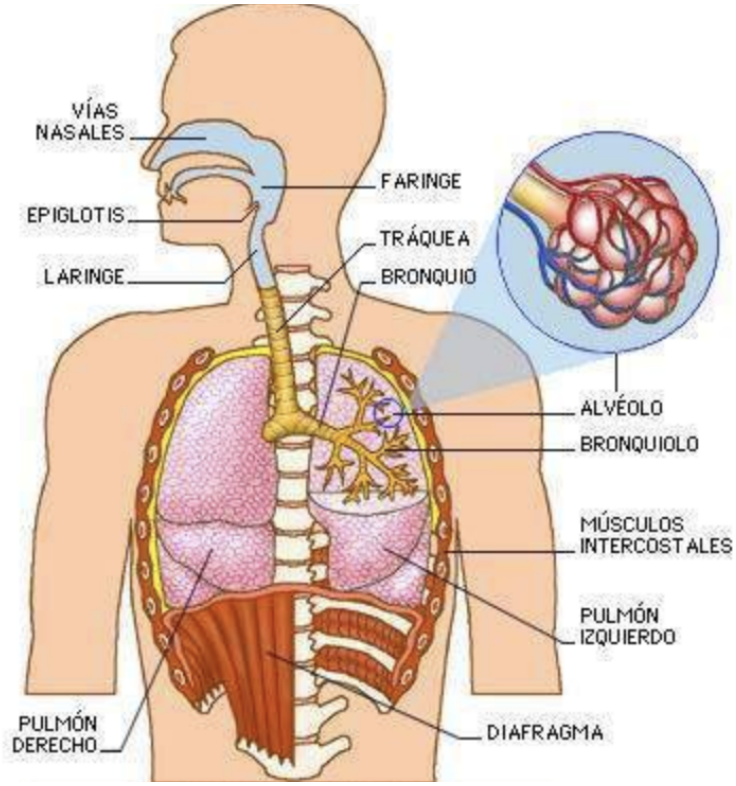
2.1.1. Vía aérea superior

En primer lugar, se describe la vía aérea superior, ilustrada en la Figura 2.2. Es posible distinguir la cavidad oral, comprendida por las piezas dentales, la lengua y el paladar. El paladar rígido está constituido por tejido óseo y define el techo de la boca. El paladar blando, en cambio, se conforma de tejido muscular, y colabora en la deglución, cerrando la nasofaringe para evitar el ingreso de alimentos a la vía aérea, culminando en la úvula. Por otra parte, la cavidad nasal está compuesta por el tabique nasal, que la divide en dos cavidades y las paredes nasales laterales. Las paredes nasales contienen protuberancias óseas denominadas cornetes (superior, medio e inferior en la Figura 2.3) que cumplen la función de humectar, filtrar, direccionar y amortiguar el aire ingresante. Cuando un broncoscopio se introduce por la cavidad nasal, es posible visualizar el cornete inferior lateralmente y el tabique nasal medialmente. Algunos rasgos para evaluar son la presencia de pólipos y la integridad de la mucosa [11].

A continuación, siguiendo la dirección del flujo entrante del aire, se encuentra la orofaringe, que nace en la base de la lengua y culmina en la epiglotis, siendo bordeada superiormente por el velo del paladar. En las paredes laterales de esta estructura anatómica es posible observar las amígdalas. En las paredes laterales, entre la base de la lengua y la superficie anterior de la epiglotis, se ubican las valéculas, que usualmente se corresponden con lugares de atrapamiento de cuerpos extraños y obstrucción de la vía aérea superior, en consecuencia [11].



(A) Resumen de la anatomía del sistema respiratorio [26]

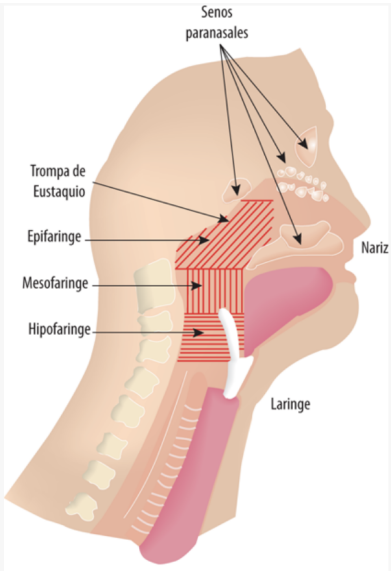


(B) Anatomía del sistema respiratorio [27]

FIGURA 2.1: Sistema Respiratorio

A la orofaringe la procede la hipofaringe, que es delimitada por la epiglotis (su extremo distal) hasta el cartílago cricoides, continuando hacia atrás para convertirse en esófago. Esta estructura anatómica facilita la deglución del bolo alimenticio en el tracto digestivo superior y se encuentra

(A) Algunas estructuras de la vía aérea superior[11]



(B) Otras estructuras de la vía aérea superior [28]

FIGURA 2.2: Vía aérea superior

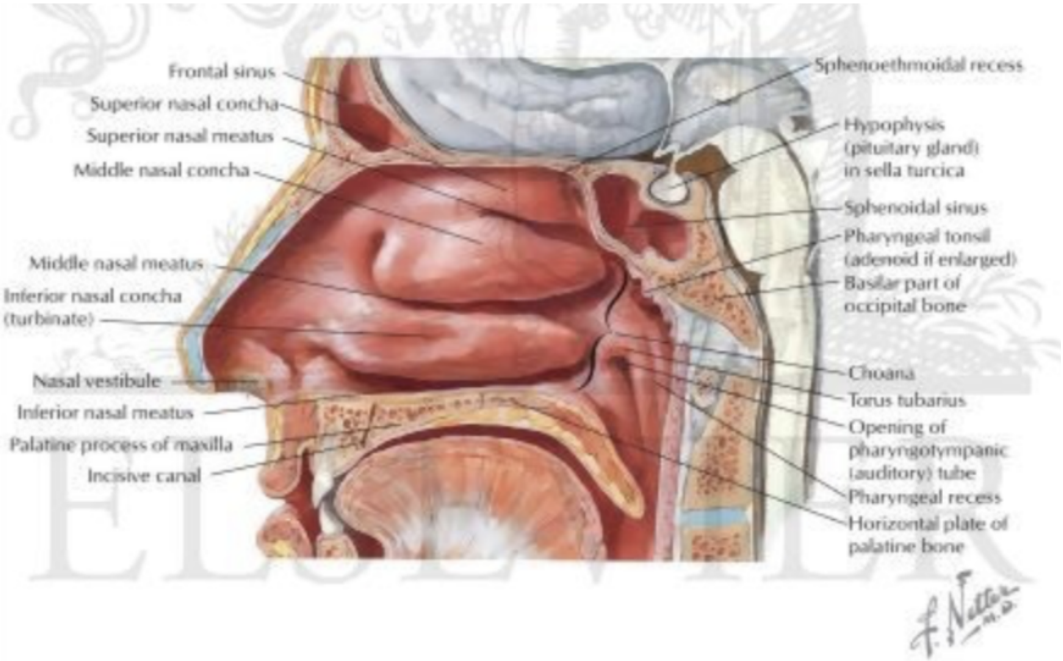


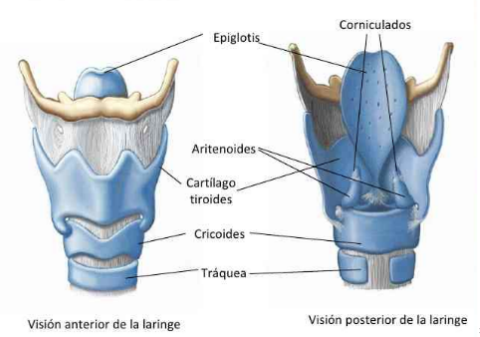
FIGURA 2.3: Visualización de los cornetes superior, medio e inferior en la cavidad nasal [1]

inervada por el nervio gástrico y el nervio vago. Una sección importante de esta estructura es el receso piriforme, en el cual descansa la cuchilla del laringoscopio en el caso de una intubación [11].

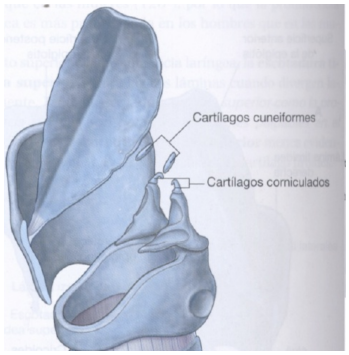
La laringe es una de las estructuras de mayor función. Tiene responsabilidad en el proceso de fonación, en la respiración y en su objetivo más relevante, evitar las aspiraciones. Se ubica desde

la tercera hasta la sexta vértebra cervical. En hombres se sitúa inferior al hueso hioides mientras que en mujeres y niños, se sitúa más alto. Esta estructura posee pliegues en la mucosa, y la misma se encuentra revestida de epitelio columnar escamoso en las cuerdas bocas, superficie posterior de la epiglotis, pliegues ariepiglóticos y la comisura posterior. En cambio, en el resto del aparato laríngeo, la mucosa se reviste de epitelio ciliado. Asimismo, posee nueve cartílagos que se pueden observar en la Figura 2.6 con sus correspondientes membranas y ligamentos de conexión:

- **Tiroides** (impar): consiste en un par de láminas que se fusionan hacia abajo en un ángulo de 90 grados en hombres ("Nuez de Adán") y 120 grados en mujeres. Este cartílago se encuentra cerca de la superficie de la piel, es relativamente avascular y sirve como punto de referencia para una cricotiroidotomía emergente.
- **Cricoides** (impar): se ubica por debajo del cartílago tiroides y es reconocido por ser el único anillo cartilaginoso completo.
- **Epiglotis** (impar): fibrocartílago elástico que se encuentra revestido con membrana mucosa. La cara anterior de la epiglotis forma la entrada de la laringe.
- **Aritenoides** (pares)
- **Corniculados** (pares)
- **Cuneiformes** (pares)



(A) Cartílagos de la laringe [29]



(B) Cartílago cuneiforme [30]

FIGURA 2.4: Cartílagos de la laringe

2.1.2. Vía aérea inferior

La vía aérea inferior está compuesta, en primer lugar, por la tráquea. Este órgano tiene 12 cm de largo y se ubica entre las vértebras C6-C7 hasta D4-D5. Si se la observa lateralmente, se distingue un curso oblicuo de superoanterior a inferoposterior. Termina dividiéndose en dos bronquios con un ángulo de 60 grados. Sus dimensiones varían según la edad, el sexo y otros factores. Son característicos sus anillos traqueales en el borde posterior conformados por una membrana fibromuscular.

La tráquea sana es un órgano rígido que sirve de vía de ingreso y egreso del aire inhalado y exhalado. Está compuesta por una serie de cartílagos hialinos, entre 16 y 20, en forma de C. Sus extremos abiertos son estabilizados por fibras musculares lisas y tejido conjuntivo, por lo que esta superficie que se encuentra en contacto con el esófago es plana. Esta fisonomía culmina en la carina [31]. Es frecuente que se obstruya total o parcialmente por elementos extraños. También es protagonista del procedimiento denominado traqueostomía que crea una vía alternativa para el ingreso y egreso de aire de manera quirúrgica [32].

A continuación de la tráquea, la carina establece la división en dos ramificaciones que dan inicio al árbol bronquial. El árbol resulta en veinticinco bifurcaciones principales (pueden haber variaciones anatómicas). La pared bronquial está compuesta principalmente por tejido cartilaginoso, muscular y mucosa. Cabe destacar que la bifurcación izquierda es distinta que la derecha por la presencia del corazón:

- Bronquio Principal Derecho (BPD)
  1. Lóbulo Superior Derecho (LSD)
     - Segmento bronquial superior/apical (B1)
     - Segmento bronquial posterior/dorsal (B2)
     - Segmento bronquial anterior/ventral (B3)
  2. Lóbulo Medio Derecho (LMD)
     - Bronquio segmental medial (B5)
     - Bronquio segmental lateral (B4)
  3. Lóbulo inferior derecho:
     - Bronquio segmental superior/apical (B6)
     - Bronquio segmental basal medial (B7)
     - Bronquio segmental basal posterior/dorsal (B10)
     - Bronquio segmental basal lateral (B9)
     - Bronquio segmental basal anterior (B8)
- Bronquio Principal Izquierdo (BPI): a diferencia del BPD, no hay ramificaciones tempranas y es alrededor de 5 cm. más largo.
  1. Lóbulo Superior Izquierdo (LSI)
     - Bronquio segmental lingular superior e inferior (B4 y B5)
     - Bronquio segmental posterior apical (B1) y posterior dorsal (B2)

- Bronquio segmental anterior/ventral (B3)

2. Lóbulo Inferior Izquierdo (LII)
   - Bronquio anteriomedial o Bronquio anterior y bronquio medial (si se encuentran separados) (B7 y B8)
   - Bronquio lateral (B9)
   - Bronquio posterior/dorsal (B10)
   - Bronquio superior/apical (B6)

Los bronquios también pueden sub-clasificarse en generaciones, que en total son 23 [33]. En primer lugar se encuentran los bronquios mayores, de la generación 1 a la 4. A continuación, los bronquios menores, de la generación 5 a la generación 11 como se observa en la Figura 2.5 [34].

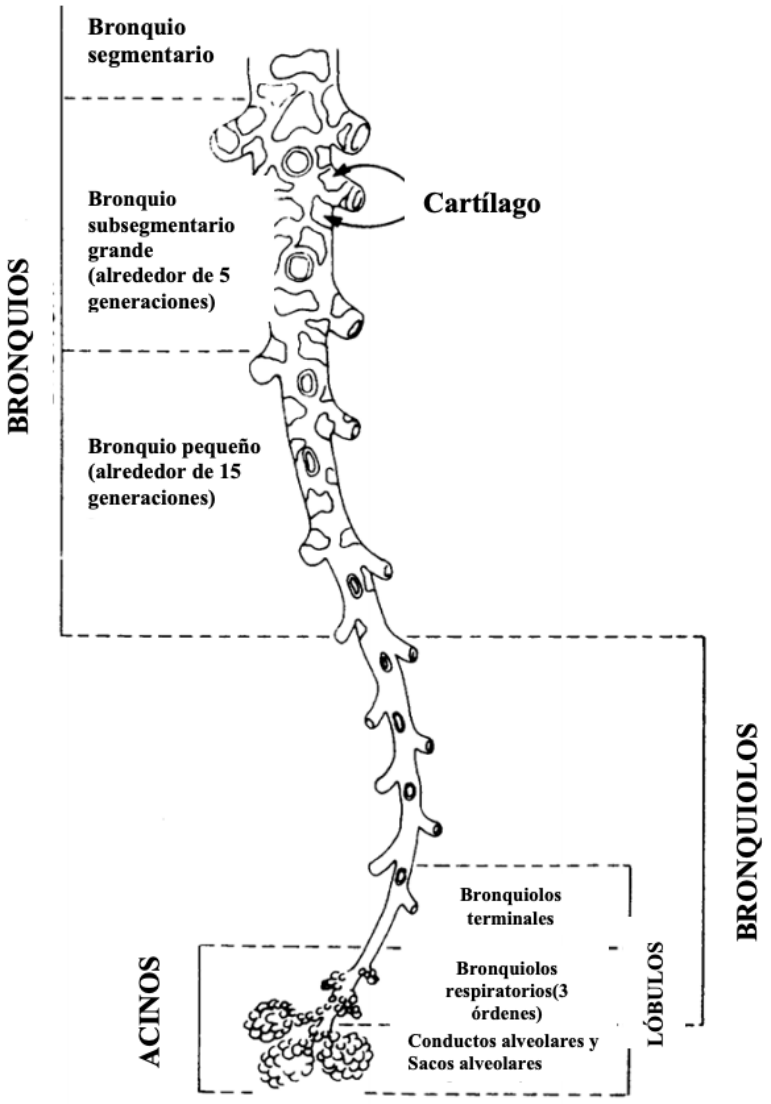


FIGURA 2.5: Subdivisión progresiva del árbol bronquial y sus correspondientes generaciones [2]

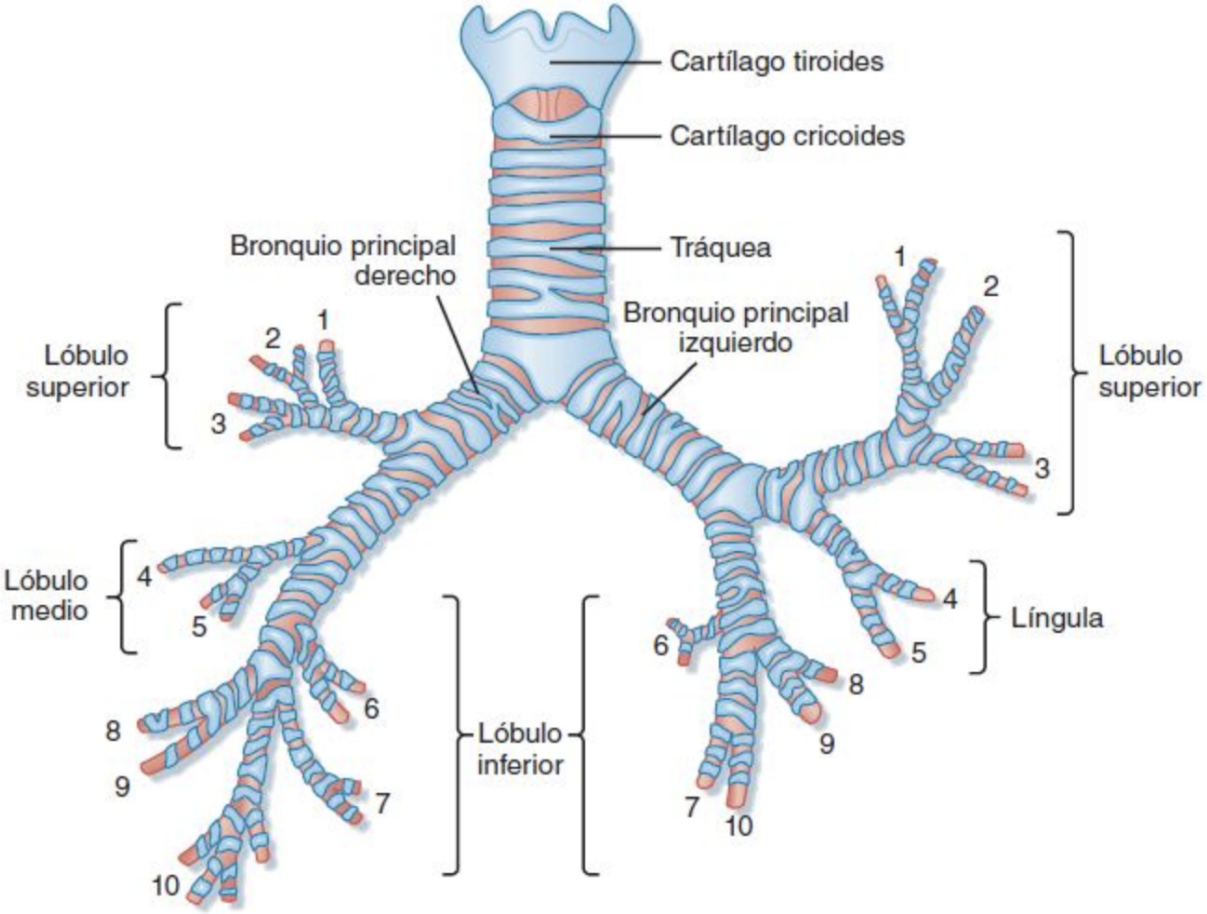


FIGURA 2.6: Visualización de la tráquea y su contexto anatómico [3]

Los bronquíolos (que van desde la generación 12 a la generación 16) proceden a los bronquios y son menores en diámetro. Sus paredes se componen de músculo liso, sin presencia de tejido cartilaginoso. Cada pulmón posee 30.000 bronquíolos, dando un total de 60.000. La generación 16 incluye a los bronquíolos terminales, siendo estos la porción de la vía aérea más pequeña que no contiene alvéolos. Luego, el acino pulmonar, se compone de aquellas generaciones en las cuales existe algún alvéolo en su pared (generaciones 17, 18, y 19). Las generaciones 20, 21 y 22 se constituyen por los conductos alveolares [35].

En última instancia, siendo el último eslabón del árbol bronquial, se mencionan los sacos alveolares (generación 23), responsables del intercambio gaseoso. El alvéolo está rodeado de capilares pulmonares que aportan la sangre desoxigenada proveniente del resto del organismo, para que la misma se aprovisione del oxígeno proveniente de la respiración, por medio de la difusión de este gas a través de la membrana alvéolo-capilar. El dióxido de carbono, contenido en mayor concentración en la sangre proveniente del corazón derecho, es difundido al recinto alveolar, resultando así en un intercambio de gases. La sangre, luego del intercambio, es conducida al corazón izquierdo para que el mismo la bombee al resto del organismo (ver Figura 2.7).

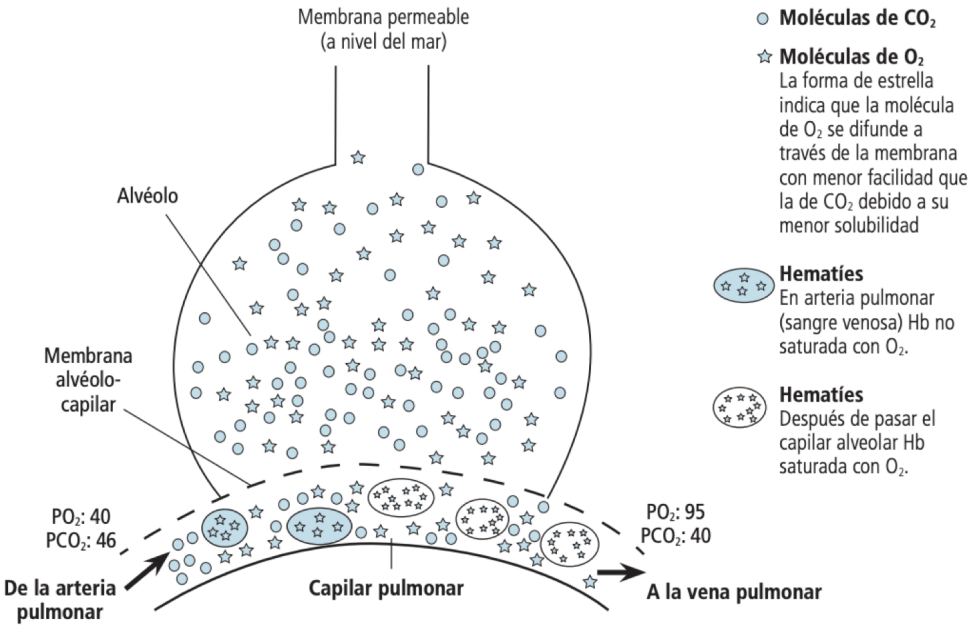


FIGURA 2.7: Difusión de gases a través de la membrana alvéolo capilar [4]

2.2. Patologías

La vía aérea puede verse afectada por diversas patologías que precisen de la broncoscopía para su diagnóstico o tratamiento, cuyas causas pueden ser benignas (por ejemplo, por una estenosis traqueal post intubación) o malignas (como lo puede ser una neoplasia). Se pueden subdividir en tres categorías: patologías obstructivas intrínsecas, patologías obstructivas extrínsecas, y otro tipo de patologías.

2.2.1. Patologías obstructivas intrínsecas

Estas patologías involucran la reducción del diámetro de alguna porción de la vía aérea debido a causas benignas (por intubación prolongada, por ejemplo [36]) o malignas (debido a un carcinoma [37]).

- **Neoplasia:** implica la presencia tumoral tanto maligna como benigna. Pueden ser primarios o secundarios.
- **Formación de granulomas post traqueostomía o remoción de *stent*:** debido a una técnica defectuosa o infección postoperatoria, es posible que aparezcan granulomas en el interior de la luz tráqueal, cerca de la ubicación del *stent* o de la traqueostomía [38]. Un granuloma es una formación local de células específicas como lo son los macrófagos y linfocitos, como reacción inflamatoria ante una infección [39].
- **Cuerpo extraño:** ocurre debido a la aspiración de un objeto –cuerpo extraño– y su alojamiento en la laringe, la tráquea o bronquios [40].
- **Traqueobroncomalacia:** implica debilidad de la pared bronquial en conjunto con una disminución dinámica de la luz de la tráquea. Se la suele confundir con el EPOC, aunque sus causas pueden ser congénitas o adquiridas [41].

2.2.2. Patologías obstructivas extrínsecas

Involucran la reducción de la luz de la vía aérea por factores externos a la vía aérea como lo pueden ser linfoadenopatías o masas cervicales [42].

- **Linfoadenopatías:** Es el término que se utiliza ante la presencia de hinchazón de los ganglios linfáticos [43]. Esta patología puede ocurrir en los ganglios ubicados en el mediastino (espacio de la caja torácica entre las dos pleuras, la columna vertebral y el esternón), y es denominada adenopatía mediastinal. La hinchazón puede deberse a procesos infecciosos, inmunológicos o neoplásicos, como lo es el linfoma de Hodgkin [44].
- **Tumores compresivos:** Los tumores pueden originarse en cualquiera de las estructuras del mediastino o derivarse de anomalías provenientes durante el desarrollo embrionario. Pueden ser asintomáticos y encontrarse luego de un examen radiológico. En otras ocasiones, pueden aparecer síntomas en consecuencia de la compresión y/o desplazamiento de estructuras próximas al tumor, dependiendo de la localización del mismo y su respectivo alcance. En el caso particular de un tumor que cause compresión bronquial, sus síntomas pueden ser tos seca, sibilancias, disnea, o hasta causar otras enfermedades como asma y bronquiolitis [45].

2.2.3. Otras patologías

La **EPOC**, o enfermedad pulmonar obstructiva crónica, es una enfermedad respiratoria crónica, caracterizada por la obstrucción al paso del aire. Es consecuencia de diferentes enfermedades respiratorias, como la bronquitis crónica y el enfisema. El tabaco es la causa más importante de la EPOC. La gravedad de la enfermedad depende de cuánto se fuma, y desde cuándo se fuma [46]. Otros factores que contribuyen son: la contaminación atmosférica, el trabajo en ambientes con mucho polvo, las alergias y las infecciones respiratorias. La EPOC trae como consecuencia que el tejido pulmonar enfermo sea menos elástico que el sano, por lo que durante la inspiración, el mismo no cede. En adición, durante la espiración, el mismo no empuja el aire hacia afuera. Por consiguiente, el volumen espirado reducido ejerce menos fuerza sobre la vía aérea, que se cierra previo a la eyección total ya que el tejido enfermo de menor tono no puede mantener las vías aéreas abiertas hasta el fin de la exhalación. Resumiendo, la capacidad pulmonar se ve disminuida. Asimismo, el intercambio gaseoso es menos eficiente ya que el volumen residual aumentado hiper extiende el tejido de los pulmones, desplazando a la pleura y afectando la funcionalidad del tejido adyacente, que, pese a estar sano, se encuentra comprimido y realiza menos hematosis.

La **hemoptisis** es la expectoración de sangre proveniente del árbol traqueobronquial. Puede ocasionarse debido a bronquiectasias, la bronquitis crónica y el carcinoma broncogénico [47]. La bronquiectasia es una condición en la cual el daño a las vías respiratorias ocasiona su ensanchamiento y flacidez, con posible presencia de cicatrices [48]. El trastorno bronquial más común es la bronquitis, una inflamación de los tubos. La bronquitis puede ser aguda o crónica. En cambio, la bronquiolitis es una inflamación de las vías aéreas pequeñas que se ramifican desde los bronquios.

Otra patología común es la **tuberculosis**, causada por la bacteria Mycobacterium tuberculosis, que se contagia de persona a persona a través del aire [49]. Puede acarrear consecuencias crónicas como la estenosis traqueobronquial, que tiene mayor prevalencia en mujeres (por el menor diámetro de las vías aéreas y menor expectoración). Se trata mediante intervención endoscópica

dilatando con balón o con la colocación de *stents* [50].

El **enfisema pulmonar** involucra la destrucción irreversible de las paredes pulmonares. Esta patología ocurre debido a que los sacos alveolares se han llenado de aire de manera excesiva ante un desequilibrio químico de los agentes de contracción y expansión. Esta patología involucra la disminución de la función respiratoria [51].

2.3. Broncoscopía

La broncoscopía forma parte de un grupo de procedimientos denominado técnicas endoscópicas. La endoscopía permite explorar estructuras anatómicas, así como diagnosticar y tratar patologías [52]. A continuación se enumeran las técnicas endoscópicas más relevantes y los órganos/áreas que explora [53]:

- **Broncoscopía:** tráquea y bronquios
- **Colonoscopía:** colon e intestino grueso
- **Colposcopía:** vagina y cuello uterino
- **Esofagoscopía:** esófago
- **Gastroscopía:** estómago y duodeno
- **Laringoscopía:** laringe
- **Rectoscopía:** recto y colon

2.3.1. Reseña histórica

Cabe anticipar la diferencia de broncoscopios existentes para comprender mejor su evolución a lo largo del tiempo; existen dos grandes grupos de broncoscopios, los rígidos y los flexibles. Ambos son utilizados en la actualidad, según la indicación, necesidad y disponibilidad, e incluso pueden ser complementarios en su uso. La visualización, difiere en el tipo de broncoscopio que se utilice. El broncoscopio flexible ofrece dos alternativas de visualización: los fibrobroncoscopios, ofrecen imágenes analógicas gracias a las propiedades ópticas de la fibra óptica y permiten observar la imagen mediante una lente; los videobroncoscopios, reemplazan la fibra óptica por cámaras que transducen la imagen analógica de manera digital. En un broncoscopio rígido, se puede observar mediante el uso complementario de un broncoscopio flexible, en cualquiera de sus dos alternativas.

La broncoscopía flexible es una técnica endoscópica incorporada en el año 1966 por Shigeto Ikeda [54]. Sin embargo, los orígenes de la técnica se remontan al año 1807. En este año, Philipp Bozzini logra enviar luz de las velas a las cavidades corporales mediante la invención del 'conductor de la luz'[6], como muestra la Figura 2.9. En el año 1823, Horace Green introduce una esponja embebida en nitrato de plata en un catéter dentro de un bronquio, con el fin de quemar lesiones. Luego, se logra la visualización directa de las cuerdas vocales y de las vías respiratorias proximales mediante un esofagoscopio.

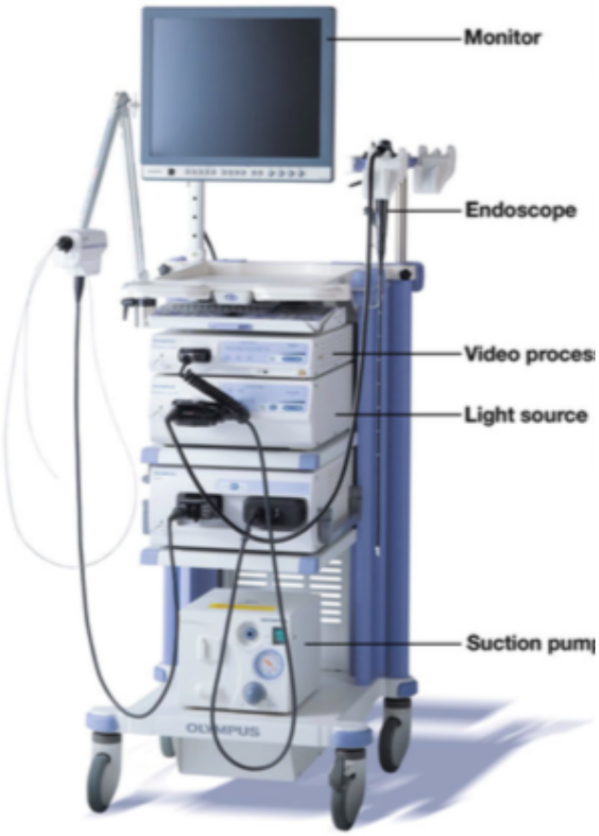


FIGURA 2.8: Sistema endoscópico moderno completo [5]

FIGURA 2.9: Conductor lumínico de Philip Bozzini [6]

En el año 1885, el otorrinolaringólogo Gustav Killian investiga la laringe y la tráquea de un paciente utilizando el esofagoscopio combinado con pinzas rígidas, logrando extraer un hueso derecho del bronquio principal derecho. Este suceso marca un hito en el crecimiento de la técnica, y es por ello que Killian es considerado el padre de la broncoscopía.

En el año 1904, Chevalier Jackson desarrolla el broncoscopio rígido. Le coloca una pequeña luz en la porción distal y un tubo de succión. La broncoscopía rígida se utiliza, entonces, como la

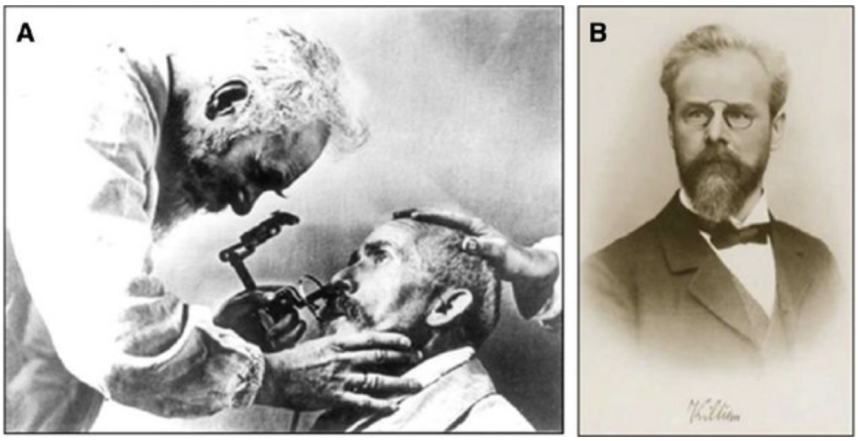


FIGURA 2.10: Gustav Killian, padre de la broncoscopía [7]

práctica de cabecera durante los siguientes 70 años.

Como se menciona previamente, Shigeto Ikeda (Figura 2.11), un cirujano torácico de Tokio, introduce el primer prototipo de un broncoscopio flexible fibro-óptico en Copenhagen. Su diámetro es menor a 6mm, conteniendo alrededor de 15000 fibras de vidrio. Debido a que el Dr. Ikeda era zurdo, el fibrobroncoscopio se diseña para ser utilizado con la mano izquierda. El uso de la broncoscopía rígida disminuye desde la creación del broncoscopio flexible hasta que se inventan herramientas especiales para la colocación de *stent* en 1990 y la aplicación de láser en 1981.

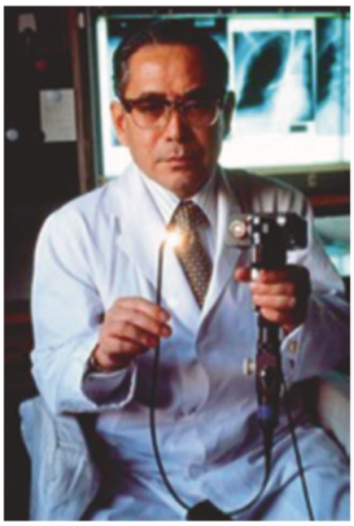


FIGURA 2.11: Dr. Shigeto Ikeda, [7]

En 1984 se lanza al mercado el modelo Olympus BF-1.8T, el primer broncoscopio ultrafino desarrollado por Tanaka et al., con un diámetro de 1.8 mm, el cual no posee un canal de trabajo. Este broncoscopio sólo puede doblarse de manera pasiva y se debe introducir en el canal de trabajo del broncoscopio convencional. Cabe destacar que, pese a los esfuerzos, todavía no se ha logrado un *stent* ideal para apaliar los efectos de la estenosis. Este *stent* debería ser fácil de insertar y extraer, personalizable a la anatomía de la estenosis. También debería restablecer la permeabilidad luminosa al resistir las fuerzas compresivas pero ser lo suficientemente elástico como para

adaptarse a los contornos de las vías respiratorias sin causar isquemia o erosión en estructuras adyacentes [55].

En 1987 la firma Pentax introduce un cambio de tecnología en la visualización de las imágenes, reemplazando la fibra óptica por una cámara de video en el extremo distal del broncoscopio, por lo que comienza la era de la videobroncoscopía. En los años siguientes, se desarrolla la sonda radial de ultrasonido y la técnica de ablación termal mediante gas argón.

El nuevo milenio implicó el desarrollo de herramientas para el tratamiento de patologías del árbol bronquial. Un ejemplo es la incorporación de válvulas unidireccionales para la reducción del volumen pulmonar para pacientes con enfisema. Un dispositivo de control de flujo puede regular el flujo de aire a través del pasaje bronquial permitiendo el flujo de fluido en una primera dirección (por ejemplo, la dirección de exhalación) mientras limita o impide el flujo de fluido en una segunda dirección (por ejemplo, la dirección de inhalación). Asimismo, se exploran sondas convexas de ultrasonido, coagulación por microondas y por láser.

En el 2006 surge la termoplastía bronquial, que implica el uso del calor para inducir cambios estructurales en la pared de la vía aérea. Las investigaciones preliminares en perros muestran que la aplicación de energía térmica a las vías respiratorias disminuye la hiperreactividad de las vías respiratorias y reemplaza el músculo liso con tejido conectivo sin evidencia de cicatrización a los 3 años.

Los últimos avances en la broncoscopía involucran el uso de tomografía de coherencia óptica doppler endoscópica y autofluorescencia de imágenes. La imagen de autofluorescencia brinda información bioquímica de los tejidos mediante la fluorescencia del colágeno y la elastina. El tejido canceroso emite una señal de fluorescencia disminuida y desplazada al rojo debido a la descomposición de los componentes de la matriz extracelular y una mayor absorción sanguínea. Sin embargo, la inflamación y la bronquitis crónica también pueden reducir la fluorescencia [7].

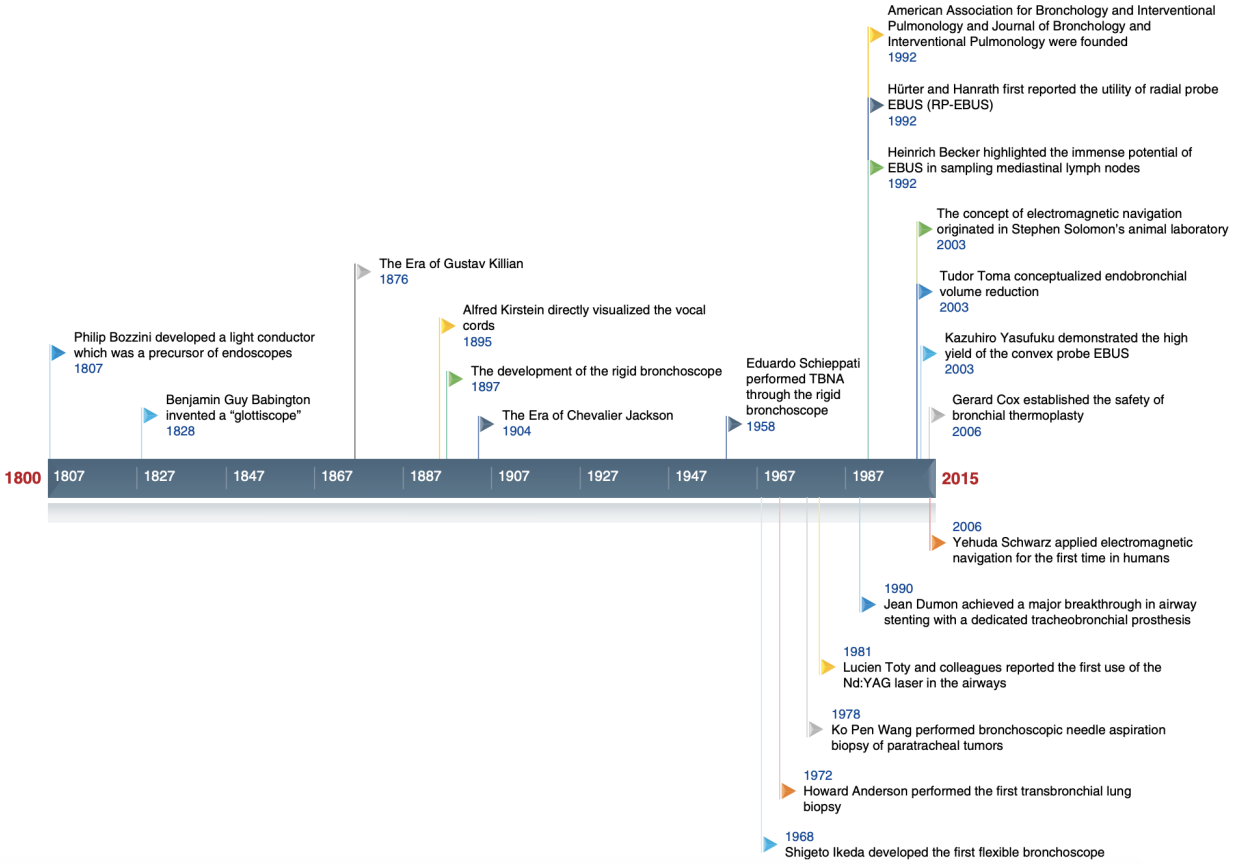


FIGURA 2.12: Línea de hitos en la historia de la broncoscopía [7]

2.3.2. El broncoscopio, clasificación y características

Broncoscopio rígido

Se compone de un tubo abierto, rígido, de acero inoxidable, que puede tener distintos diámetros y longitudes. Se implementa principalmente en aplicaciones terapéuticas. El tubo posee aberturas o fenestraciones laterales que permiten la ventilación lateral y garantizan una oxigenación adecuada mientras se realiza la práctica. Su rigidez tiene como principal ventaja un control total en el lumen de la vía aérea, que resulta provechoso en casos de obstrucción o hemorragia. Asimismo, puede actuar como tubo endotraqueal, permitiendo el ingreso de otros elementos por el canal de trabajo (incluso un broncoscopio flexible).

Las principales empresas que fabrican broncoscopios rígidos son Novatech SA, Richard Wolf Medical Instruments Corporation y Lymol Medical [8]. Su diámetro varía según el tipo de paciente al que esté destinado, pudiendo ser de entre 3 y 18 mm. Su largo varía entre 33 y 43 cm (Figura 2.14). El equipamiento que involucra la técnica de broncoscopía rígida puede incluir, además del tubo, fórceps para biopsia, catéteres de succión, medios de colocación de *stents*, fibras de argón y de láser, sondas de criocauterización y balones de dilatación. Es muy común el uso de un broncoscopio rígido como canal de trabajo de un broncoscopio flexible para acceder a regiones distales de la vía aérea o para asistir en otros procedimientos endobronquiales [8]. En la actualidad esta técnica es comúnmente utilizada por el médico neumonólogo Roberto Duré, Jefe de la Unidad de Endoscopía Peroral del Hospital Municipal Muñiz en Buenos Aires.

FIGURA 2.13: Broncoscopio rígido de la marca Lymol[8]).

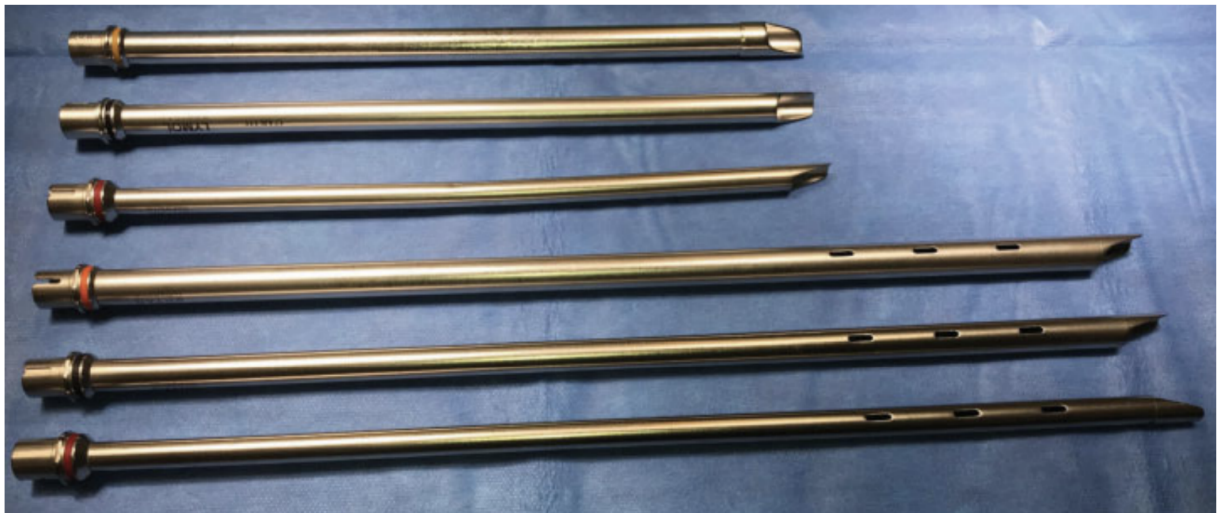


FIGURA 2.14: Tubos bronquiales y traqueales de diferentes tamaños (Lymol) [8].

La broncoscopía rígida se encuentra especialmente indicada en los siguientes casos:

- Estenosis intrínseca de la vía aérea
- Obstrucciones de la vía aérea debido a cuerpos extraños
- Compresión extrínseca
- Hemoptisis masiva
- Procedimientos terapéuticos, como por ejemplo la colocación de *stents*, necesidad de electrocauterización o crioterapia, coagulación mediante plasma de argón, entre otros.

Broncoscopio flexible

La broncoscopía flexible es una herramienta diagnóstico-terapéutica que permite la inspección y acción desde las fosas nasales hasta los segmentos más distales de la vía aérea [56]. Esta técnica requiere de sedación mínima y tiene una baja tasa de complicaciones, ya que en menos del 1 % de los casos se presenta sangrado, depresión respiratoria, paro cardiorrespiratorio, arritmias o neumotórax. El índice de mortalidad de la práctica se encuentra en 0-0.04 %, tomando más de

68.000 procedimientos [57].

Los broncoscopios flexibles varían en diámetro, con la posibilidad de ingresar a distintos niveles bronquiales. El broncoscopio flexible estándar puede acceder hasta la quinta generación de bronquios, mientras que los más modernos, denominados ultrafinos (2.8 mm de diámetro externo) pueden inspeccionar hasta la doceava generación bronquial en adultos.

El primer fibrobroncoscopio ultrafino fue desarrollado en el año 1984 por la firma Olympus [57]. En la actualidad, los videobroncoscopios están reemplazando a los fibrobroncoscopios, ya que la incorporación de una cámara en el extremo distal permite capturar imágenes digitalmente para ser transmitidas a un procesador de video, sin necesidad de fibra óptica [58]. También se implementa la adición de una cámara a la pieza ocular del fibrobroncoscopio, para digitalizar la imagen analógica obtenida por las fibras ópticas. Este artilugio es especialmente implementado en ambientes económicos precarios. Un broncoscopio flexible cuenta con tres componentes principales: la sección de control, el tubo de inserción y el cable de conexión.

- **Sección de control:** se toma con la mano izquierda y aloja el movimiento de mando con el fin de controlar la porción distal del tubo de inserción mediante una palanca de control que se mueve hacia arriba o hacia abajo. Asimismo, aporta un orificio que permite introducir materiales al canal de trabajo, como por ejemplo, una pinza de biopsia. Además posee un puerto de succión con su correspondiente pulsador, el cual activa la aspiración mediante vacío, y un orificio para introducir líquido, también con su pulsador. Todos estos elementos confluyen al canal de trabajo.
- **Tubo de inserción:** se corresponde con el segmento del broncoscopio que ingresa a la vía aérea, y se compone de un catéter flexible de entre 50 y 60 cm de largo. Su diámetro puede ser variable, como se menciona previamente.
- **Cable de conexión:** En un videobroncoscopio (Figura 2.17), transporta la información al procesador de video y provee de energía a la fuente lumínica. Por otro lado, en un fibrobroncoscopio (Figura 2.16) la visualización se realiza por su pieza ocular, por lo que requiere únicamente fuente lumínica.

Un broncoscopio debe cumplimentar ciertos requerimientos como por ejemplo, proveer de iluminación artificial con el fin de permitir la inspección de la vía aérea. Asimismo, debe ofrecer la posibilidad de direccionar su movimiento, de manera que sea posible la navegación a lo largo de las distintas bifurcaciones bronquiales. Otro aspecto a considerar es la posible necesidad de recolectar muestras con el fin de analizar posibles anomalías [13]. Para satisfacer estos requisitos, los broncoscopios flexibles pueden caracterizarse según diversos parámetros (tabla de la Figura 2.22):

- Tipo de broncoscopio (de fibra óptica, videobroncoscopio).
- Largo del tubo de inserción (Figura 2.18)
- Diámetro externo
- Diámetro interno
- Angulación superior e inferior del extremo distal (Figura 2.19)
- Diámetro del canal de trabajo
- Campo de visión de su cámara (*Field of View (FOV)*, Figura 2.20)

FIGURA 2.15: Principio de diseño de un fibrobroncoscopio [5]

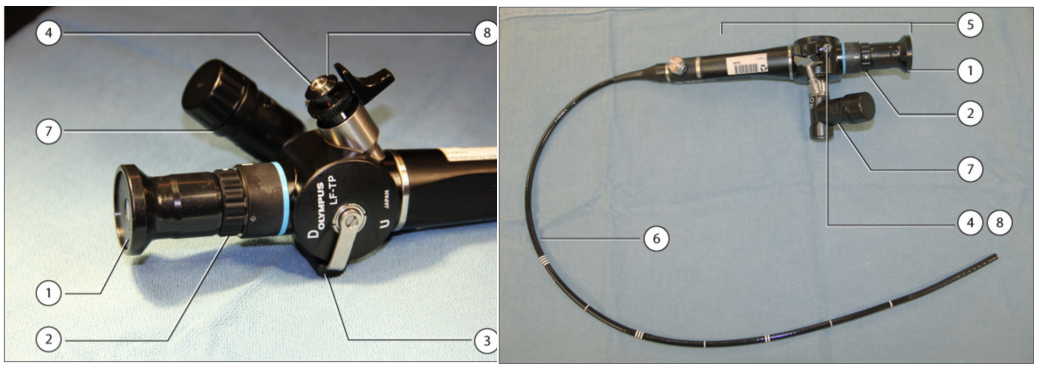


FIGURA 2.16: Partes de un fibrobroncoscopio: (1) Pieza ocular, (2) Anillo de dioptría, (3) Palanca de control, (4) Orificio para el canal de trabajo, (5) Cuerpo, (6) Fibra de inserción, (7) Fuente lumínica y (8) Válvula de succión [9]

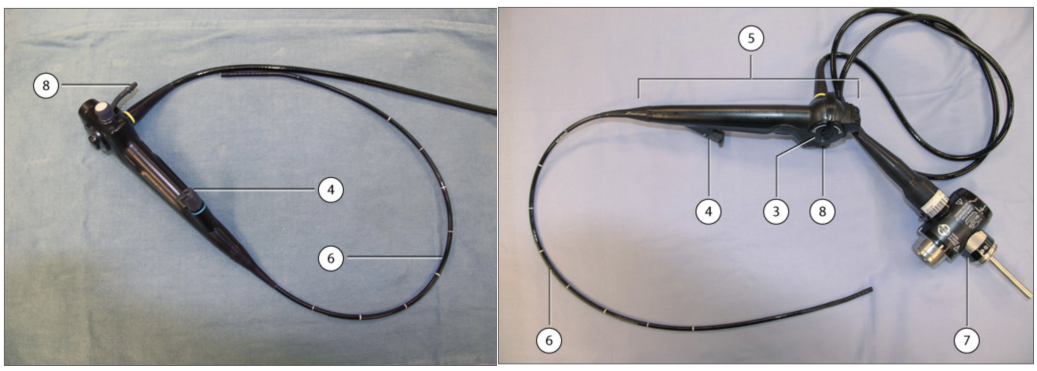


FIGURA 2.17: Partes de un videobroncoscopio: (3) Palanca de control, (4) Orificio para el canal de trabajo, (5) Cuerpo, (6) Fibra de inserción, (7) Fuente lumínica y (8) Válvula de succión [9]

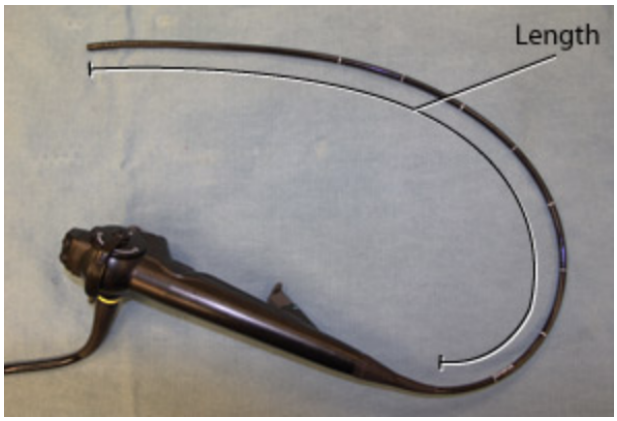


FIGURA 2.18: Largo de un broncoscopio [9]

FIGURA 2.19: Movimiento de angulación [9]

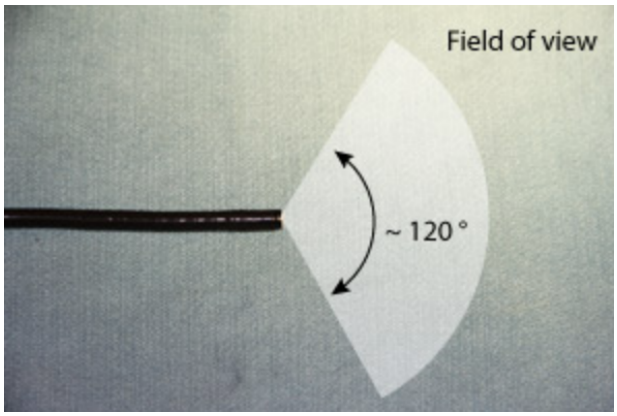


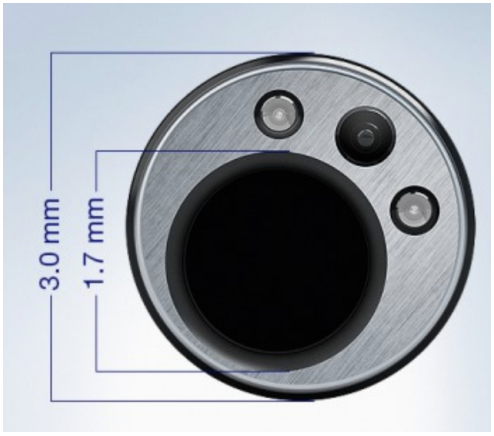
FIGURA 2.20: FOV de un broncoscopio [9]

En cuanto a la porción distal, es posible distinguir diversos elementos que se pueden apreciar en la Figura 2.21. En general, un broncoscopio consta de: fuente lumínica; el canal de trabajo, que posee un diámetro determinado, y del cual se desprende el tamaño de los diversos instrumentos de trabajo que pueden pasar a través de él; cámara de video o fibras ópticas.

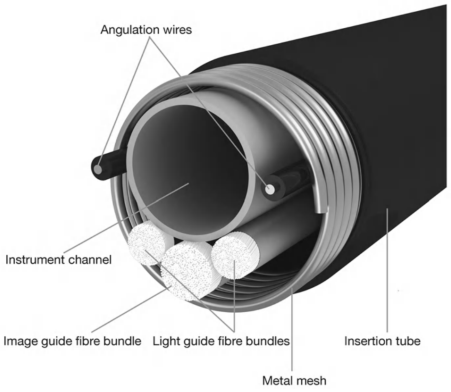
Caracterización de los movimientos de un broncoscopio flexible

Con el fin de comprender los movimientos posibles de un broncoscopio flexible, se los clasifica de la siguiente manera:

- **Movimiento de avance:** movimiento de la fibra de inserción hacia adelante y hacia atrás para entrar en las vías respiratorias (eje Z de la Figura 2.23).



(A) Vista frontal del broncoscopio Olympus BF-MP290F



(B) Componentes de la porción distal de un broncoscopio

FIGURA 2.21: Porción distal

Image[a]	Year	Type	Working length (mm)	External diameter (mm)	Internal diameter (mm)	Tip angulation (up/down)	Additional imaging techniques	Instruments
nF	1984	Olympus BF-1.8T	950	1.8	–	–	–	–
F	1990	Olympus BF-2.2T	1150	2.2	–	120°/120°	–	–
F	1994	Olympus BF-2.7T	1200	2.7	0.8	120°/120°	–	Brush
F	1999	Olympus BF-XP40	600	2.8	1.2	180°/130°	–	
F	2004	Olympus BF-XP60	600	2.8	1.2	180°/130°	–	Brush and forceps
H	2004	Olympus BF-XP160F	600	2.8	1.2	180°/130°	–	
V	2014	Olympus BF-XP190	600	3.1	1.2	210°/130°	NBI	
H	2015	Olympus Y-0025[b]	600	3.0	1.7	180°/130°	–	Brush, forceps, and radial EBUS probe

[a] *F* fiber-optic bronchoscope, *H* hybrid bronchoscope, *V* video bronchoscope
[b] Prototype

FIGURA 2.22: Evolución temporal de los parámetros de un broncoscopio flexible) [8]).

- **Movimiento de rotación:** giro del broncoscopio alrededor de su dirección de avance, generado por la rotación de la fibra de inserción por parte del operario (α en la Figura 2.23).
- **Movimiento de angulación:** denota el movimiento hacia arriba y hacia abajo, generado por la palanca de control (ϑ en la Figura 2.23)

Los primeros dos movimientos, de avance y de rotación, no tienen límites, mientras que el movimiento de angulación sí tiene. Se puede modelar dicho comportamiento generando una estructura de eslabones contiguos con límites en el ángulo de movimiento que tienen entre cada uno de ellos, conformando una cadena (ángulo ϑ en la Figura 2.23). El largo de cada eslabón y

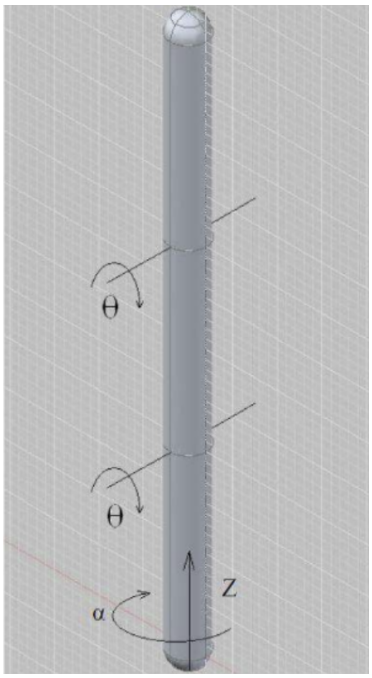


FIGURA 2.23: Modelo de caracterización del movimiento de la fibra de inserción de un broncoscopio [10]

grado de flexión de los mismos dependen de la sección del tubo de inserción que se busque representar.

2.3.3. Procedimientos

Procedimientos diagnósticos

La broncoscopía es una de las herramientas más importantes en el diagnóstico de enfermedades del árbol bronquial. Más aún, los avances en técnicas diagnósticas como la ecografía y la tomografía se trasladaron a este procedimiento, abarcando un mayor abanico de instrumentos que el médico puede utilizar. A continuación se describen las técnicas más relevantes [59]:

- **Punción aspiración transbronquial**: esta modalidad permite obtener muestras de tejido bronquial en niveles más profundos del tejido para ser evaluadas por un patólogo. Se encuentra especialmente indicada ante sospechas de linfoadenopatías, lesiones endobronquiales, nódulos o masas periféricas, patologías de la submucosa, entre otras [60]. El procedimiento se lleva a cabo con un broncoscopio que contiene una aguja fina de aspiración y puede ser combinado con técnicas ecográficas [61].
- **Biopsia con fórceps**: la biopsia con fórceps puede ser utilizada tanto en broncoscopía rígida como flexible. Involucra el uso de una pinza con el fin de obtener una muestra de tejido endobronquial que puede observarse endoscópicamente [11]. Existen diferentes tipos de fórceps como por ejemplo, el de tipo cocodrilo, el ahuecado, o con copa oval, que se pueden observar en la Figura 2.24. La elección del instrumento a utilizar suele estar sujeta a la experiencia y preferencia del operador.
- **Lavado bronquial/broncoalveolar**: el lavado se utiliza con el fin de obtener información tanto diagnóstica como prognóstica [11]. Consiste en instilar suero salino por el canal de trabajo del broncoscopio flexible, con el fin de recolectar muestras celulares que dicho suero arrastre. Este fluido es aspirado con la misma jeringa con la que fue introducido, teniendo especial cuidado en no colapsar las paredes bronquiales ejerciendo una presión inadecuada [62].
- **Ultrasonido endobronquial/ecobroncoscopía**: técnica conocida como EBUS, utiliza el ultrasonido en conjunto con un broncoscopio para visualizar estructuras de la vía aérea [63]. Su transductor puede ser lineal, emitiendo ondas de ultrasonido con una frecuencia de entre 5 y 7,5 MHz (campo visual de 50 a 90 grados), o radial, cuya frecuencia se encuentra entre los 12 a 20 MHz y aporta un campo visual de 360 grados [12]. Esta técnica se utiliza en acompañamiento a la punción transbronquial ya que permite ubicar la estructura a punzar (Figura 2.25).
- **Broncoscopia de autofluorescencia (BAF)**: se utiliza para identificar lesiones pre-cancerígenas, ya que el tejido pulmonar sano fluoresce verde ante la exposición a luz en el espectro del azul-violeta (400-450 nm), como se observa en la Figura 2.26. El tejido patológico decrece en fluorescencia debido a la degradación de sus cromóforos (moléculas capaces de absorber energía en el espectro visible), por lo que es posible identificar visualmente alteraciones celulares tales como instancias previas a tumores [13].
- **Criobiopsia pulmonar transbronquial (TBLC)**: se utiliza una criosonda con el fin de obtener una biopsia pulmonar, introduciéndola hasta el área seleccionada con ayuda de un broncoscopio flexible. El tejido congelado se une a la criosonda y se retira en conjunto con la misma [64].

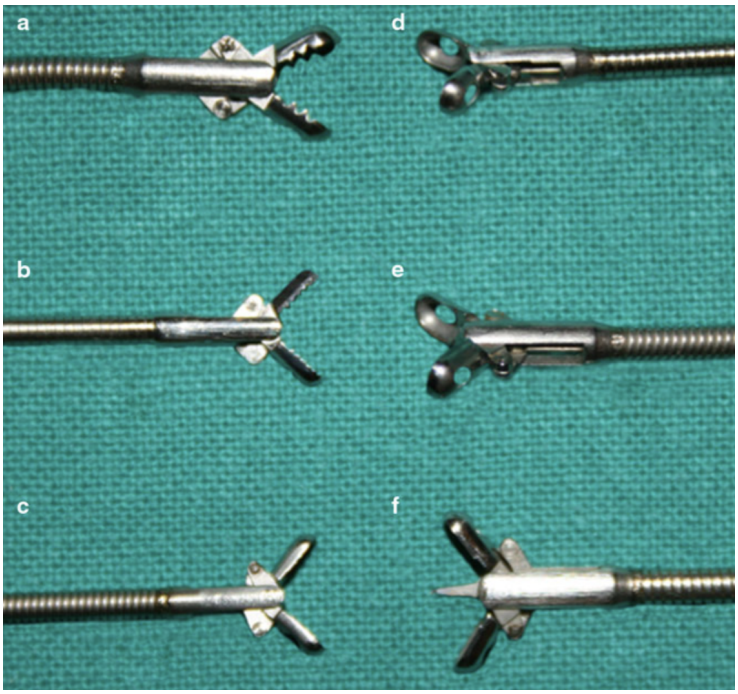


FIGURA 2.24: Tipos de fórceps para biopsia: (a y b) pinza cocodrilo; (c) copa oval con extremo cortante; (d y e) copa cocodrilo con extremo cortante; (f) con una aguja entre copas [11]

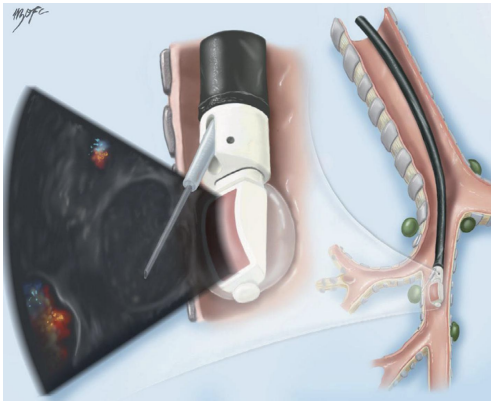


FIGURA 2.25: Sonda para ultrasonido endobronquial en conjunto con aguja de punción [12]

Procedimientos terapéuticos

Los procedimientos terapéuticos son aquellos que implican el tratamiento de cierta patología de la vía aérea. En la tabla de Figura 2.27 se enumeran los principales tipos de procedimientos terapéuticos y sus diferentes abordajes. Los abordajes se describen a continuación:

- **Dilatación**: La dilatación de una zona de angostamiento es una de las prácticas más comunes. Puede ser abordada con diversos mecanismos, siendo el más común la insuflación de un balón. También puede realizarse con un broncoscopio rígido o con fórceps [14].
- **Ablación/coagulación con láser**: se aprovecha la emisión direccionada, en fase y monocromática del láser para coagular la lesión endobronquial mediante el efecto térmico que se produce del encuentro entre la emisión y el tejido [11]. En la tabla de la Figura 2.28

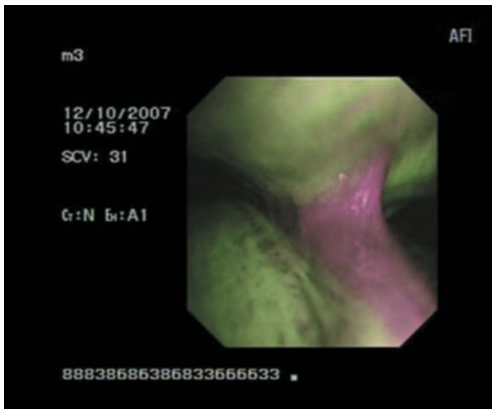


FIGURA 2.26: Carina de color verde, cuyo epitelio es normal, con regiones en color rosado que implican tejido en etapa previa al carcinoma [13]

Technique for Endobronchial Intervention	
Dilatation	Balloon
	Bougies
	Forceps
	Rigid or flexible bronchoscope (blunt dissection under direct vision)
Tissue destruction	Nd-Yag laser
	Argon plasma
	Electrocautery
	Cryotherapy
	Microdebridement
Stent deployment	Short term e.g. silicone stents
	Long term e.g. expandable metallic stents
Haemostatic techniques	Local adrenaline (liquid/gauze)
Airway isolation	
Clot removal	
Biopsy	
Foreign body retrieval	
Closure of fistulae e.g. using glue	

FIGURA 2.27: Tipos de intervenciones y sus técnicas [14]

se describen los distintos tipos de láser y sus propiedades características. Uno de los más utilizados en la broncoscopía es el láser de Nd:Yag (granate de itrio y aluminio dopado con impurezas de neodimio) ya que puede utilizarse con una fibra larga de cuarzo debido a su longitud de onda.

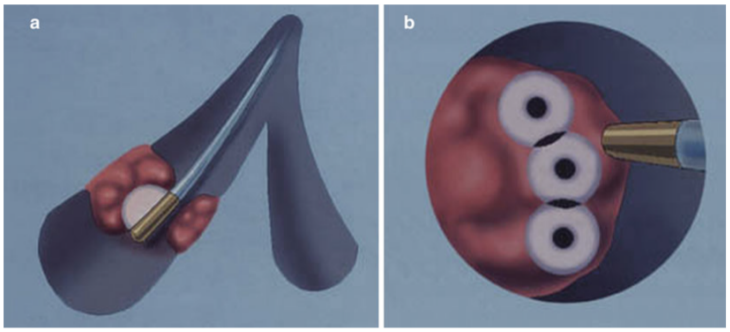
- **Crioterapia**: la crioterapia implementa las bondades de la aplicación de frío a tejidos enfermos para que sus células se destruyan mediante la formación de cristales de hielo intracelulares [11]. En esta técnica se utiliza una criosonda que es introducida en el canal de trabajo del broncoscopio. En la Figura 2.29a se observa el extremo distal de una criosonda

en contacto con el tejido que se desea tratar. En la Figura 2.29b se observa la remoción de un carcinoma en la vía aérea, previo tratamiento con crioterapia.

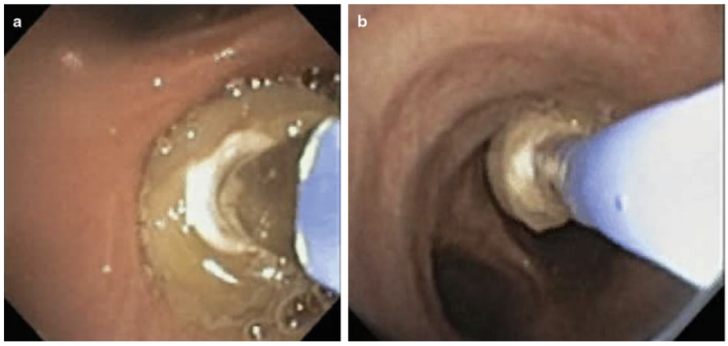
- **Colocación de *stent***: puede ser colocado tanto con un broncoscopio rígido como flexible y provee de soporte estructural a la vía aérea ante el diagnóstico de una estenosis, obstrucción o cicatrización de una fístula [14]. Existen distintos tipos de *stents*, que se pueden observar en la Figura 2.30. En la Figura 2.31a se observa la colocación de un *stent* y en la Figura 2.31b el proceso de remoción de un stent de silicona mediante el uso de un broncoscopio rígido como medio de intubación y de acceso. Ambos, el tubo y el *stent*, se retiran en conjunto.
- **Colocación de válvula endobronquial**: el EPOC genera una *compliance* o distensión aumentada del tejido pulmonar, y, en consecuencia, un atrapamiento de gas (ya que el tejido debilitado no puede expulsar ese aire en su completitud durante la exhalación). La hiperinsuflación tiene otras consecuencias mencionadas con anterioridad en la caracterización de las patologías de la vía aérea [65]. En algunos casos es posible aislar la región pulmonar más afectada mediante la colocación de válvulas unidireccionales que previenen el ingreso de aire en la región enferma, mientras que dejan que el aire y los fluidos atrapados en la misma salgan de ella. De este modo, las regiones restantes pueden hacer un mejor uso del espacio torácico y de este modo se mejora la función respiratoria [15]. La Figura 2.32 muestra el proceso de colocación.
- **Electrocauterización**: la técnica de electrocauterización es análoga a la coagulación con láser, con la diferencia de que en la electrocauterización se utiliza una corriente eléctrica que genera calor, con el fin de cauterizar [14].
- **Terapia fotodinámica**: implica la aplicación específica por vía intravenosa de agentes fotosensibles específicos a tumores, o que tengan toxicidad selectiva a la vasculatura del tumor. Se activan mediante un láser monocromático generando, en consecuencia, la necrosis tumoral [13]. Algunos ejemplos de estos agentes son la Fotofrina, Benzoporfirina, entre otros [60].

					Uses	
Laser	Wavelength (nm)	Delivery device	Scope	Depth of penetration	Coagulation	Cutting
Nd:YAG	1,060	Quartz fiber	RB/FB	0.5–1.5 cm	+++	+
CO_2	10,600	Coupler and waveguide	RB	0.23 mm	+	+++
Argon	516	Quartz fiber	RB/FB	1.0–2.0 mm	++	+
KTP	532	Quartz fiber	RB/FB	1.0 mm	++	+
Nd:YAP	1,340	Quartz fiber	RB/FB		++++	+

FIGURA 2.28: Características de los principales lásers [11]



(A) Crioterapia en un tejido de la vía aérea [11]



(B) Crioterapia en un carcinoma, previo y post tratamiento [11]

FIGURA 2.29: Crioterapia

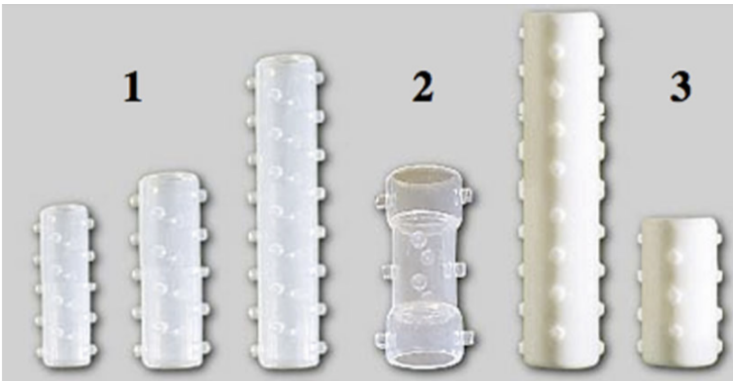
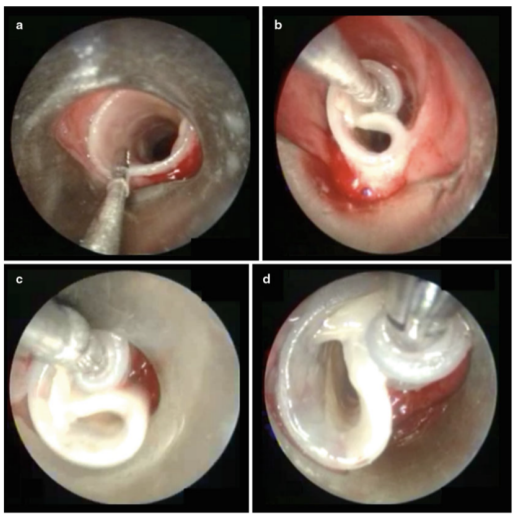


FIGURA 2.30: Modelo Dumon de *stent* y sus posibles variantes: radiotransparente, con forma de reloj de arena y radioopaco [11]

2.3.4. Visita al Hospital de enfermedades infecciosas "Dr. Francisco Javier Muñiz"

Se realizó una visita al Hospital de enfermedades infecciosas "Dr. Francisco Javier Muñiz", ubicado en la calle Uspallata 2272 de la Ciudad Autónoma de Buenos Aires, con el fin de presenciar una intervención de broncoscopía realizada por el Dr. Duré. Durante la visita no se intervino en la situación para no interferir en la actividad y así poder registrar lo sucedido de manera fidedigna. Se realizó un relevamiento de los actores clave y usuarios potenciales en el recorrido completo durante la intervención, y se estudiaron in situ los siguientes aspectos:

(A) Colocación de un stent con ayuda de un broncoscopio rígido, que ensancha la estenosis en primer lugar, para su posterior colocación [11]



(B) Remoción de un *stent* de silicona [11]

FIGURA 2.31: Colocación y remoción de un stent

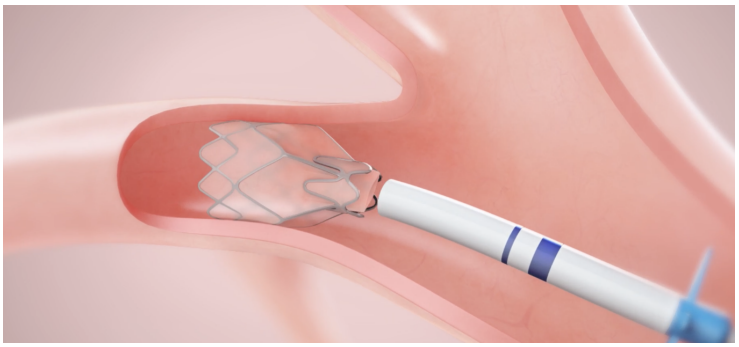


FIGURA 2.32: Válvula endobronquial Zephyr® y su colocación con un broncoscopio flexible en la región a aislar [15]

Personas involucradas: Médico especialista en neumología, asistente y dos residentes.

Objetos: Fibrobroncoscopio Pentax FB-15V con las siguientes características (en las Figuras 2.33 y 2.34 se detalla la medición de estos parámetros):

- Diámetro del canal de trabajo: 2.2 mm
- Diámetro del extremo distal: 4.9 mm
- Largo de trabajo: 600 mm
- Angulación hacia arriba/abajo del extremo distal del tubo de inserción: 180/130 grados
- Campo de visión (FOV): 120 grados

Situaciones y actividades: Extracción de un *stent* bronquial de 10 años de antigüedad que se había colocado en un paciente masculino con estenosis consecuencia de complicaciones durante una intubación prolongada. La zona del *stent* presenta infección debido al excesivo período de tiempo presente del cuerpo extraño en la vía aérea, en conjunto con una tuberculosis reciente. El paciente recibe anestesia general debido a la complejidad del procedimiento y se le coloca un broncoscopio rígido. Se monitorean de manera constante sus signos vitales, especialmente el porcentaje de saturación de oxígeno. El *stent* logra retirarse con la ayuda visual de un broncoscopio flexible que es insertado dentro del rígido, y una pinza, luego de dificultades debido a la severidad del caso. El sujeto queda en observación en el servicio de internación del Hospital Municipal.

Duración de la actividad: 30-60 minutos

FIGURA 2.33: Fibrobroncoscopio Pentax FB-15V del Hospital Muñiz

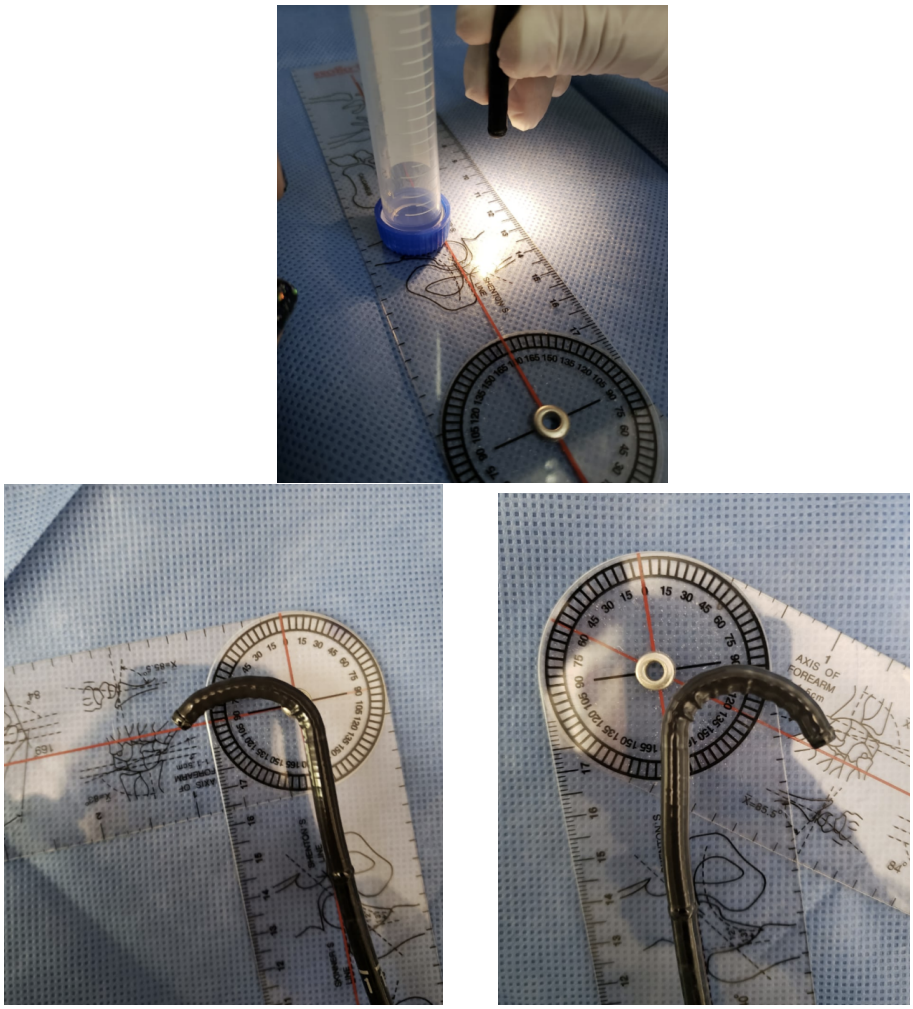


FIGURA 2.34: Mediciones realizadas al Pentax FB-15V. La porción distal del tubo de endoscopio la cual es flexible y se controla con el cuerpo se corresponde a 40 mm.

2.4. Simulación y educación

En el afán de la educación médica instruir al estudiante en la medicina, existen diversas herramientas que pueden fortalecer este camino. Esto es especialmente relevante en el aprendizaje de prácticas y procedimientos, ya que el estudiante debe enfrentarse con el desafío de aprender un ámbito real. Más aún, al tratarse de la salud humana, estas experiencias pueden ser incómodas y estresantes tanto para el alumno como para el docente, que debe abstenerse intencionalmente de interferir en ciertos procedimientos, con el fin de que el alumno las aprenda [11]. En la visita realizada en el Hospital Muñiz, uno de sus residentes expresó haberse sentido cómodo en el uso de un fibrobroncoscopio luego de 50 prácticas aproximadamente, explicitando la importancia de la práctica en el aprendizaje médico. Lo que es más, en el año 2003 el *American College of Chest Phyicians* estableció un número promedio de 100 intervenciones para que un practicante alcance un nivel de competencia básico, bajo el modelo "veo un procedimiento, realizo un procedimiento"[66]. Este modelo tiene como desventaja la carencia de metodología, ya que el instructor debe adecuarse a los casos reales que se presentan en la práctica cotidiana [11].

Una alternativa para sumar práctica metodológica se corresponde con la incorporación de herramientas de simulación en el contexto educativo. La simulación aporta herramientas que permiten el perfeccionamiento de técnicas invasivas y quirúrgicas, y es definida como “el proceso de diseñar un modelo de un sistema real [...] con la finalidad de comprender su comportamiento [...] o evaluar nuevas estrategias dentro de los límites impuestos"[67]. La simulación médica permite, en primer lugar, tener una experiencia inicial controlada de una práctica, por lo que el humano no se convierte en el primer sujeto de prueba en el conocimiento de los elementos de trabajo. Asimismo, su práctica contribuye a entrenar la minuciosidad de los movimientos disminuyendo traumas innecesarios en la vía aérea, creando oportunidades de trabajo donde las tareas pueden practicarse repetidamente. Otra ventaja de la simulación es que la educación puede adecuarse a las necesidades pedagógicas individuales del estudiante, mejorando su curva de aprendizaje [68].

En el caso particular del software desarrollado en el presente trabajo, el beneficio suple la necesidad principal de un residente entrevistados en la visita al Hospital Alemán, quien nunca había realizado una intervención hasta el momento, y cuyo mayor desafío es aprender la anatomía de las vías aéreas. Actualmente el residente asiste a intervenciones e interactúa verbalmente con el médico que realiza la broncoscopía; este último le pregunta al residente en reiteradas ocasiones la ubicación de dónde se encuentra la punta del broncoscopio y qué movimiento haría a posterior, para probar el conocimiento del residente y que se vaya familiarizando con la técnica.

Lo que es más, los simuladores pueden ser utilizados con fines de estandarizar la evaluación práctica de los profesionales, como su utiliza en la A.A.A.R.B.A. (Asociación de Anestesia Analgesia y Reanimación de Buenos Aires). En adición, la simulación no representa una experiencia traumática para el alumno: en SIMMER se utiliza como política educativa el hecho de no dejar morir a un paciente, sino frenar la práctica antes de dicho suceso. Otras ventajas incluyen el perfeccionamiento del manejo de las relaciones interpersonales en un contexto de presión, como lo son la escucha activa, el liderazgo, toma de decisiones y coordinación de equipo.

2.4.1. Tipos de simuladores en el mercado

La simulación puede clasificarse según su nivel de complejidad: simulación de baja fidelidad o de alta fidelidad [11].

Los modelos de baja fidelidad son una alternativa de bajo costo que permiten al operador conocer la anatomía de la vía aérea mientras practica técnicas manuales que incorporan en el bagaje de aprendizaje mediante repetición. Sin embargo, estos modelos no aportan posibilidad de interacción ni variaciones situacionales, por lo que son una buena herramienta inicial, pero con acotadas posibilidades. La Figura 2.35 ejemplifica un simulador de baja fidelidad.

Los modelos de alta fidelidad, por otra parte, utilizan tecnología anatómica virtual tridimensional y retroalimentación háptica, en muchas ocasiones [11]. Esto último implica que el usuario percibe una devolución sensorial táctil de la situación virtual en la que se encuentra inmerso. La simulación de alta fidelidad tiene varias ventajas: simula la interacción basada en escenarios que imitan situaciones de la vida real en la cual es posible repetir los movimientos realizados y, en consecuencia, aprender por repetición; permite la capacitación en un entorno seguro libre de estrés; posibilita la exposición a escenarios raros o difíciles; devuelve comentarios inmediatos

FIGURA 2.35: Simulador de baja fidelidad [11]

sobre el rendimiento que no son frustrantes, ya que si el resultado de la práctica es negativo, no se daña a un paciente real. Por otra parte, tienen como desventaja ser costosos, y que todavía existe apego al modelo tradicional de aprendizaje y dudas sobre los beneficios de la simulación [11]. En el Hospital Muñiz, por ejemplo, sugieren la presencia de simuladores in situ, ya que entienden que la simulación situada en un contexto real puede ser potenciada mediante la sinergia entre las situaciones reales y virtuales.

A continuación se describen los simuladores de alta fidelidad más conocidos:

EndoVR bronchoscopy (CAE Healthcare, Montreal, Quebec, Canadá)

El simulador EndoVR bronchoscopy permite que el usuario se familiarice con la broncoscopía flexible y su equipamiento en situaciones análogas a las reales. Profundiza en las técnicas de lavado bronquial, punción aspiración transbronquial y biopsiado endobronquial. En la Figura 2.36 se lo ilustra.

FIGURA 2.36: Simulador EndoVR, de la compaña CAE Healthcare [16]

GI-BRONCH Mentor (3D Systems-Symbionix)

El simulador BRONCH Mentor (Figura 2.37) también aborda a la broncoscopía flexible y está destinado a a entrenar médicos de terapia intensiva, anestesiólogos y médicos especialistas en broncoscopía intervencionista. Se jacta de proveer herramientas para desarrollar habilidades motoras, de coordinación y cognitivas. El Centro de Simulación SIMMER (Simulación Médica Roemmers), ubicado en Olivos, Buenos Aires, Argentina, cuenta con un simulador GI-BRONCH que pudo ser probado gracias a la gentileza del centro.

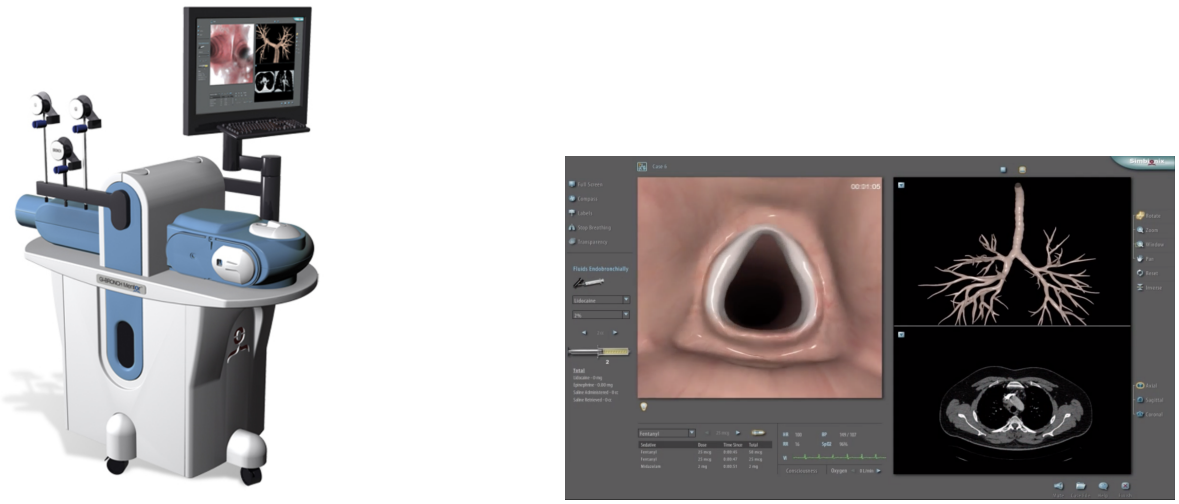


FIGURA 2.37: Hardware y UI del simulador GI-BRONCH Mentor [17]

ORSIM (Airway Limited, South Yorkshire, UK)

El similador ORSIM, de la firma Airway Limited, es un sistema compacto y portable, pero a la vez robusto, ya que resiste las numerosas prácticas que sufre una fibra de broncoscopía. También ofrece gráficos de calidad en conjunto con resultados de rendimiento a modo de que el usuario pueda mejorar su desempeño con el correr de las prácticas (Figura 2.38). La Asociación de Anestesia Analgesia y Reanimación de Buenos Aires cuenta con un simulador ORSIM, el cual fue probado.

FIGURA 2.38: Simulador portable ORSiM [18]

Thoracic Anesthesia Bronchoscopy Simulator (University Health Network)

Este simulador consiste en un software desarrollado para conocer y aprender acerca de la anatomía de la vía aérea. Es un simulador gratuito al que se puede acceder desde el siguiente enlace web: http://www.thoracic-anesthesia.com/ (Figura 2.39).

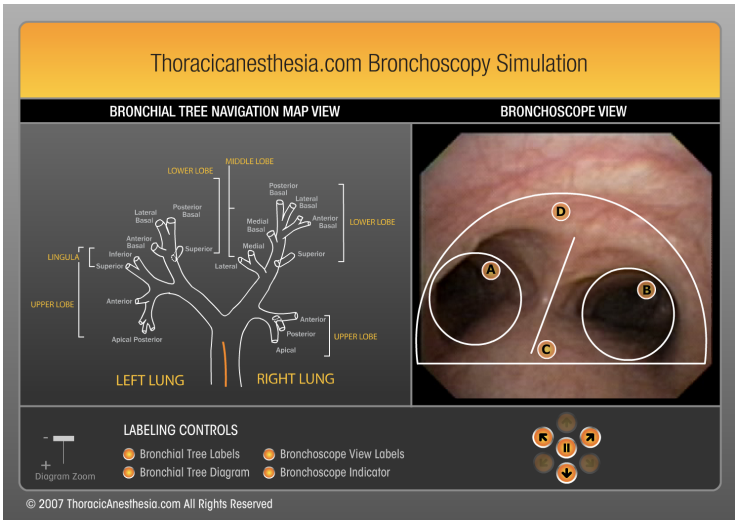


FIGURA 2.39: Thoracic Anesthesia Bronchoscopy Simulator [19]

BronchoGuide (Olympus Europa SE & Co. KG)

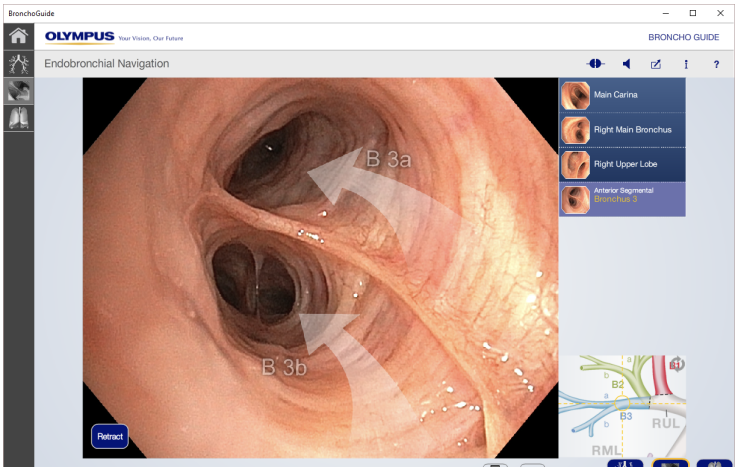


FIGURA 2.40: Display del software de simulación BronchoGuide, de Olympus [20]

Aplicación desarrollada por la firma Olympus con el fin de replicar virtualmente recorridos de examinaciones broncoscópicas, comenzando desde la carina hacia el resto de la vía aérea. Posee comentarios en audio y visuales con el fin de entrenar la orientación espacial del usuario (Figura 2.40).

Capítulo 3

Marco metodológico

3.1. Introducción

El uso de un broncoscopio rígido o uno flexible depende del tipo de diagnóstico o terapia que se necesite realizar. Sin embargo, utilizar el broncoscopio rígido posee la desventaja de no alcanzar las bifurcaciones bronquiales ni generaciones más profundas, y la visión limitada al eje bronquial principal. Es por eso que actualmente la broncoscopía flexible está adquiriendo mayor importancia. La misma posee mayor accesibilidad a los bronquios segmentarios y ramas, con la posibilidad de obtener imágenes que permiten tener una visión más detallada, y ofrece maniobras endoscópicas de precisión. Generalmente, si se usa el broncoscopio rígido, se implementa de manera complementaria con el flexible, como fue el caso de la intervención presenciada en el Hospital Muñiz [69].

En el caso particular del presente trabajo se simula la imagen proveniente de un broncoscopio flexible en una torre de endoscopía. A su vez, se simula únicamente un tipo de lesión endobronquial denominada granuloma, una pequeña masa de tejido con tamaño y forma variada, formada por células aglomeradas que aíslan algún tipo de elemento extraño que el cuerpo no puede eliminar [70]. Se incluye una pinza de biopsia y la posibilidad de expulsar agua para el lavado.

La simulación se realiza en el motor de videojuegos Unity3D y el proyecto se comparte en GitHub. GitHub es una herramienta de desarrollo colaborativo utilizado para crear y compartir código fuente en donde cada proyecto posee su propia página web y funcionalidades que permiten a muchos programadores trabajar un mismo repositorio de manera colaborativa y *online*. Asimismo, las estructuras anatómicas utilizadas son segmentadas (aisladas) de tomografías computadas de tórax, con el fin de ser incorporadas en el motor de videojuegos.

La jugabilidad de la simulación es en formato tridimensional, es decir que los objetos pueden moverse en todas las direcciones. Los gráficos también están en 3D, es decir que los objetos en vez de ser dibujos, son cuerpos modelados. El eje principal circunda alrededor de un comportamiento físico convincente, por lo que se busca modificar los componentes de Unity que manejan la simulación física mediante aceleraciones, colisiones, gravedad, y otras fuerzas, de manera precisa y controlada. De esta forma se trata de aproximar la simulación a los movimientos reales de un broncoscopio.

La fase de familiarización de la plataforma Unity consistió en la realización de diversos ejercicios simples consignados por los tutores. Ejemplos de los primeros fueron la creación de objetos

primitivos, como por ejemplo cubos y esferas, y sus movimientos por posiciones y por fuerzas con las flechas del teclado. Se simularon distintos casos como por ejemplo colisiones entre objetos y modificaciones de tamaños.

3.2. Segmentación de modelos en 3DSlicer

El 3D Slicer es un programa Open Science, es decir que es Open Source, Open Data y Open Community. Permite la visualización de imágenes médicas y su procesamiento y análisis.
Con el fin de obtener la estructura anatómica de interés, es posible segmentar imágenes de una Tomografía Computada o de Resonancia Magnética. En procesamiento de imágenes, segmentar implica dividir una imagen digital en grupos de píxeles, con el fin de "separar los objetos de interés del resto encontrando los límites del mismo, reteniendo su forma y tamaño"[71]. Existen distintos algoritmos de segmentación, basados en detección de bordes, en reconocimiento de regiones, entre otros.

En este caso se utiliza un *DataSet* de Tomografías Computadas proveniente de fuentes públicas y un curado/cupendio de las mismas (máscaras) que pre-segmentan la vía aérea, realizadas en la cátedra de Procesamiento de Imágenes Biomédicas del Instituto Tecnológico de Buenos Aires. Se utiliza 3D Slicer para segmentar la estructura tridimensional deseada, que debe ser hueca, al igual que la estructura anatómica que simula.

Para ello, se abre la máscara en el software, pudiéndose observar en la Figura 3.1 la pantalla de inicio que ofrece el mismo, con las vistas coronal, sagital y axial en cada uno de los recuadros. El recuadro vacío es el destinado a la reconstrucción tridimensional que será utilizada posteriormente.

Mediante comandos en el módulo de segmentación, se definen los criterios de la misma. Algunos de ellos son:

- ***Threshold***: segmenta el nivel de gris seleccionado por el umbral que se determina.
- ***Region growing***: mediante el uso de semillas e iteraciones, se hace crecer la región segmentada.
- ***Hollow***: ahueca una estructura. Es posible determinar el ancho de la pared y disposición respecto al límite de segmentación, pudiendo definir paredes exteriores, mediales e internas, según conveniencia.
- ***Scissors***: posibilita la realización de cortes customizados con el *mouse*.
- ***Draw/Erase***: pudiendo establecer las dimensiones del trazo, permite dibujar/borrar con el fin de agregar/quitar zonas para considerar en la segmentación o corregir manualmente falencias de los métodos de segmentación automática (*Threshold*, por ejemplo).

3.2.1. Proceso

Con el fin de obtener la estructura tridimensional, es necesario segmentar la estructura anatómica deseada mediante el módulo *Segmentation* y se realiza una nueva segmentación denominada

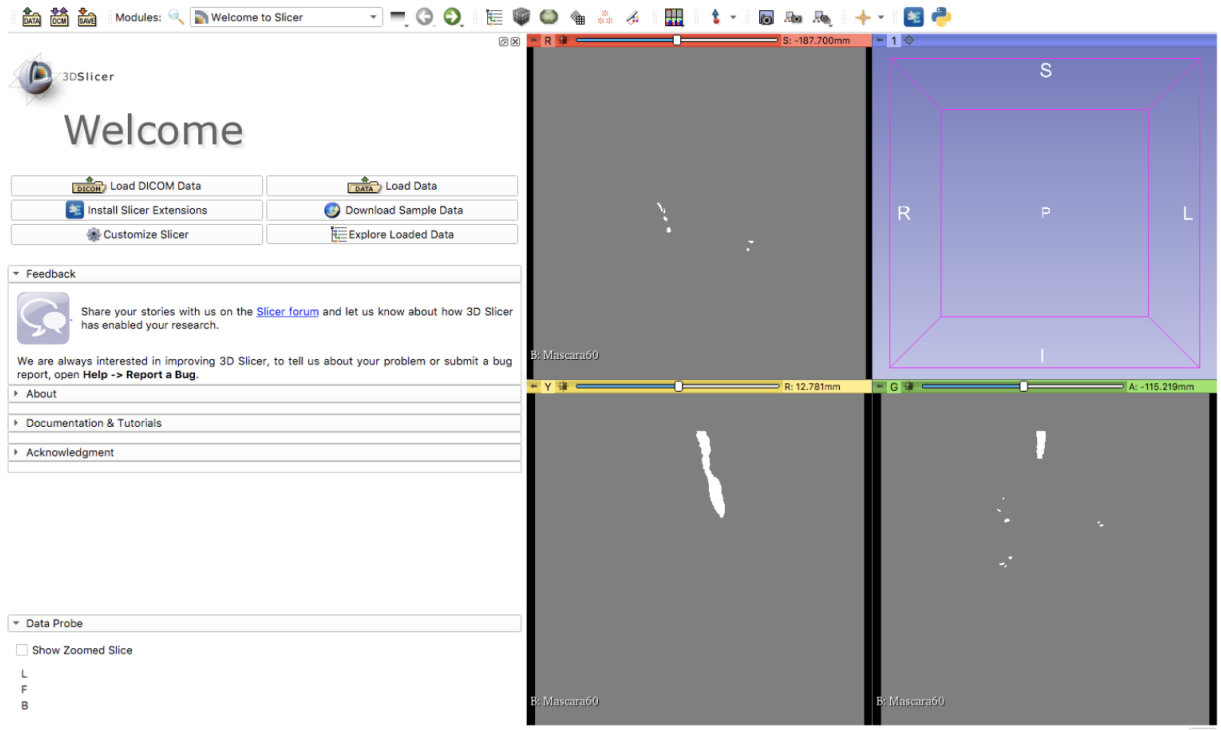


FIGURA 3.1: Pantalla de inicio del 3D Slicer, con la máscara abierta en el programa

Segment_1, caracterizada por el color verde, que es customizable (Figura 3.2). Este rol no es relevante en nuestro trabajo, pero la diversidad de colores cobra importancia a la hora de segmentar estructuras anatómicas diversas, con el fin de poder distinguirlas.

El módulo de segmentación ofrece distintas herramientas, algunas de ellas previamente mencionadas. Dado que la máscara fue provista, en este caso particular el método de segmentado recomendado es el de selección por umbrales o *threshold*. Este método de segmentación por regiones permite seleccionar un rango de nivel de grises que va a ser el que se considerará al procesar la imagen, para aislar dicho rango únicamente.

Las imágenes de Tomografía Computada ofrecen distintos niveles de gris, acordes a la capacidad de absorción de rayos X de cada material que atraviesen. En este caso, las máscaras basadas en imágenes de Tomografía Computada segmentan el aire que se encuentra en la vía aérea y no la estructura anatómica bronquial.

Es posible observar en la Figura 3.3 que, al determinar en el *Threshold* un rango amplio de tonos de gris, la segmentación abarca toda la imagen, en tonalidad verde. La máscara posee la bondad de distinguir en un tono blanco o cercano al blanco a la vía aérea, y en un nivel de gris más oscuro al resto de la imagen. Es por ello que si se limita el umbral a valores cercanos a los niveles de gris más claros, como muestra la Figura 3.4, se puede aislar la estructura bronquial, ya que se encuentra determinada en la máscara por colores claros.

Mediante la herramienta *Show 3D* es posible reconstruir el área segmentada, como se puede ver en la Figura 3.5. La misma, a su vez, deja ver que la reconstrucción es sólida, por lo que se necesita ahuecarla. Para ello se utiliza la herramienta *Hollow*. Ella permite definir áreas segmentadas únicamente en sus bordes, con la posibilidad de establecer grosores customizados para

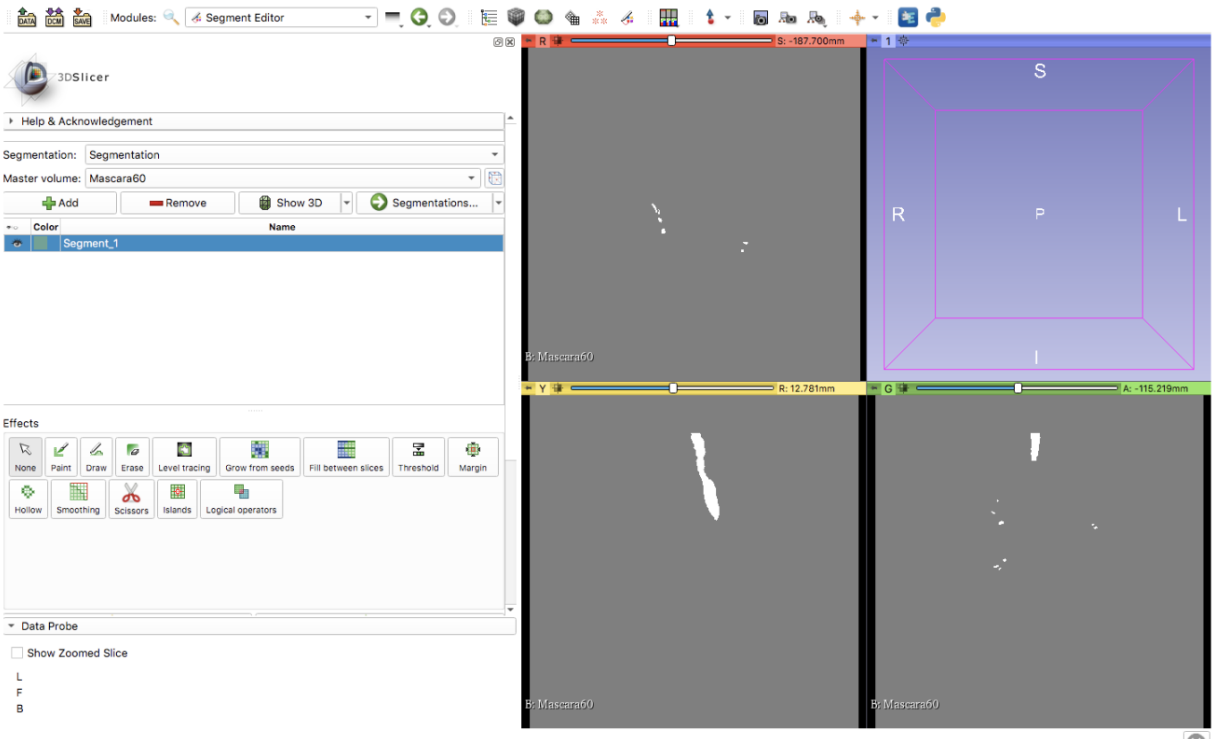


FIGURA 3.2: Pantalla principal del módulo de segmentación de 3DSlicer, que permite visualizar las diversas herramientas de segmentado que el programa ofrece

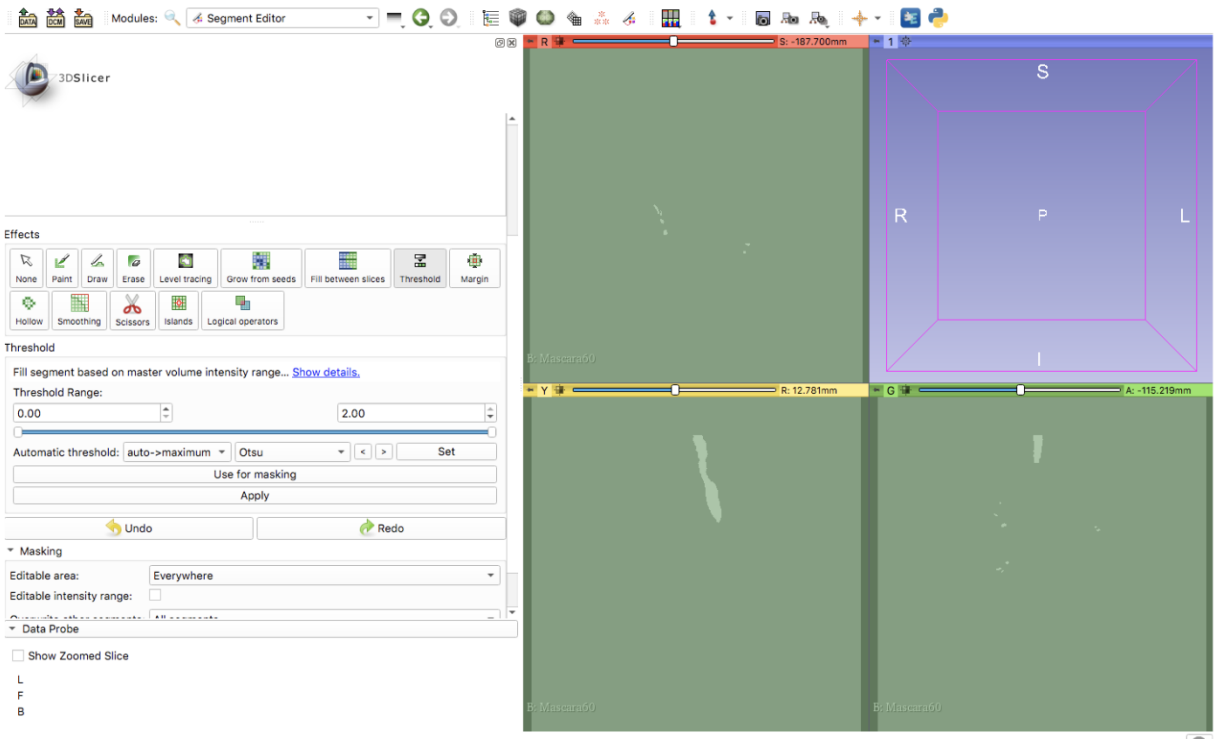


FIGURA 3.3: Herramienta de Threshold con todo el rango de grises establecido (de 0 a 2.0). El valor 2.0 representa los tonos de gris más claros, y el 0 los más oscuros.

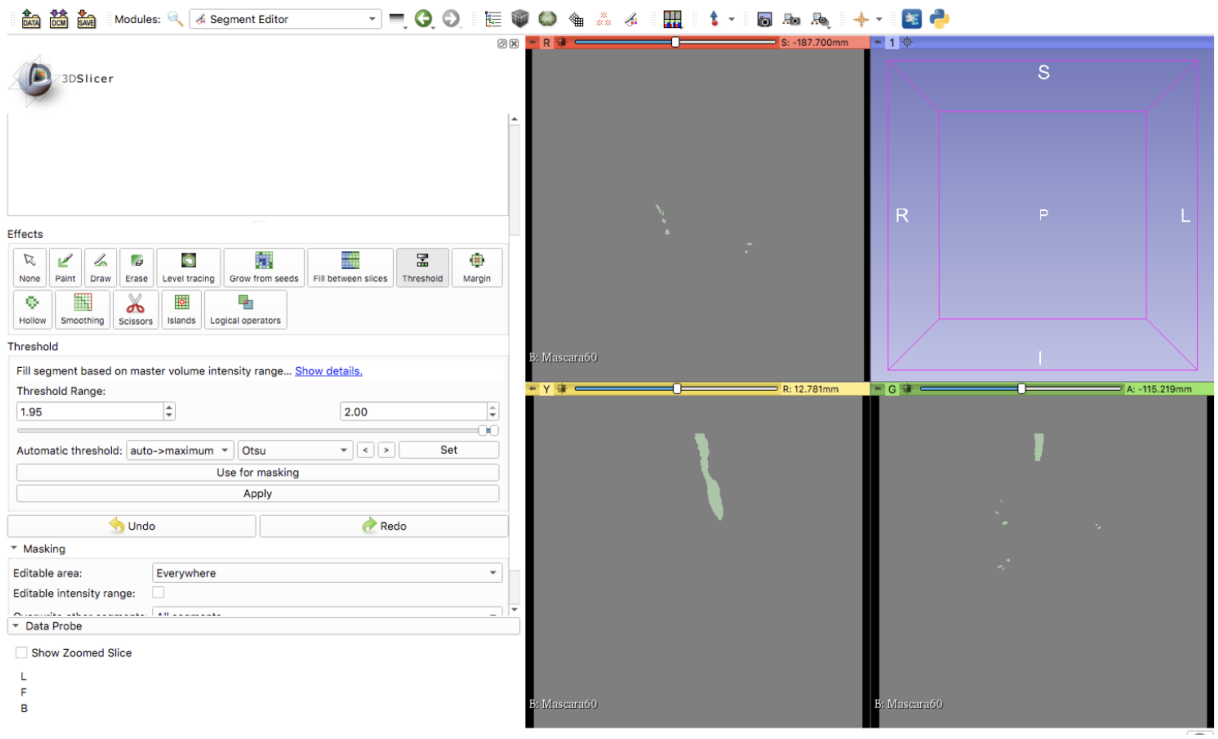


FIGURA 3.4: Herramienta de Threshold con un umbral establecido entre 1.95 y 2.00

este borde, y definir qué parte de la estructura va a representar dicho borde (cara interna, externa o punto medio). Debido a que la reconstrucción segmenta el aire y la estructura bronquial que lo rodea no es segmentada, es correcto definir un borde externo al aire que la represente gráficamente.

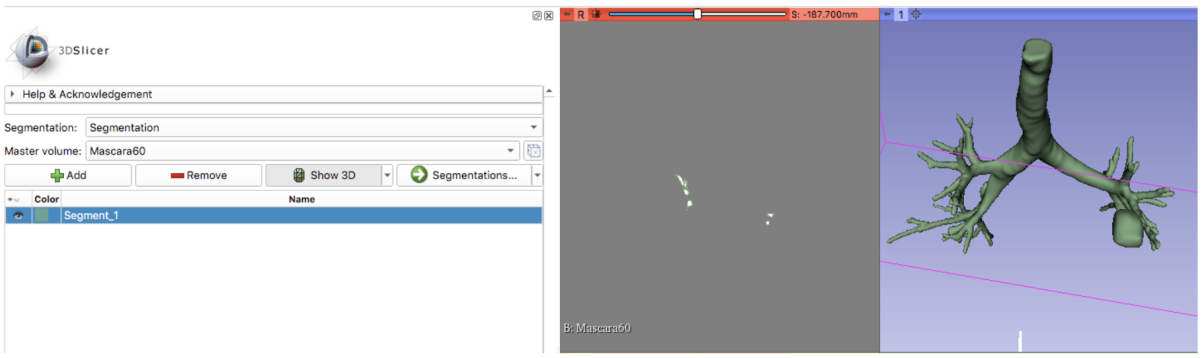
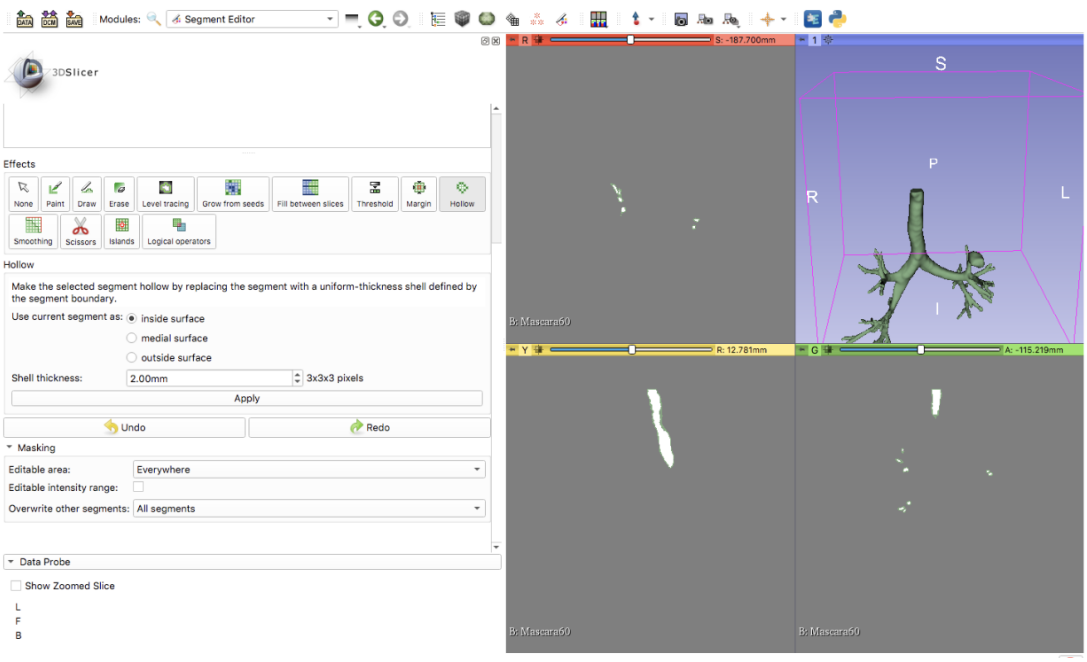


FIGURA 3.5: Visualización de la estructura tridimensional segmentada con umbrales

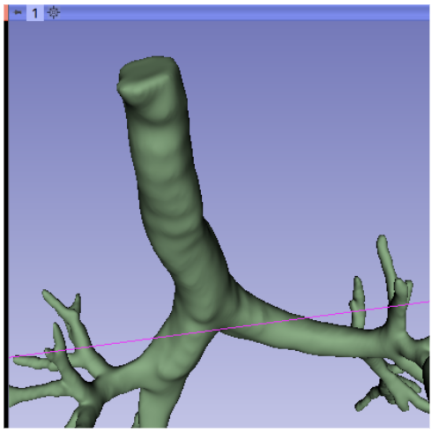
El ancho del contorno se define como el mínimo posible. Esto radica en que la herramienta no permite una disminución gradual del mismo, por lo que los bronquiolos quedan obstruidos si se definen anchos mayores.

Para definirlo, se evalúan distintos anchos. Un ancho de 1.8 mm equivale a un vóxel de 3x2x1 píxeles, que en muchas ocasiones, no es suficiente para establecer una estructura completamente cerrada, por lo que se pueden ver huecos en la misma. Un ancho de 2 mm se computa mediante

un vóxel de 3x3x3 píxeles que sí es suficiente para que 3D Slicer genere una estructura cerrada, por lo que se utiliza este ancho (Figuras 3.6)



(A) Opciones disponibles en la herramienta *Hollow*



(B) Ancho definido en 2.0 mm

(C) Ancho definido en 1.8 mm

FIGURA 3.6: Comparación de las estructuras tridimensionales obtenidas variando el ancho de la herramienta *Hollow*

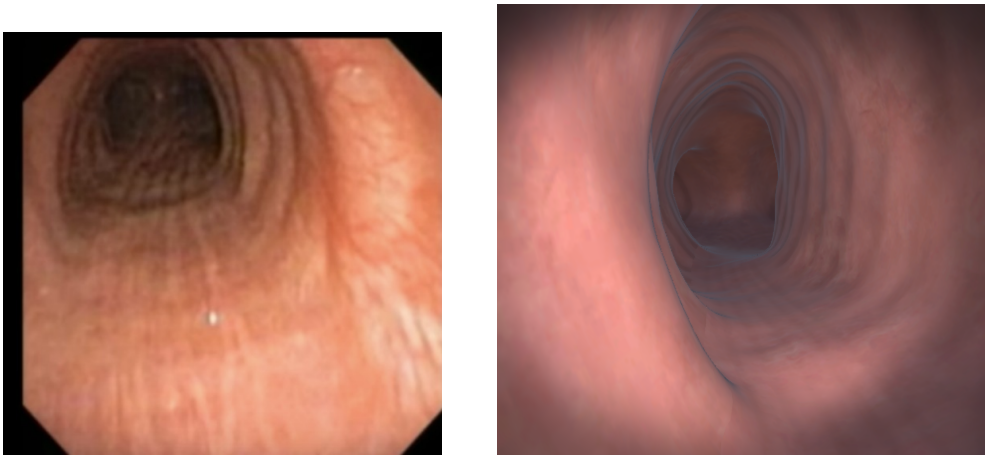
Asimismo, se observa que la vía aérea segmentada y ahuecada posee una tapa, por lo que se utiliza la herramienta *Scissors* para generar la apertura del extremo proximal a la cavidad bucal.

Las estructuras segmentadas son exportadas en formato .obj y se le realizan retoques estéticos previo a la importación al motor de físicas en este formato.

3.2.2. Diseño

Con el fin de realizar retoques estéticos y así aumentar el realismo de los modelos y sus respectivas lesiones endobronquiales, el diseño gráfico de las texturas y morfologías patológicas estuvo

a cargo del diseñador de Sensus3D, Alberto Gentile, en Cinema4d, Zbrush y Adobe Photoshop . Éstas son aplicaciones digitales de modelado 3D, texturizado y renderización de características como iluminación, color, material y profundidad. Se utiliza una textura que se aproxima a la superficie real de las vías, con cierta luminosidad para tratar de representar la mucosa que tapiza. En el mismo se pueden diferenciar los cartílagos hialinos en forma de C en la parte superior y las fibras musculares longitudinales en el extremo abierto de la base.



(A) Vista real (B) Diseño modelado

FIGURA 3.7: Imágenes de Broncoscopía

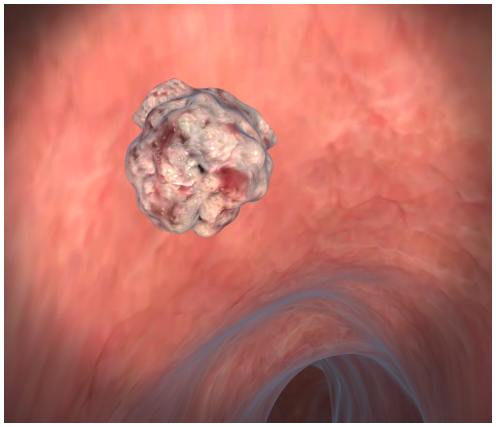


FIGURA 3.8: Diseño de la lesión endobronquial utilizada

3.2.3. Escena principal

3.3. El simulador

3.3.1. Unity 3D

Unity es un motor de videojuegos lanzado en 2005 por la firma Unity Technologies. Esta multiplataforma utiliza como lenguajes de programación C, C++ y C#; en el caso del presente trabajo se utiliza C#. Un motor de videojuegos se compone de una serie de rutinas de programación que se utilizan para diseñar y crear un videojuego. Unity utiliza motores gráficos y físicos para

renderizar (crear una imagen a partir de información digital) y simular las leyes de la física respectivamente, e incluye animación digital y *scripts*, entre otros elementos.

Para poder comprender los distintos elementos que componen el motor de videojuegos, se los define:

Editor

Centro visual con el contenido del juego y funcionalidades para editarlo.

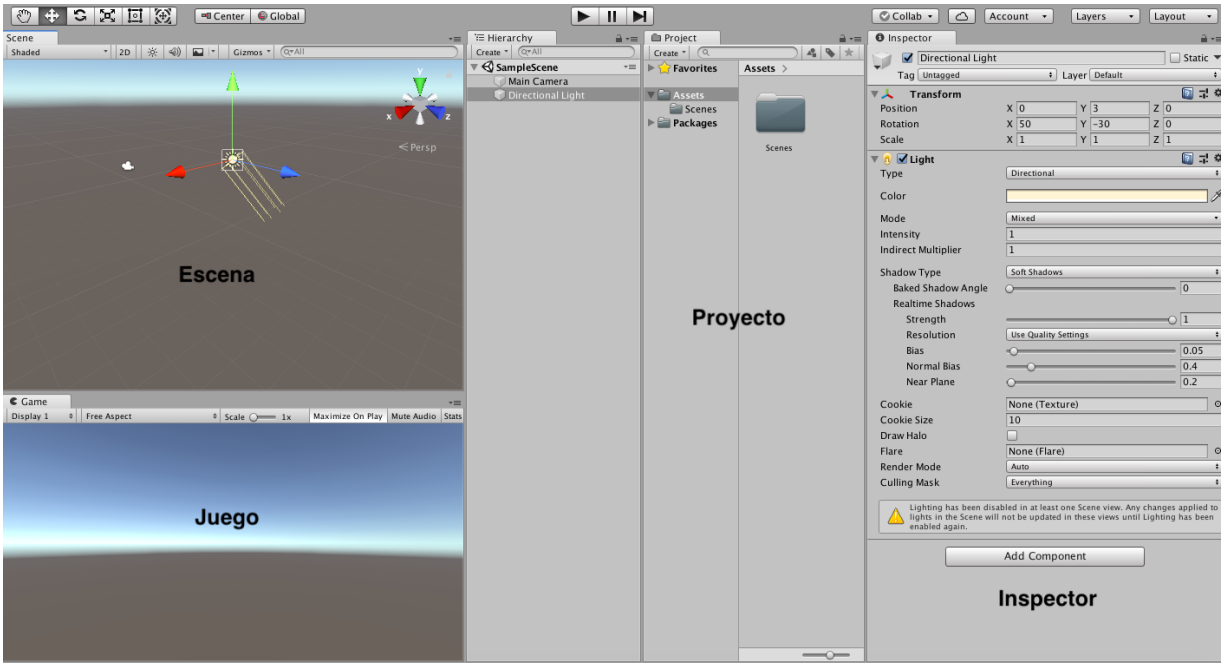


FIGURA 3.9: Interfaz del Editor de Unity3D

Juego

La vista de Juego representa lo que se observa en el juego final. La vista se obtiene a partir de la/s cámaras utilizada/s en la escena.

Script

Documento con código de programación que contienen instrucciones para ejecutar funciones y definir el comportamiento del juego.

Asset

Bloque constructivo que se encuentra en un proyecto de Unity y se usa para crear un juego, se puede guardar en formato de archivo de imagen, de sonido y modelo 3D.

Escena

Un proyecto en Unity puede poseer una o varias escenas. En la Figura 3.9 se observa una escena referenciada con ejes cartesianos, carente de objetos. Cada escena contiene todos los entornos, ambientes, decoraciones y objetos, permitiendo armar el juego de a piezas. Una escena es creada por mallas (más adelante, *Mesh*) tridimensionales o poligonales denominadas objetos. Una malla es creada a partir de puntos (vértices) y líneas dibujadas entre ellos. Estas líneas (bordes), crean polígonos, que en Unity se llaman triángulos, y componen la malla de un objeto, es decir, la superficie.

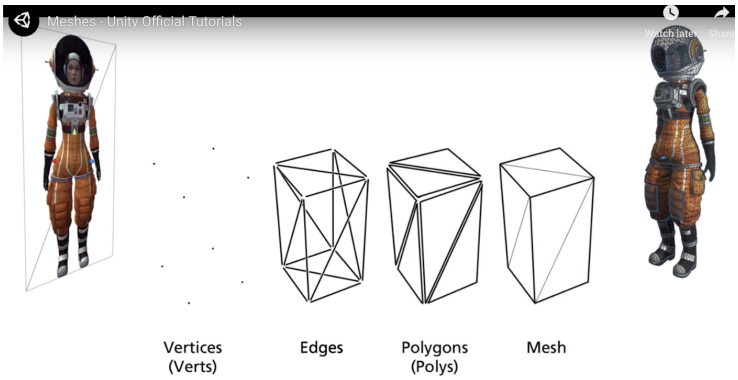


FIGURA 3.10: Malla poligonal de un objeto en Unity3D

Una escena contiene los ejes globales XYZ absolutos a todos los objetos, para describir la localización de los puntos y direcciones de desplazamientos y fuerzas. Está definido a un origen y el conjunto de tres ejes cartesianos.

Además de los ejes globales, cada objeto posee un sistema de ejes coordenados solidario a si mismo, o local. Se puede utilizar este sistema de coordenadas local si se precisa adjudicar al objeto una acción y la misma tiene relación directa con su ubicación. En consecuencia, no se necesita transformar las coordenadas globales de la escena a coordenadas locales del objeto. Por ejemplo, si se decide que un objeto arroje disparos, es más sencillo que se arrojen desde su propio origen de coordenadas con un cierto *offset*, que obtener las coordenadas globales del objeto para determinarle un origen de disparo a las municiones. La Figura 3.11 muestra la diferencia entre los ejes locales de distintos objetos y el sistema de coordenadas global único.

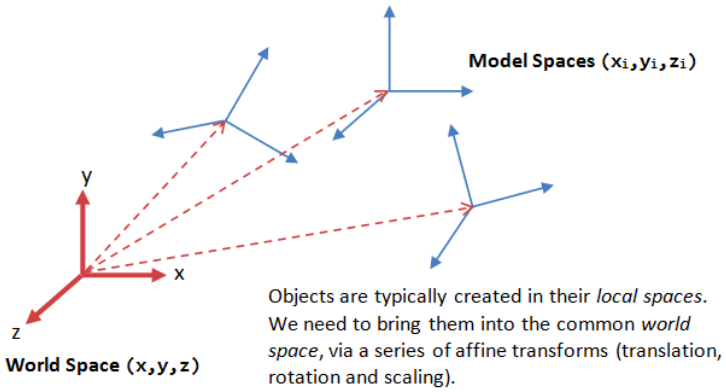


FIGURA 3.11: Sistema de coordenadas global y ejes locales de distintos objetos [21]

GameObject

Cuando un *Asset* se usa en una escena de juego, se convierte en un *GameObject*. Un *GameObject* es el concepto más importante en Unity, el cual puede representar un personaje, objeto, una fuente de luz o una cámara (Figura 3.12). Un *GameObject* por si sólo no puede realizar ninguna acción, debe contener al menos un componente; el mismo funciona como contenedor para componentes, los cuales le dan funcionalidad. Los componentes vienen en distintas formas y afectan distintas características (ver Componentes, más adelante)

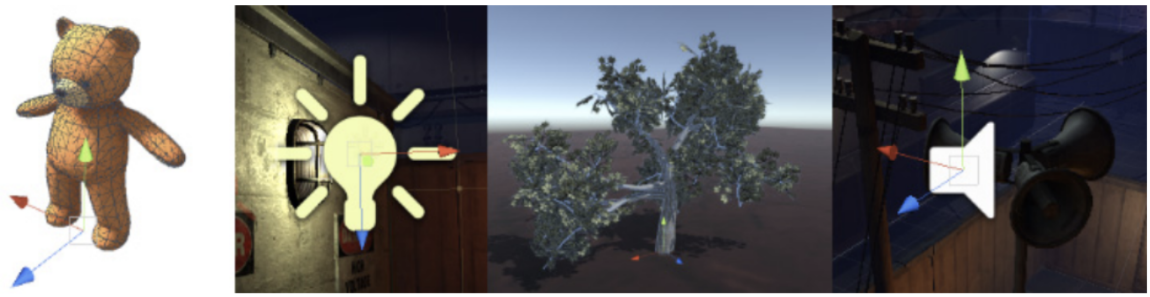


FIGURA 3.12: Cuatro tipos de GameObject en Unity3D

Inspector

El Inspector muestra información sobre el GameObject que está seleccionado en la Escena, incluyendo sus componentes y propiedades. Es posible editar estas funcionalidades desde el Inspector.

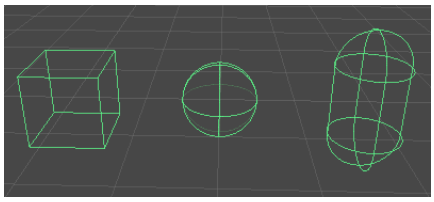
Componentes

- ***Transform*:** es utilizado para definir la posición, rotación y escala de un objeto dentro de una escena en los ejes X, Y y Z.
- ***Rigidbody*:** es el componente más importante a la hora de determinar el comportamiento físico de un objeto ya que su presencia implica que el mismo responda por física de fuerzas. Esto quiere decir que una vez adjuntado dicho componente, el objeto comienza a responder a la gravedad y, en caso de aplicarle una fuerza, moverse en consecuencia de los cálculos realizados por el motor de física.

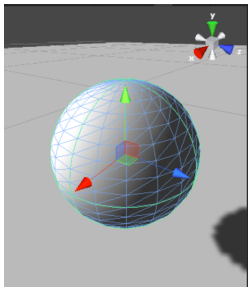
  En caso de que se desee que el objeto tenga un *Rigidbody* pero que su movimiento no sea controlado por el motor de físicas, se le debe incluir una propiedad denominada *Is Kinematic*. De esta manera el motor de física ya no tiene control sobre el movimiento del objeto sino que se mueve de forma cinemática desde un *script* (la cinemática hace referencia al movimiento de objetos sin tener en cuenta las fuerzas que lo originan). Esto es utilizado en situaciones en donde no se quiere que el objeto sea afectado por colisiones o fuerzas; el objeto va a afectar a otros objetos pero el mismo no será afectado por física. Un ejemplo puede ser controlar las características del objeto de forma cinemática desde un *script* y que pueda ser detectado por otros objetos cuando entran en contacto y se desencadenan otras acciones (ejemplo de la propiedad *Trigger* del Componente *Collider*, más adelante).

- ***Collider*:** define la forma de los objetos para poder computar colisiones. A diferencia de la malla que define la superficie visual de un objeto, el *Collider* es invisible y no necesariamente se superpone con ella. Con un *Collider*, las colisiones se computan preguntándose si determinado objeto está dentro o intersecta con la posición del *Collider* de otro objeto. Por ejemplo, si hay un cubo que está en (0,0,0) y tiene un tamaño unitario, cualquier objeto que esté entre (-0.5, -0.5, -0.5) y (0.5, 0.5, 0.5) va a colisionar con él. Por otro lado, si el volumen del *Collider* es más pequeño que el de la malla del objeto, este último puede ser atravesado visualmente hasta donde se encuentra el *Collider*, no representando de manera fidedigna a la realidad.

  Existen *Colliders* primitivos con forma de caja, esfera y cápsula (Figura 3.13a), y *Colliders* más complejos como por ejemplo *Mesh Collider*, el cual coincide perfectamente con la forma de la malla. Este último sirve para colisiones más sofisticadas en las que se busca tener un mejor cómputo de las mismas, por ejemplo para conocer qué punto de la malla o vértice colisionó. Toma todos los triángulos de la malla y computa la colisión corroborando si el objeto atraviesa algún triángulo de la malla. De esta manera se plasman deformaciones y colisiones más realistas a cambio de un mayor costo computacional.



(A) *Colliders* Primitivos



(B) *Mesh Collider*

FIGURA 3.13: Tipos de *Colliders*

  Si se configura un *Collider* con la propiedad de *Trigger*, el mismo no se comporta como un objeto sólido, es decir que otros objetos pueden atravesarlo. Esto sirve para poder llamar a otras funciones cuando un objeto entra en el espacio de otro.

Debido a que un *GameObject* con *Collider* puede o no tener *Rigidbody* y, como este último puede tener la propiedad de *Is Kinematic* activada o desactivada, la interacción del *GameObject* con su entorno varía según la combinación de dichas características. La siguiente tabla muestra las combinaciones de componentes y consecuentes interacciones:

Collider	*Rigidbody*	Interacción
Sí	No	*Static Collider*: Se utiliza en objetos que no se mueven nunca; en caso de que se mueva via *script*, el costo computacional será muy alto y puede conllevar a errores en los cálculos físicos. Otro objeto con *Rigidbody* va a colisionar con él pero no moverlo.
Sí	Sí, *Is Kinematic* desactivado	*Rigidbody Collider*: reacciona a colisiones y fuerzas aplicadas.
Sí	Sí, *Is Kinematic* activado	*Kinematic Rigidbody Collider*: se puede mover el objeto via *script* pero no responde a colisiones y fuerzas. Generalmente se utiliza para *Colliders* que a veces se quieren mover y que de manera contraria deberían ser estáticos. A pesar de que se decida que no se mueva nunca este objeto por ejemplo, sigue teniendo distinto comportamiento al *Static Collider* ya que permite ser detectado por otros objetos y utilizar la propiedad de *Trigger*.

CUADRO 3.1: Resultado de interacción de *Collider* con *Rigidbody*

3.4. Software

El *Site Map* o mapa de contenido del software, el cual muestra las pantallas y sus respectivos contenidos, se muestra en la Figura 3.14.

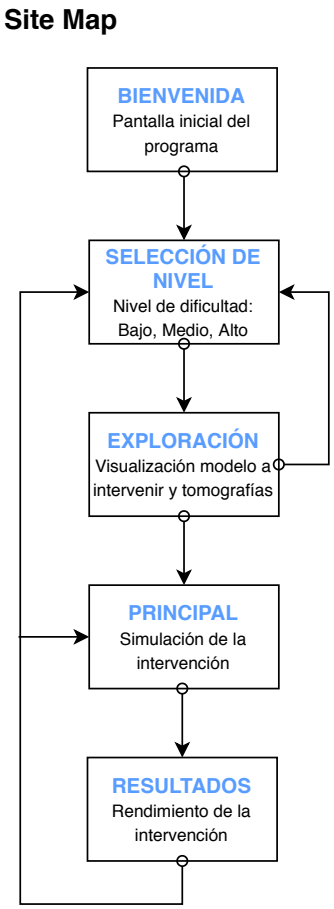


FIGURA 3.14: *Site Map* del Software

Por otro lado, el *User Flow* del software, el cual muestra las acciones del usuario y las funcionalidades del programa, se muestra en la Figura 3.15.

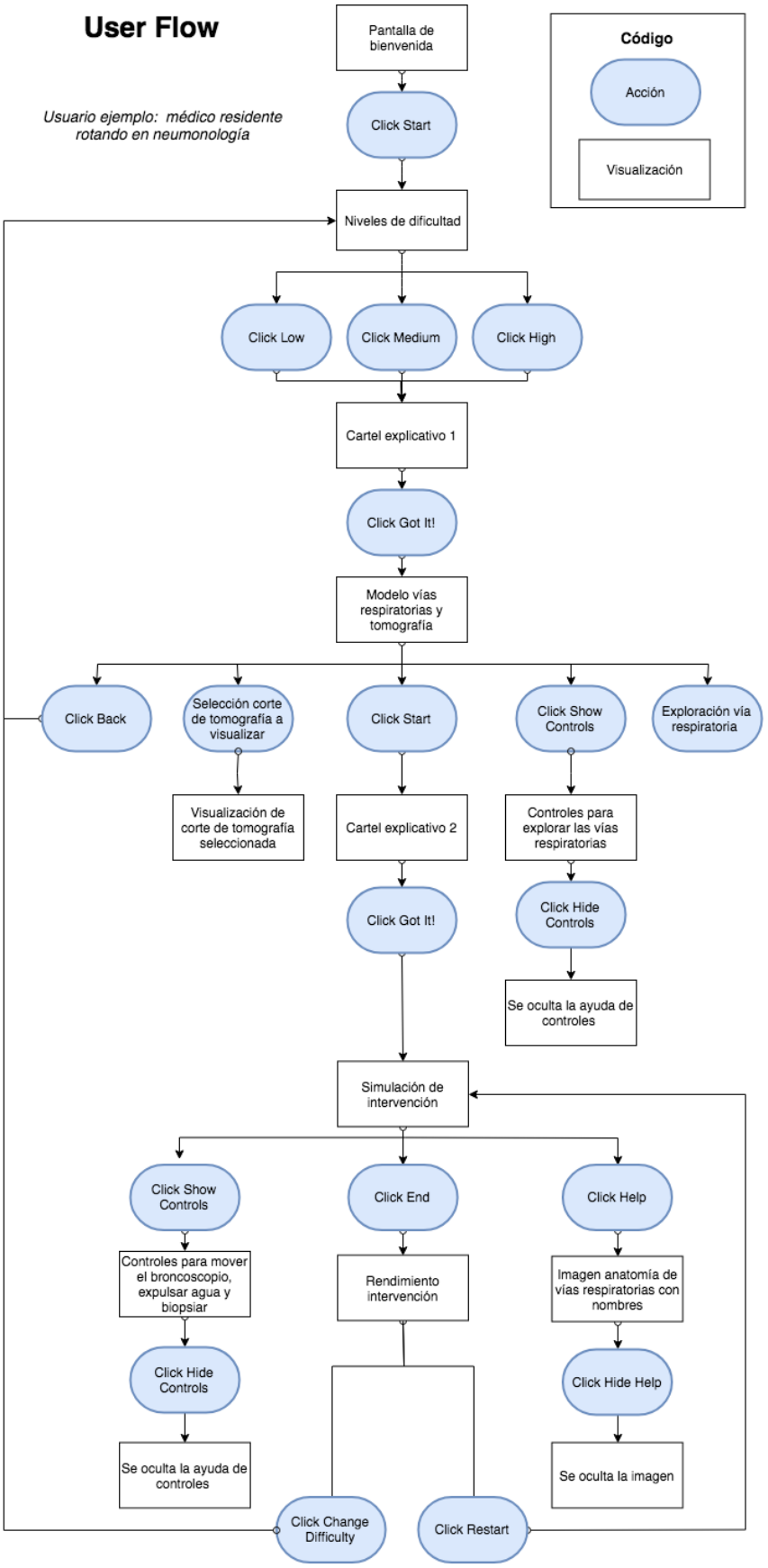


FIGURA 3.15: *User Flow* del Software

El primer cartel explicativo del *User Flow* se corresponde a la Figura 3.16 y el segundo cartel explicativo a la Figura 3.17.

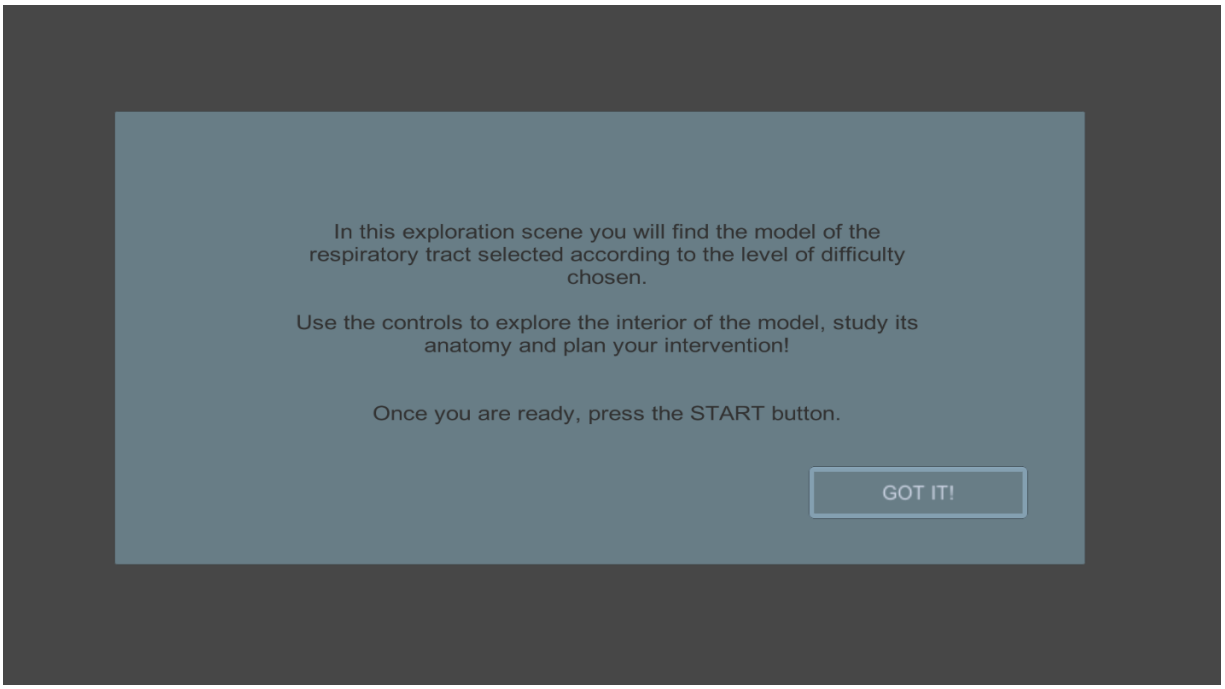


FIGURA 3.16: Primer cartel explicativo del *User Flow*

FIGURA 3.17: Segundo cartel explicativo del *User Flow*

Cabe destacar que en las interfaces se describen las teclas para acceder a los comandos de movimiento. Esto se debe a que actualmente el software existe sólo y es independiente de hardware, por lo que el *input* es el teclado de la computadora. A futuro, en caso de estar acoplado a un hardware, el *input* será el broncoscopio y los botones de ayuda de control no serán necesarios.

Como metodología de trabajo se implementa la Programación Orientada a Objetos (OOP) en la cual se delegan las responsabilidades de los objetos con distintos *scripts*. Este método es útil para crear programas escalables, simplificando la modificación a futuro de porciones de código aisladas.

El *framerate* en Unity no es constante, y va variando en dependencia del requerimiento para modificar el *frame* en pantalla. Un parámetro que se puede mantener constante, y es independiente del *framerate*, es el *FixedTimedStep*. Este dicta el intervalo entre los cálculos físicos desarrollados por el motor de físicas. Se determina de 0.05 a diferencia de 0.0167 que está por *default*. Por otro lado, el parámetro *Default Solver Iteration*, el cual define la cantidad de interacciones físicas que Unity corre en cada *frame* del *FixedTimedStep*, se coloca en 10, a diferencia del valor de 6 que está por *default*.

3.4.1. Escena de bienvenida

La escena de bienvenida es la inicial del programa y funciona como presentación, se ilustra en la Figura 3.18.

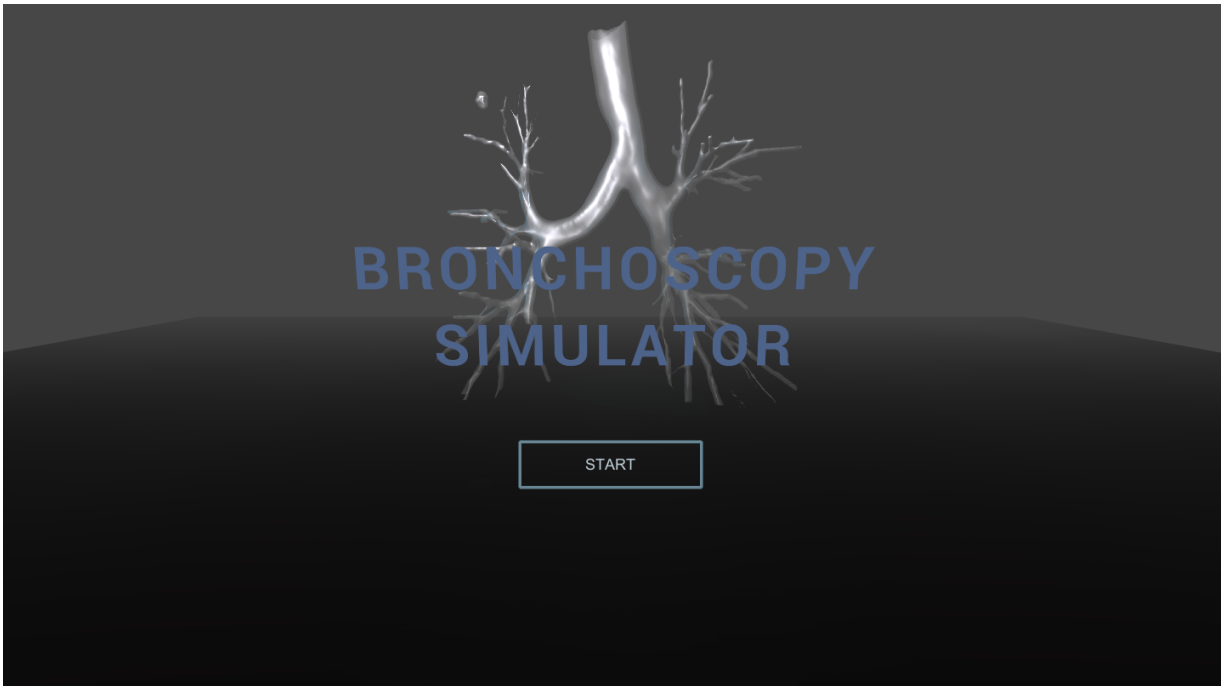


FIGURA 3.18: Escena de bienvenida

3.4.2. Escena de selección de nivel

A continuación de la escena de bienvenida se muestra la pantalla de selección de nivel (Figura 3.19), en la que el usuario debe seleccionar el nivel de dificultad que desea al utilizar el simulador.

El nivel de dificultad depende de la cantidad de lesiones endobronquiales que el modelo contenga y su respectiva profundidad, es decir cuán distales a al boca de la vía respiratoria se encuentran. Se utilizan 4 modelos base originales a partir de las máscaras definidas, y la cantidad

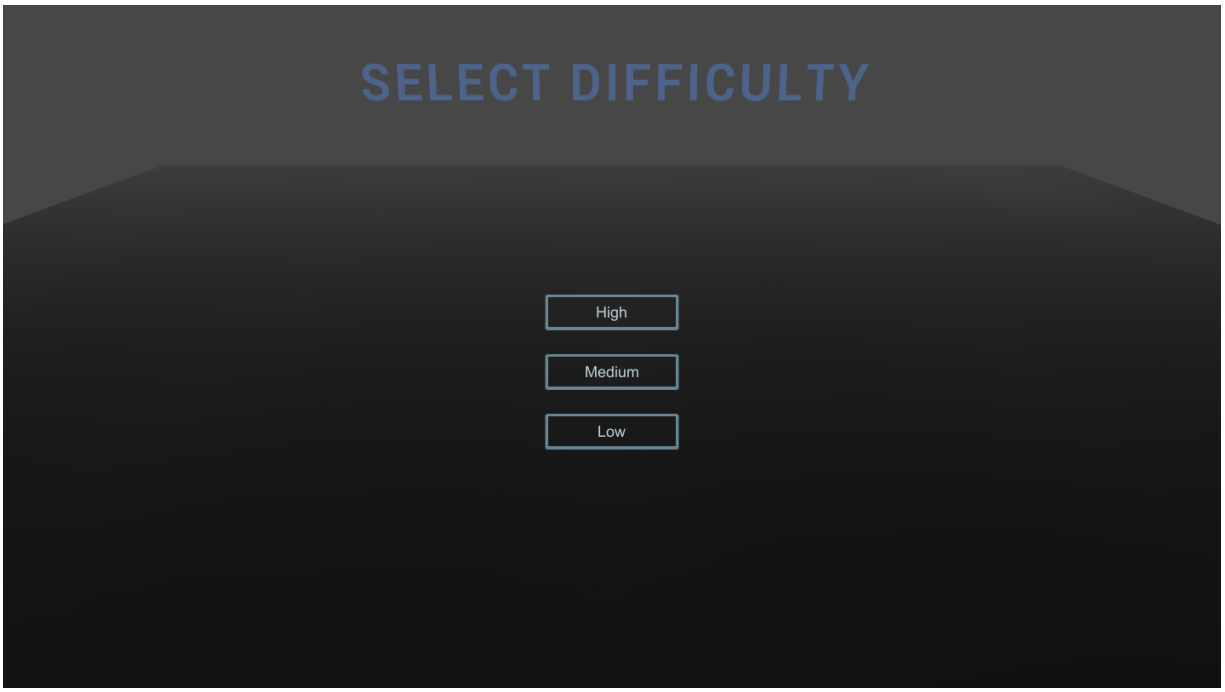


FIGURA 3.19: Escena de selección de nivel

de modelos total a consecuencia de las protuberancias agregadas se listan en la Tabla 3.2. Debido a que los modelos no poseen distinciones significativas que facilitan o dificultan el manejo del broncoscopio se utilizan indistintamente para los niveles de dificultad.

Nivel de dificultad	***Low***			***Medium***			***High***		
Cantidad de lesiones	1	1	1	2	2	2	3	2	2
Profundidad	Baja	Baja	Baja	Baja y media	Media	Media	Media	Media y alta	Alta
Máscara	01	10	60	01	10	20	10	20	60

CUADRO 3.2: Modelos utilizados y características según nivel de dificultad. Las máscaras son las proporcionadas por la cátedra Procesamiento de Imágenes Biomédicas.

3.4.3. Escena de exploración

En la escena de exploración (Figura 3.20), el usuario puede recorrer el modelo tridimensional reconstruido a partir de sus tomografías. A pesar de que previo a una intervención un médico no hace este ejercicio, se incluye esta funcionalidad en el programa con fines educativos. Sin embargo, no se especifica la cantidad de lesiones endobronquiales a biopsiar para que la experiencia del usuario no se aleje tanto de la realidad, ya que las mismas no se conocen de antemano a una intervención porque no se pueden observar en una tomografía, a diferencia de una estenosis, por ejemplo.

A su vez, se muestran las tomografía de las vías respiratorias a intervenir, con la posibilidad de poder seleccionar cualquiera de los tres cortes (sagital, frontal o transversal) como se observa en la Figura 3.21 y se puede acceder al comando de movimiento seleccionando el botón *Show Controls* de la Figura 3.22.

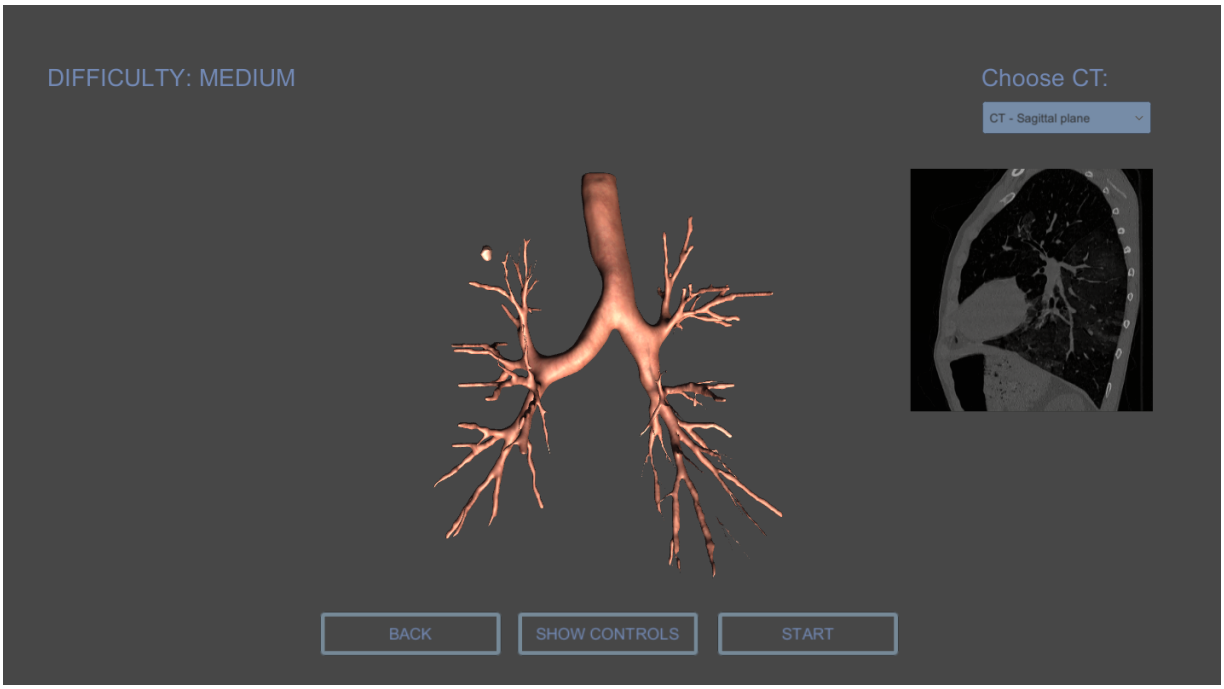
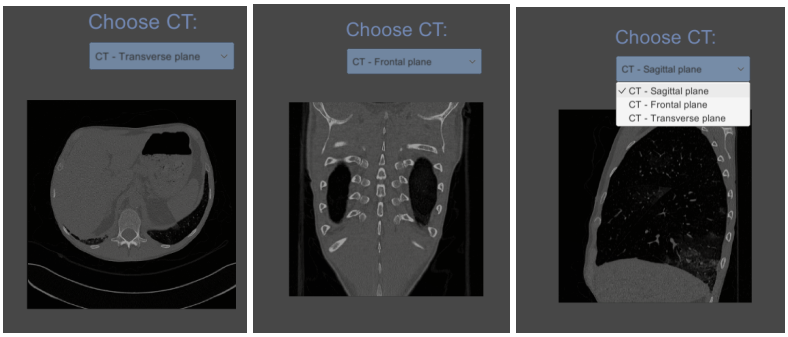


FIGURA 3.20: Planos de corte [22]



(A) Transversal (B) Frontal (C) Sagital

FIGURA 3.21: Corte de Tomografías Computadas

3.4.4. Escena principal

En esta escena se lleva a cabo la intervención, se puede observar en la Figura 3.23.

Por un lado se muestra la vía respiratoria explorada mediante la cámara asociada al broncoscopio y por otro, una vista frontal del modelo estudiado el cual es transparente para que se pueda observar el movimiento del broncoscopio en su interior en tiempo real. Las proporciones de tamaño entre el broncoscopio y el diámetro de la vía respiratoria no son reales, dicha vista es ilustrativa para mostrar la altura en la cual se encuentra el tubo de inserción con respecto a las vías respiratorias. Esta vista contiene, a su vez, un mapa de calor el cual resalta con color rojo las zonas navegadas por el broncoscopio. El mapa de calor fue proporcionado por Sensus3D y se realiza con el *script CameraPainter* el cual pinta los bronquios. En el Editor se pueden cambiar el tamaño de lo que se pinta, dependiendo de cuánto se quiere pintar durante una examinación, y la distancia de visión, es decir la distancia que se considera para pintar desde la cámara del broncoscopio a los bronquios.

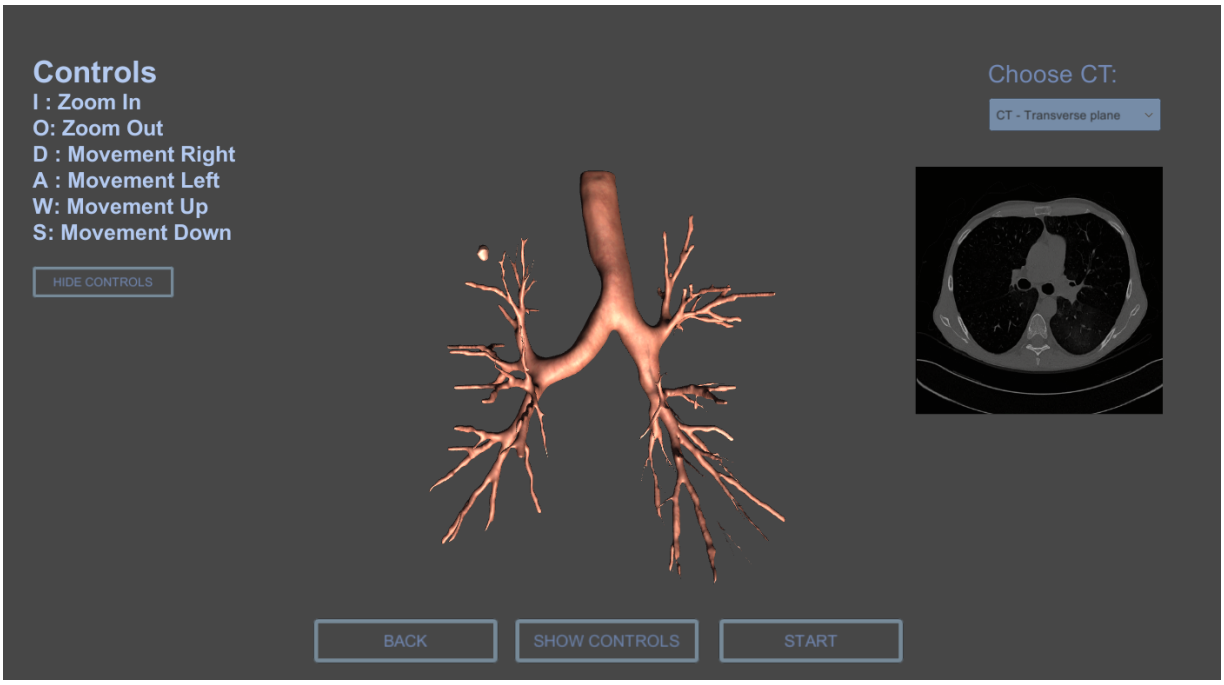


FIGURA 3.22: Controles para navegar el modelo

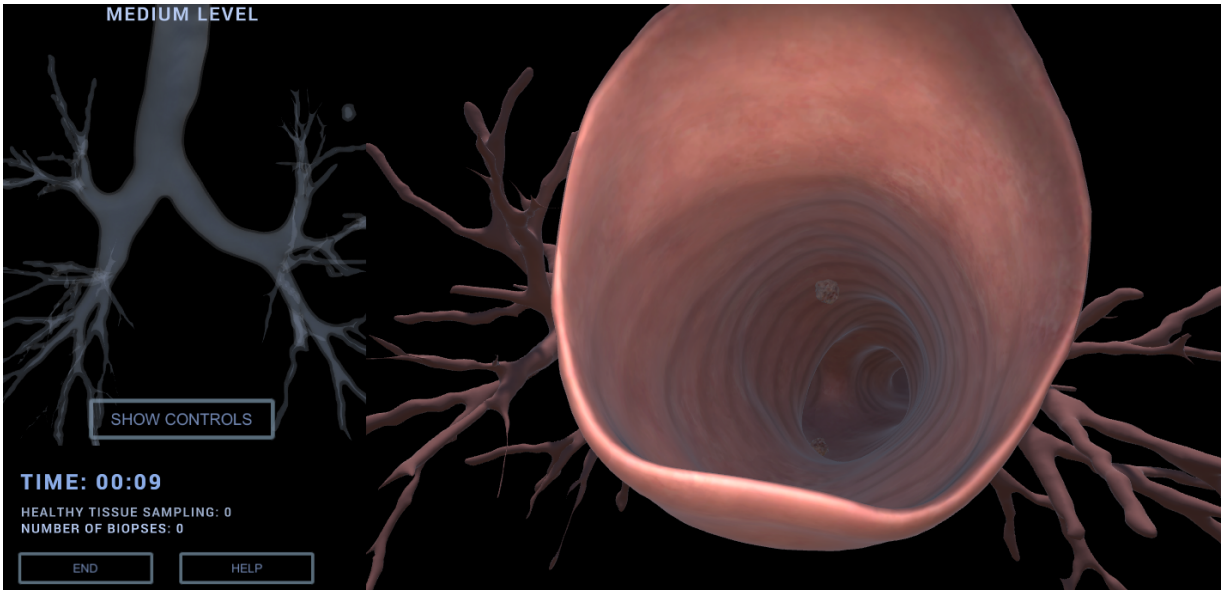


FIGURA 3.23: Escena principal

En la Figura 3.24 se puede observar por la derecha lo que muestra la cámara del broncoscopio, el cual se encuentra próximo a la carina de la vía respiratoria, y por otro lado es posible corroborar esto mismo en la vista de la izquierda.

En esta escena se muestran ciertos parámetros acerca de la intervención; a continuación se explicitan características de los mismos:

- **Tiempo de la intervención:** no se puede pausar la intervención para asemejar de manera más fidedigna la realidad en la cual no existe la posibilidad de pausar y renaudar; una intervención finaliza al seleccionar el botón *End*.

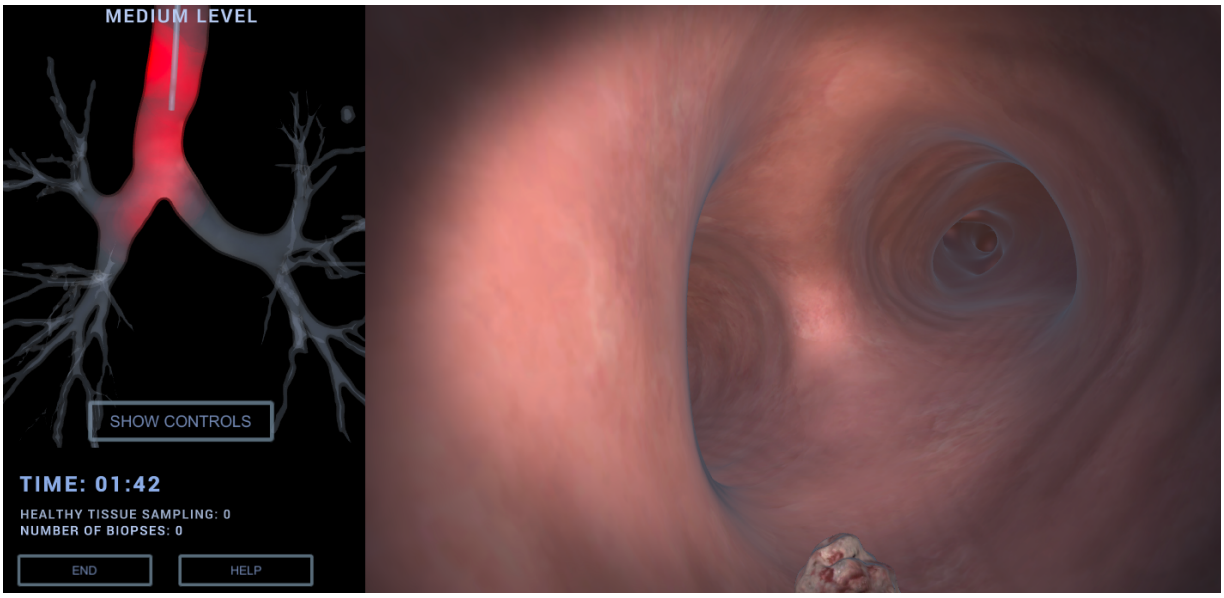


FIGURA 3.24: Mapa de calor y visualización broncoscopio de la Escena principal

- **Cantidad de biopsias realizadas:** No se distingue la cantidad de lesiones endobronquiales existentes para no facilitarle la tarea al usuario, éste debe encontrarlas sin conocer la cantidad total presente en la vía aérea.
- **Cantidad de veces que se muerde la pared sana:** Debe ser nulo ya que de manera contraria significa que se ha dañado tejido sano.

A su vez, se puede conseguir ayuda para conocer el nombre de la porción anatómica en donde se encuentra el broncoscopio seleccionando el botón *Help*, que habilita la imagen de la Figura 3.25. Para conocer los controles del movimiento del broncoscopio, expulsión de agua y pinza de biopsia, se debe seleccionar el botón *Show Controls* (Figura 3.26).

Objetos

La escena principal cuenta con dos objetos principales: un set de bronquios y el broncoscopio, el cual incluye la pinza para biopsiar.

Bronquios Los modelos de bronquios se importan en formato .obj a partir de las segmentaciones realizadas y editadas previamente. Dicha malla importada (superficie del objeto) es almacenada en un componente denominado Mesh Filter. Es el componente Mesh Renderer el que luego renderiza en la pantalla la geometría del Mesh Filter en la posición definida por el componente Transform del GameObject. Como mencionado previamente, la renderización es el proceso de generar una imagen 2D con características 3D realistas a partir de información digital.

Un modelo está compuesto por un *GameObject* para las vías y un *GameObject* para cada lesión que contenga. De esta manera se puede tratar a las lesiones y al tejido sano de forma independiente, partiendo inicialmente por distintos materiales que componen a cada tipo de tejido (Figura 3.27). A los mismos se les agrega un componente *Mesh Collider*, pero no *RigidBody*.

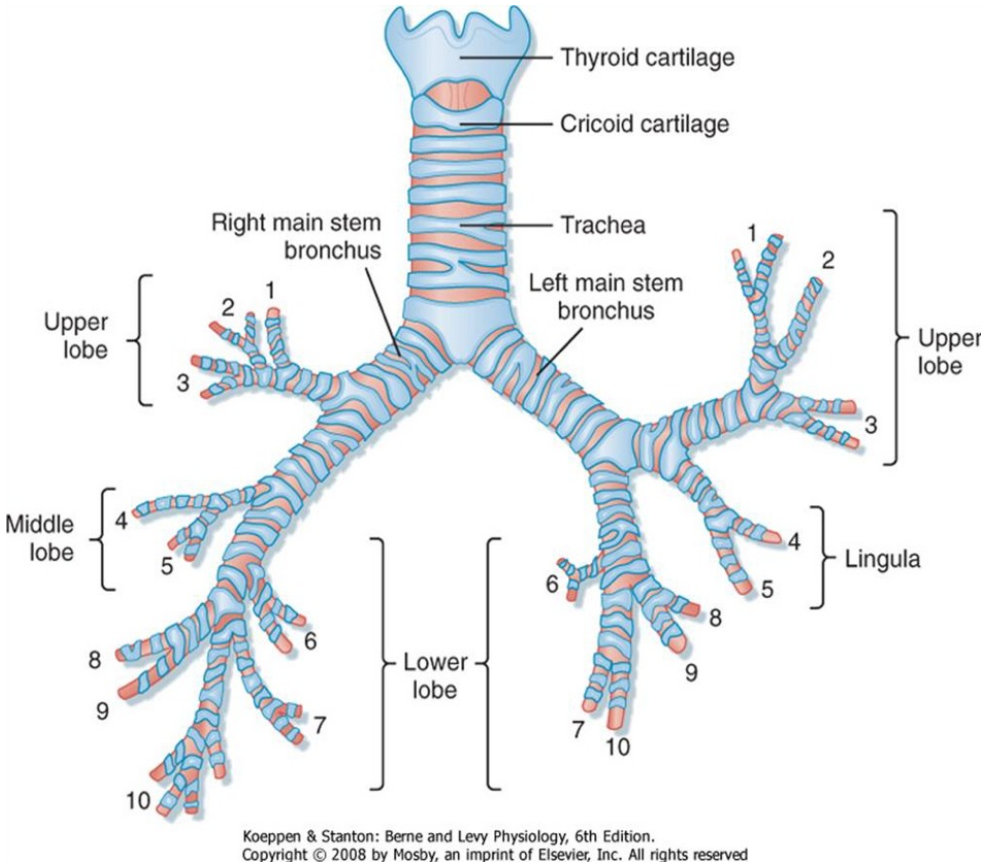


FIGURA 3.25: Anatomía de la vía respiratoria

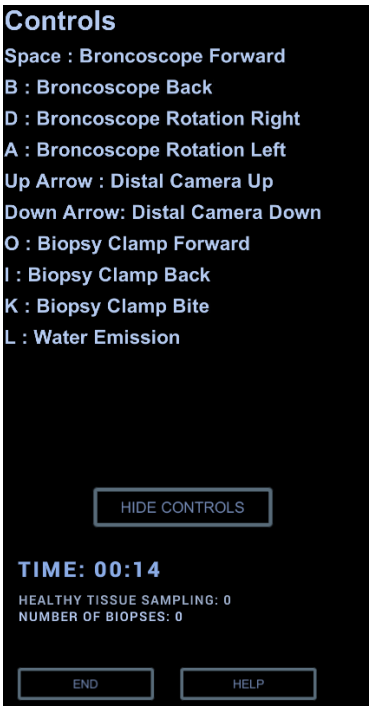


FIGURA 3.26: Comando del movimiento del broncoscopio, expulsión de agua y pinza de biopsia

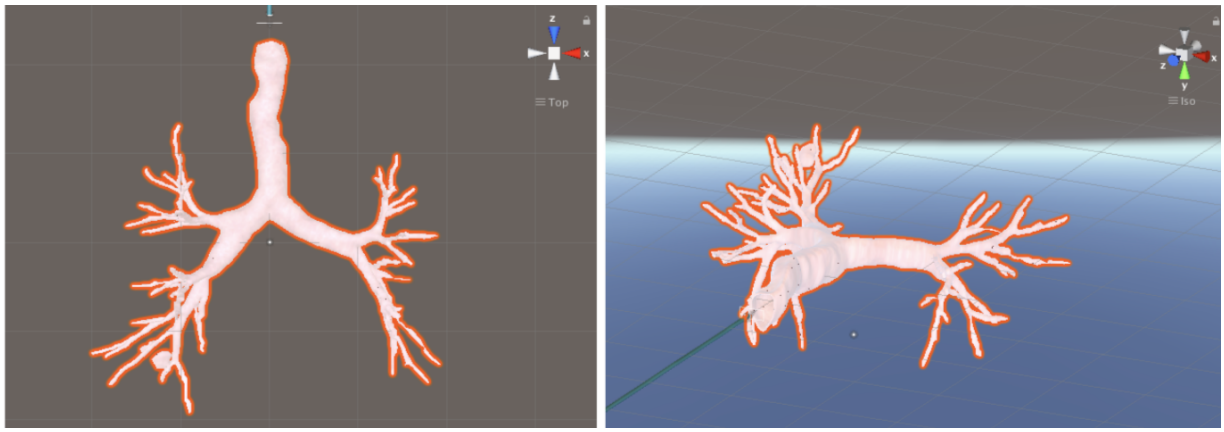


FIGURA 3.27: Modelos bronquiales segmentados en 3DSlicer importados a Unity3D como archivos .obj

Broncoscopio En cuanto al broncoscopio, se modela un broncoscopio flexible clásico con dimensiones genéricas teniendo como referencia los tres posibles movimientos mencionados en el marco teórico: el movimiento de avance, en el eje z; el movimiento de rotación, representado por el ángulo α en la Figura 2.23; y el movimiento de flexión de la porción distal cuyo ángulo es ϑ en la Figura 2.23.

Como se menciona previamente, los primeros dos movimientos, de avance y de rotación, no tienen límites, mientras que la flexión sí tiene y se aproxima en la simulación considerando el movimiento típico de un brazo robótico. El modelo se piensa como una cadena de eslabones de largo y angulación variable a la ubicación de cada eslabón en la cadena. El tubo se divide en dos porciones, una proximal y otra distal: los eslabones proximales son más largos y poseen un menor grado de angulación entre sí (imitando el comportamiento de un cable); los eslabones distales son los encargados de generar el movimiento de angulación distal, por lo que, con el fin de obtener una circunferencia de menor radio, se aumenta el número de eslabones de la cadena cinemática en la porción distal, se disminuyen sus tamaños y se aumenta la angulación relativa.

Se crea una cadena de eslabones *GameObject* con *RigidBodies*, debajo de un *GameObject* padre, siendo los eslabones los hijos, que forman un objeto cilíndrico. Cada eslabón se corresponde a un cilindro de mismo diámetro del broncoscopio. Este concepto, llamado *Parenting*, se utiliza para que todos los hijos hereden del objeto padre su movimiento y rotación. La relación del broncoscopio con los bronquios es de 1 a 5.

La única flexión del cuerpo del broncoscopio que se puede controlar mecánicamente con la palanca de control es la de la porción distal del tubo de inserción. La flexión de la porción proximal responde a las físicas del movimiento del broncoscopio y sigue la dirección de la punta flexible. En la siguiente Figura 3.28 se observa la porción distal y cómo está compuesta por bandas de metal, “vértebras”, articuladas de manera que generan una serie de vértebra-espacio-vértebra, permitiendo un movimiento de flexión hacia arriba y hacia abajo. En el caso particular del modelo de Unity las "vértebras"son reemplazadas por los eslabones de *GameObject* mencionados antes y en vez de contener espacios que permiten la flexión, se definen los ángulos de flexión. En el modelo, la sumatoria de los ángulos de flexión de los eslabones que componen la porción distal forman la angulación límite del broncoscopio de 130 grados hacia arriba y 180 grados hacia abajo.

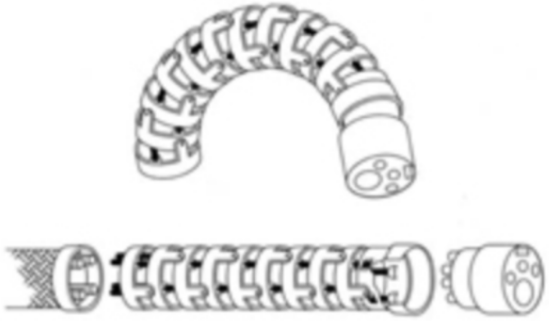


FIGURA 3.28: Sección distal del broncoscopio modelada como una cadena de eslabones con limitación angular, generando un arco de curvatura correspondiente a la angulación distal [23]

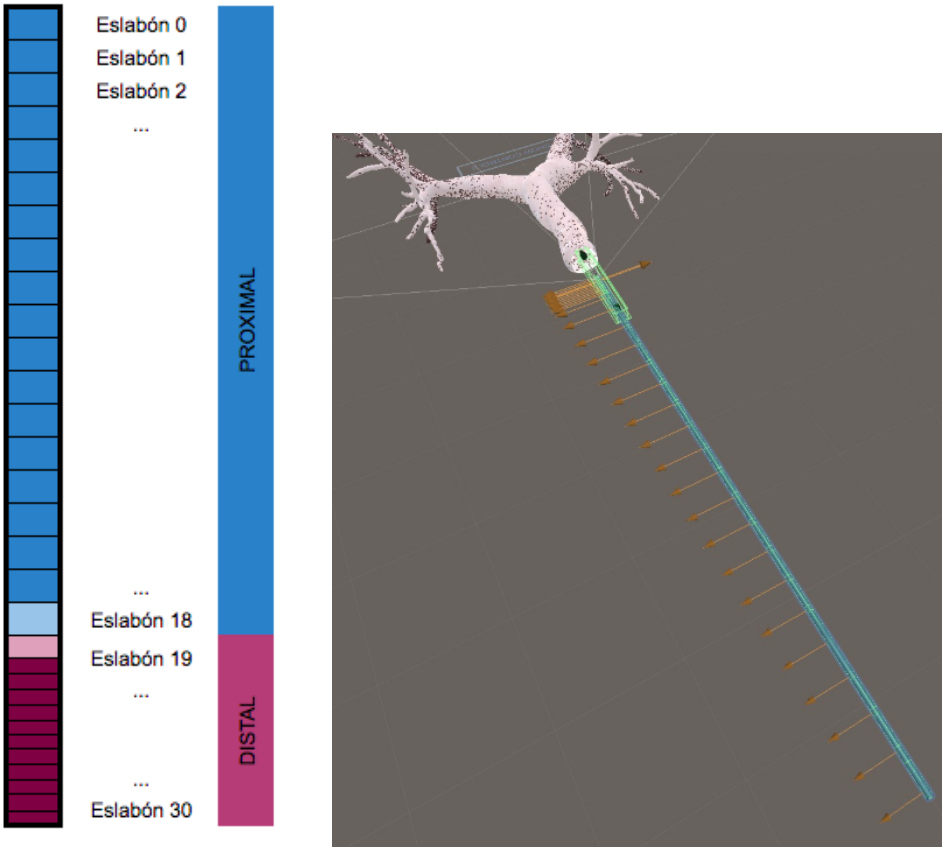
La porción distal contiene 11 eslabones de 0.3 cm de longitud, 15 grados de flexión hacia arriba y 10 grados de flexión hacia abajo. Por el otro lado, la porción proximal contiene 18 eslabones de 3.1 cm de longitud y únicamente se les permite una flexión de 2 grados, de manera de simular el movimiento natural real. A su vez, dos eslabones conectan las porciones para que la transición entre ambas no sea brusca. El eslabón en contacto directo con la porción proximal es de 2.2 cm y posee ángulos de flexión hacia arriba y hacia abajo de 2 grados. El eslabón en contacto directo con la porción distal es de 0.7 cm, con 15 grados de flexión hacia arriba y 10 grados de flexión hacia abajo (3.29).

Los eslabones contienen el Componente *HingeJoint*. Estas uniones *HingeJoint* se utilizan en Unity en objetos en donde se los quiere mover de manera análoga al comportamiento de una bisagra. En este caso, se utiliza para adjuntar los *RigidBody GameObjects* a otros pero permitiendo la rotación.

El tubo total se renderiza mediante el componente *Skinned Mesh Renderer*, el cual se utiliza en animaciones *Bone*. Un *Bone* es un objeto invisible que se une con otro formando un "esqueleto" jerárquico, y cada uno está unido a algún vértice de la malla formando articulaciones. En estos casos, la animación se define moviendo las articulaciones lo cual hace que la malla se deforme por secuencias de animación predefinidas. Cuando se mueve una articulación, los vértices se mueven con el *Bone* por lo que la "piel" acompaña el movimiento del esqueleto. Es decir que cuando los eslabones se doblan, la malla no se rompe ni muestra un movimiento de tipo bisagra si no que acompañan el movimiento y hace que el tubo total se doble.

A su vez, a la punta del broncoscopio se le coloca una cámara con un campo de visión de 120 grados, y dos luces siguiendo la ubicación que muestra en la Figura 2.21.

Muchos simuladores existentes no consideran que las vías respiratorias no tienen iluminación natural y que la misma proviene únicamente de las luces del broncoscopio. En este caso se trata de representar la realidad adjudicando una luz ambiente oscura y seleccionando la intensidad, el alcance y el ángulo de visión de las luces provenientes del bronscocopio. Esta peculiaridad es de importancia ya que el médico utiliza la luz como guía para el posicionamiento del broncoscopio dentro de la vía aérea. Si la misma se observa uniforme el broncoscopio, la fibra se encuentra en la posición correcta, como muestran las Figuras 3.30a y 3.30c; si se ilumina más en alguna dirección, implica que la fibra no se encuentra en el centro, si no próxima a alguna pared



(A) Representación en dibujo (B) Modelo en Unity3D

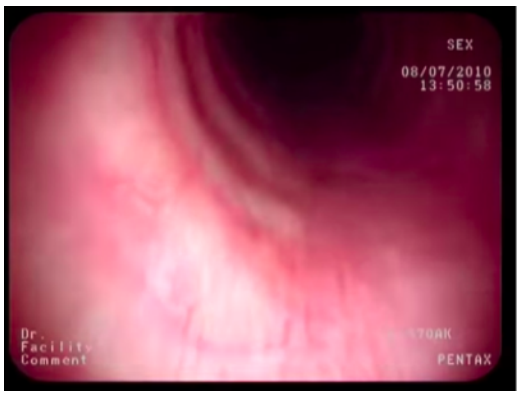
FIGURA 3.29: Eslabones

(Figuras 3.30b y 3.30d).

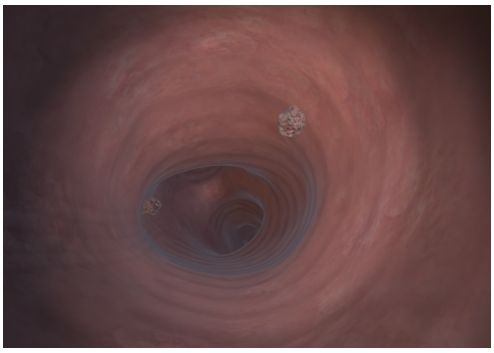
A su vez, el broncoscopio tiene la funcionalidad de poder introducir agua en la vía aérea. Para esto último se utiliza un conjunto de sombreadores que renderizan fluidos modelándolos con un sistema de partículas, denominados *Screen Space Fluids (SSF)*. Los sombreadores realizan cálculos gráficos que interactúan con la Unidad de Procesamiento Gráfico (GPU). Es posible customizar los parámetros relativos a la emisión de fluidos. Por ejemplo, se puede establecer que el sistema de partículas se emita una única vez o varias veces, cíclicamente con una determinada frecuencia. En particular, se establece que se pueda utilizar varias veces (pudiéndose detener la emisión en cualquier momento), con el fin de poder reutilizar el sistema de partículas. Asimismo, este sistema se determina con una duración en pantalla de dos segundos, y una cantidad total de 65000 partículas por emisión. Estas partículas son pequeñas esferas emitidas con una geometría seleccionada, en este caso cónica (con un ángulo de 32 grados, un arco de 360 grados, y un radio inicial de 28 % del cono). El sistema de partículas se determina hijo del eslabón más distal del broncoscopio, heredando las características de este último, con el fin de simular que proviene desde allí.

Pinza de biopsia Por último, la pinza de biopsia está compuesta por un cilindro de dimensión casi nula, el cual se ubica en la punta del broncoscopio. El mismo se escala al momento de querer utilizarlo, junto con sus dos puntas los cuales son las abrazaderas, como se observa

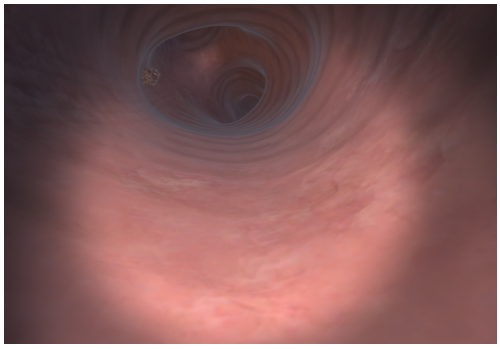
(A) Broncoscopio centrado en intervención real [72]



(B) Broncoscopio no centrado en intervención real [72]

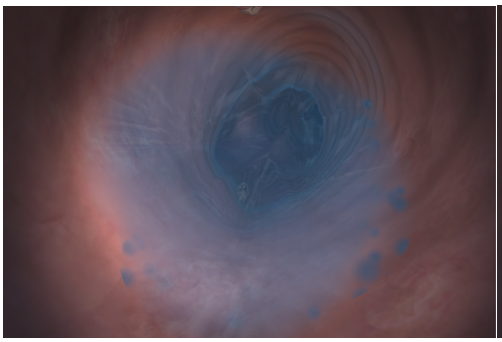


(C) Broncoscopio centrado en Unity

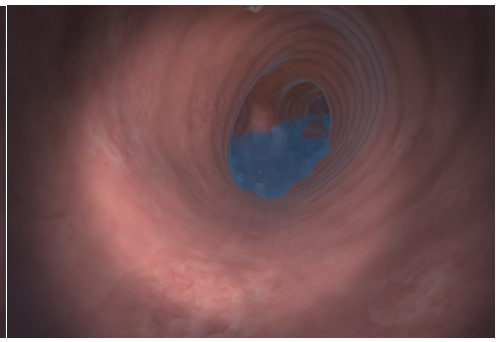


(D) Broncoscopio no centrado en Unity

FIGURA 3.30: Ubicación broncoscopio y efecto de luz



(A) Al presionar la tecla de expulsión



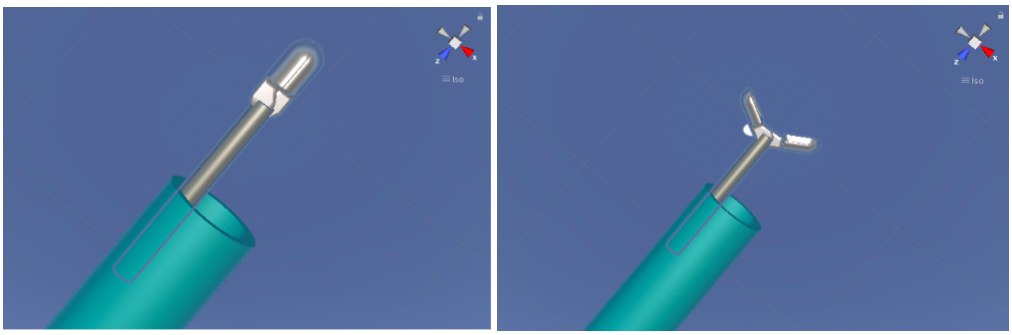
(B) Al soltar la tecla de expulsión

FIGURA 3.31: Expulsión de agua por parte del broncoscopio

en la siguiente Figura 3.32. Los tres objetos poseen *RigidBody*, sin gravedad ni cinemática, y contienen el componente *ConfigurableJoint* para articular la unión entre ellos. A diferencia del *HingeJoint* que conecta los eslabones que componen el tubo del broncoscopio, el *ConfigurableJoint* se diferencia por su mayor nivel de customización, ya que se pueden configurar más parámetros de las uniones.

Caracterización de los movimientos del cuerpo del broncoscopio

El movimiento del broncoscopio se controla mediante un *script* y el *input* del usuario equivale a presionar una tecla en la computadora. Para ello, se crean tres scripts: *MovementForwardTop*,



(A) Pinza cerrada (B) Pinza abierta

FIGURA 3.32: Modelo de Pinza

MovementDistal y *Rotate*, que controlan los tres tipos de movimiento que el broncoscopio puede realizar.

MovementForwardTop genera que el broncoscopio se mueva hacia adelante y hacia atrás dependiendo de qué tecla se presiona, y simula el movimiento del usuario al querer introducir y retirar el broncoscopio de la vía respiratoria. Todas los eslabones que forman el tubo del bronscoscopio lo contienen, por lo que todos son afectados al mismo tiempo. En este caso, el movimiento se produce definiendo una velocidad, en una dirección global, con un sentido definido, y aplicándole fuerza.

$$Movimiento\ del\ eslabón\ =\ (sentido\ del\ movimiento)\ x\ (fuerza) \tag{3.1}$$

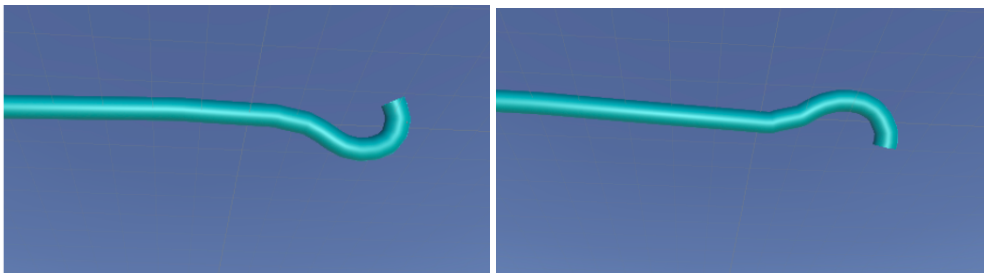
La dirección se determina por el eje longitudinal de la vía aérea, y el sentido del movimiento queda determinado por la tecla presionada. La fuerza es un parámetro fijo definido empíricamente.

MovementDistal es utilizado para regular el movimiento de la punta del broncoscopio hacia arriba presionando una tecla y hacia abajo al presionar otra tecla, simula el movimiento del broncoscopio a consecuencia de que el usuario mueva la palanca de control. Dicho *script* se le agrega únicamente al último eslabón de la cadena que forma el tubo del broncoscopio, ya que las restantes acompañan el movimiento al estar conectados por *HingeJoints* y que sus respectivos ángulos se encuentren limitados. El movimiento se produce definiendo una velocidad constante en una dirección local, en sentido contrario al movimiento hacia adelante:

$$Movimiento\ del\ eslabón\ =\ (dirección\ del\ movimiento)\ x\ (velocidad) \tag{3.2}$$

Donde la dirección de movimiento queda determinada por la tecla presionada, el sentido es siempre contrario al movimiento hacia adelante, y la velocidad es un parámetro fijo definido empíricamente.

Este movimiento del extremo distal se diferencia del longitudinal en el hecho de que está definido localmente. Esto implica que cada eslabón se mueve relativo a su sistema de coordenadas propio. Puesto que la fibra del broncoscopio toma distintas angulaciones por la desviación inherente de la vía aérea, el eje global no siempre es paralelo a la dirección del movimiento distal, pero sí lo es a su eje local, como lo ilustra la Figura 3.34.



(A) Movimiento distal hacia arriba (B) Movimiento distal hacia abajo

FIGURA 3.33: Movimientos distales del broncoscopio

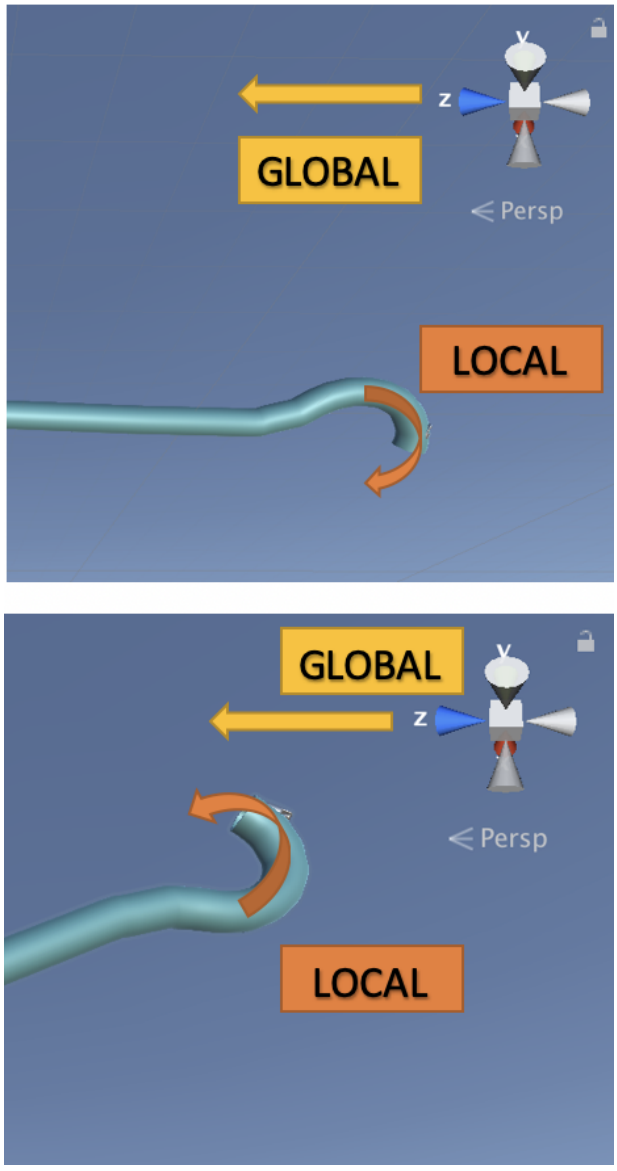


FIGURA 3.34: Diferencia entre el sistema de coordenadas global, y el local al broncoscopio.

Por último, *Rotate* controla la rotación del broncoscopio y simula el movimiento del usuario al querer rotar el broncoscopio una vez dentro de la vía respiratoria. El *script* se le aplica a todos los eslabones y en este caso se altera la rotación del componente *Transform* de cada eslabón de

forma local.

La Tabla 3.3 resume la función de cada *script*, qué eje controla y a qué eslabón se le agrega. La última fila, la variable de entrada, hace referencia al nombre del *input* necesario para realizar cada movimiento.

Script	*MovementDistal*	*MovementForwardTop*	*Rotate*
Movimiento	Punta hacia arriba y hacia abajo	Traslación, hacia adelante y hacia atrás	Rotación sobre eje longitudinal
Eslabón	El más distal	Todos	Todos
Variable de entrada	*moveUp*; *moveDown*	*forward*; *backwards*	*rotateLeft*; *rotateRight*

CUADRO 3.3: Resumen de los *scripts* que controlan los movimientos del broncoscopio, a qué eslabón se asocian y en qué eje actúan.

Cabe destacar que a los eslabones, una vez dentro de la vía aérea, se les activa el efecto de la gravedad en ellos. De esta manera se arrastran por la pared inferior de la vía aérea, representando en mejor medida una intervención a causa de la posición supina del paciente. Esto se logra ubicando un plano circular transparente en la entrada de la vía aérea. Este umbral, tiene asociado un código que genera que a todo *GameObject* eslabón que pase a traves de él, se le active la gravedad (Figura 3.35).

Por último, se crea un *script* denominado *KinematicManager* el cual contiene dos listas, *distalList* y *proximalList*, en donde se guardan los *RigidBodies* de los eslabones correspondientes a cada porción. Debido a que se pueden realizar distintas combinaciones de movimientos, las características de los *GameObjects* deben variar para así poder simular en mejor medida el movimiento real. Es por eso que desde este *KinematicManager* se controla la cinemática de los eslabones, el ángulo de cada eslabón y la activación/desactivación de la rotación o algún otro *script* correspondiente a un movimiento.

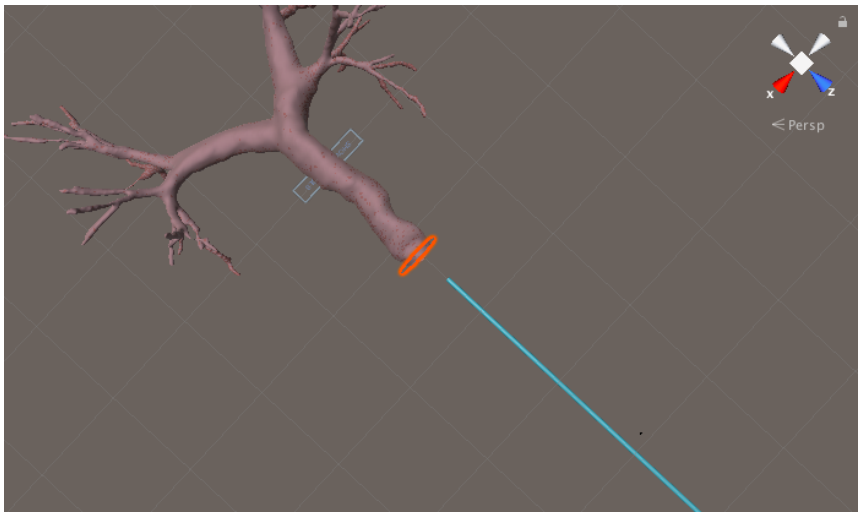
La siguiente tabla muestra las características que varían dependiendo del movimiento deseado, de forma individual.

Característica	**Movimiento**			
	Avanzar/Retroceder	Punta distal arriba/abajo	Rotación	Sin movimiento
Rotación congelada	No	No	No	Sí
Ángulos de los eslabones proximales en 0 grados	Sí	No	Sí	Sí
Componente *Is Kinematic*	Desactivado	Desactivado	Desactivado	Activado, excepto último eslabón

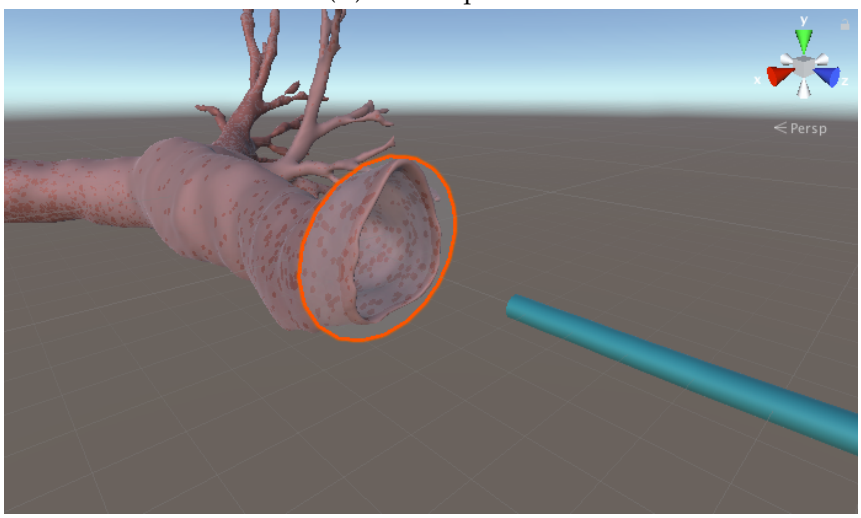
CUADRO 3.4: Caracterización de los movimientos del broncoscopio en *KinematicManager*

Analizando dichos movimientos individuales se consigue que al no presionar ninguna tecla correspondiente a alguno de los movimientos individuales o combinados mencionados anteriormente, se fuerza a que los eslabones tengan la propiedad de *Is Kinematic* activada para que el broncoscopio no continúe moviéndose solo por inercia, y, en consecuencia se congela la rotación.

A su vez, cualquier sea la combinación de los movimientos se desactiva la cinemática de los eslabones, se descongela la rotación, se activa nuevamente el *script MovementForwardTop* y se establecen las angulaciones de los eslabones correspondientes. En el caso particular de que se



(A) Vista superior



(B) Vista acercada

FIGURA 3.35: Cilindro transparente con borde anaranjado en la entrada de la vía aérea

quiera ir únicamente hacia adelante o hacia atrás, los ángulos de los eslabones proximales se establecen en 0 grados para que el tubo obtenga cierta rigidez y su forma no se vea afectada por las colisiones contra las vías aéreas.

Además, dicho *script* desactiva el movimiento del broncoscopio hacia adelante y hacia atrás si la pinza está colisionando para que el mismo no patine contra la pared de la vía aérea, y lo activa nuevamente una vez que la misma deja de colisionar.

Es importante resaltar que el origen del movimiento de la punta del broncoscopio en la simulación es distinto a la realidad por lo que los resultados obtenidos son distintos. En la práctica, el movimiento de mando del broncoscopio para el movimiento de la punta distal se genera a partir de una posición inicial de la palanca de control que se mueve hacia abajo y hacia arriba. Dependiendo de la distancia que recorre la palanca con respecto a su posición inicial, el grado de angulación de la punta. Se mantiene fija dicha angulación siempre y cuando la palanca permanezca en la misma posición; dependiendo de cuanto se desvíe la palanca de la posición

original, la angulación. Una vez que se suelta la palanca y vuelve a su posición original, la punta del broncoscopio también vuelve a su posición original.

En la simulación, debido a que para generar el movimiento se presiona una tecla de la computadora, el grado de angulación se obtiene a través del tiempo que se mantiene presionada la tecla y se traduce la distancia que recorre la palanca de la posición original en el grado de angulación obtenido. Para obtener una angulación determinada simplemente se suelta la tecla y cualquier movimiento consecuente parte de esta última angulación. De esta forma no se puede lograr el regreso de la punta del broncoscopio a su posición original al soltar la tecla, como ocurre en la realidad al soltar la palanca, ya que al soltar la tecla se comanda que la angulación se quede en la última posición. De manera contraria no se podrían conseguir angulaciones intermedias a la máxima, hecho que limitaría la obtención de biopsias, ya que su ubicación es precisa y requiere de una angulación que permita alcanzar las lesiones endobronquiales.

Caracterización de los movimientos de la pinza

Como se ha mencionado anteriormente, la pinza está compuesta por un cilindro de ancho cercano a cero y dos puntas. El cilindro contiene el *script nipperCombined* el cual regula el movimiento de la pinza hacia afuera y adentro del canal de trabajo del broncoscopio mediante el escalado del cilindro. También controla la apertura de las puntas para poder biopsiar.

Para que la pinza pueda avanzar mediante el escalado del cilindro se deben cumplir las siguientes condiciones:

- Que se esté presionando la tecla para permitir que la pinza salga del canal de trabajo
- Que no posea ningún vértice de tejido colisionando con las puntas

El cilindro se escala con una proporción determinada y local a su eje longitudinal. Su posición local se va actualizando de manera que se mueva hacia delante lo que el cilindro se escala, es decir que se mueve longitudinalmente la proporción previamente mencionada multiplicada por la escala del objeto.

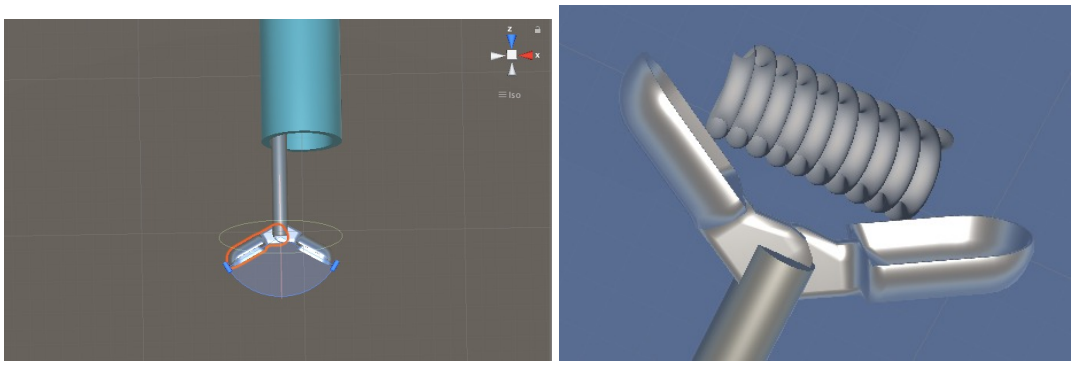
Por el contrario, para que la pinza vuelva a entrar dentro del canal de trabajo del broncoscopio, el cilindro se escala con un factor negativo. Para este caso, las condiciones a cumplir son las siguientes:

- Que se esté presionando la tecla para ingresar al canal
- Que la escala del cilindro sea positiva

A su vez, la apertura y cierre de las puntas se regula estableciendo el límite angular en la junta de las punta de la pinza en 60 grados y fijando un resorte con una constante de rigidez elástica muy grande para que se abran, cuando se presiona la tecla para abrir. Cuando se desea cerrar, la constante de rigidez elástica entre ambas puntas se vuelve pequeña (para que se puedan juntar) y el ángulo entre ellas se establece en 0 grados (Figura 3.36).

Colisiones

Con el fin de simular las colisiones entre los bronquios, la pinza y el broncoscopio, cada eslabón del broncoscopio posee un *CapsuleCollider* (Figura 3.37), y el extremo central de la pinza también (Figura 3.38), mientras que los bronquios poseen un *MeshCollider* (Figura 3.39).



(A) Límite angular de apertura (B) Resorte ilustrativo del efecto

FIGURA 3.36: Apertura de las puntas de la pinza

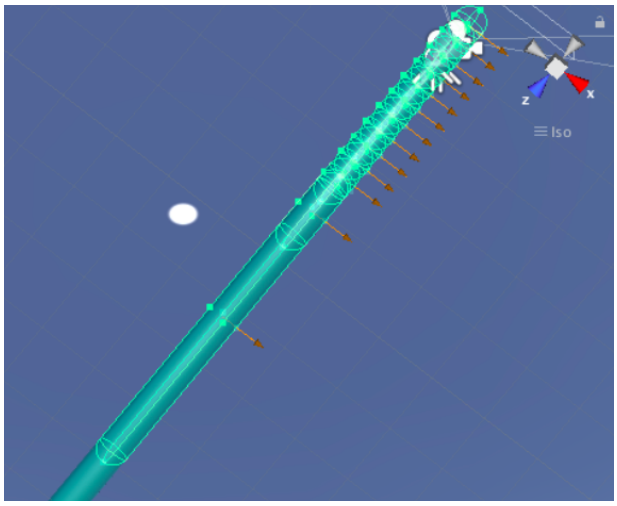
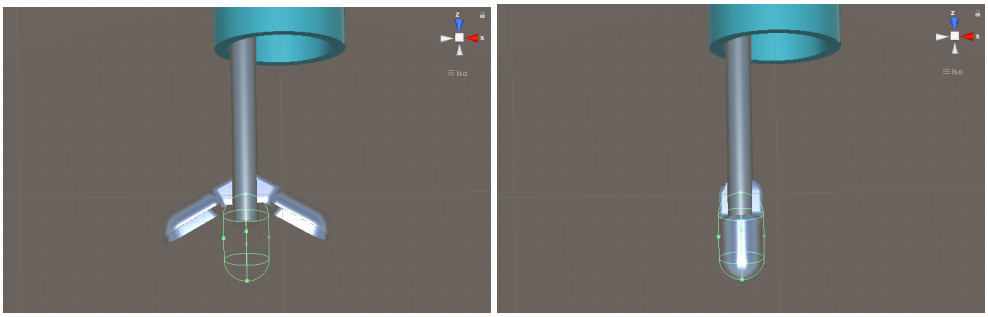


FIGURA 3.37: *CapsuleCollider* de los eslabones que forman broncoscopio

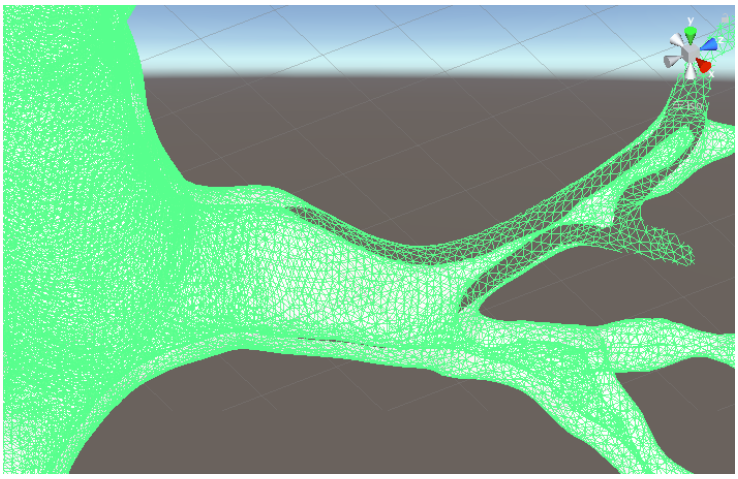


(A) Pinza abierta (B) Pinza cerrada

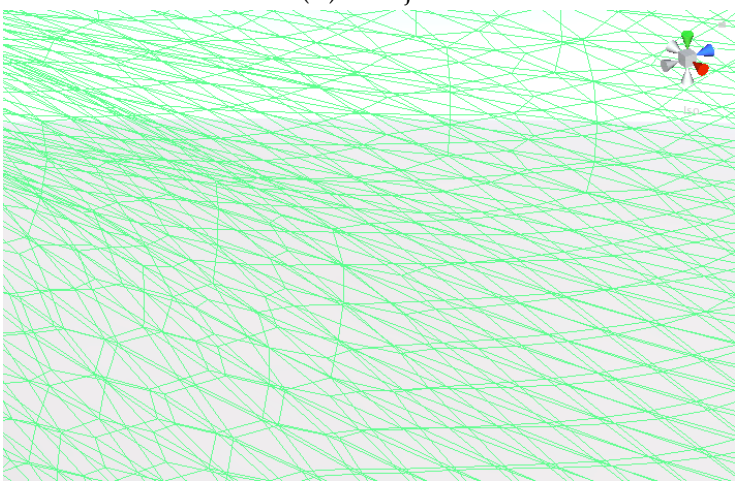
FIGURA 3.38: *CapsuleCollider* central de la pinza

Biopsia

Todo *GameObject* cuya malla pueda ser mordible (granuloma o pared de la vía aérea) contiene un *script* compartido llamado *BiteableMesh*, el cual define los parámetros que representan al área de mordida y su profundidad con un divisor; mientras más grande el divisor, menos se hunde con la mordida. Una sección mordida se hunde de forma lineal desde los bordes al centro.



(A) De lejos



(B) *Zoom*

FIGURA 3.39: *MeshCollider* de los bronquios

El *script collisionTip* controla la mordida y se le adjunta al *GameObject Collider* transparente mencionado anteriormente en forma de cápsula (Figura 3.38) que se coloca en la unión entre el tubo de la pinza y cada punta. Una mordida se compone por la apertura y cierre de las puntas de la pinza; para abrir las puntas de la pinza se debe presionar una tecla y una vez que se suelta, la pinza se vuelve a cerrar.

Dicho *script* accede a los triángulos de la malla mordida y traslada sus vértices solidariamente a la pinza, con el fin de simular la extensión del tejido debido a una mordida. Una vez que se estiran y superan cierta distancia, estos vértices se sueltan de la punta de la pinza y vuelven a una posición más hundida que la original, simulando una mordida. Inicialmente se inicializa el radio del tejido que se biopsiará y la extensión máxima que el mismo se puede estirar. La Figura 3.40 muestra el flujo de procesos que representa el *script*, que se describen a continuación. La Figura 3.41 muestra los parámetros utilizados en el *script*.

Cuando se detecta la colisión de la cápsula con algún objeto, si la misma todavía no tiene algún tejido conectado (1) y el objeto con el que chocó posee el componente *MeshCollider* (2), se guardan todos los vectores de los vértices del *MeshFilter* del tejido en dos listas denominadas *bronchiVertices* y *bronchiVerticesOriginal*.

Mientras que se continúe colisionando y se mantengan las condiciones (1) y (2), se guardan los

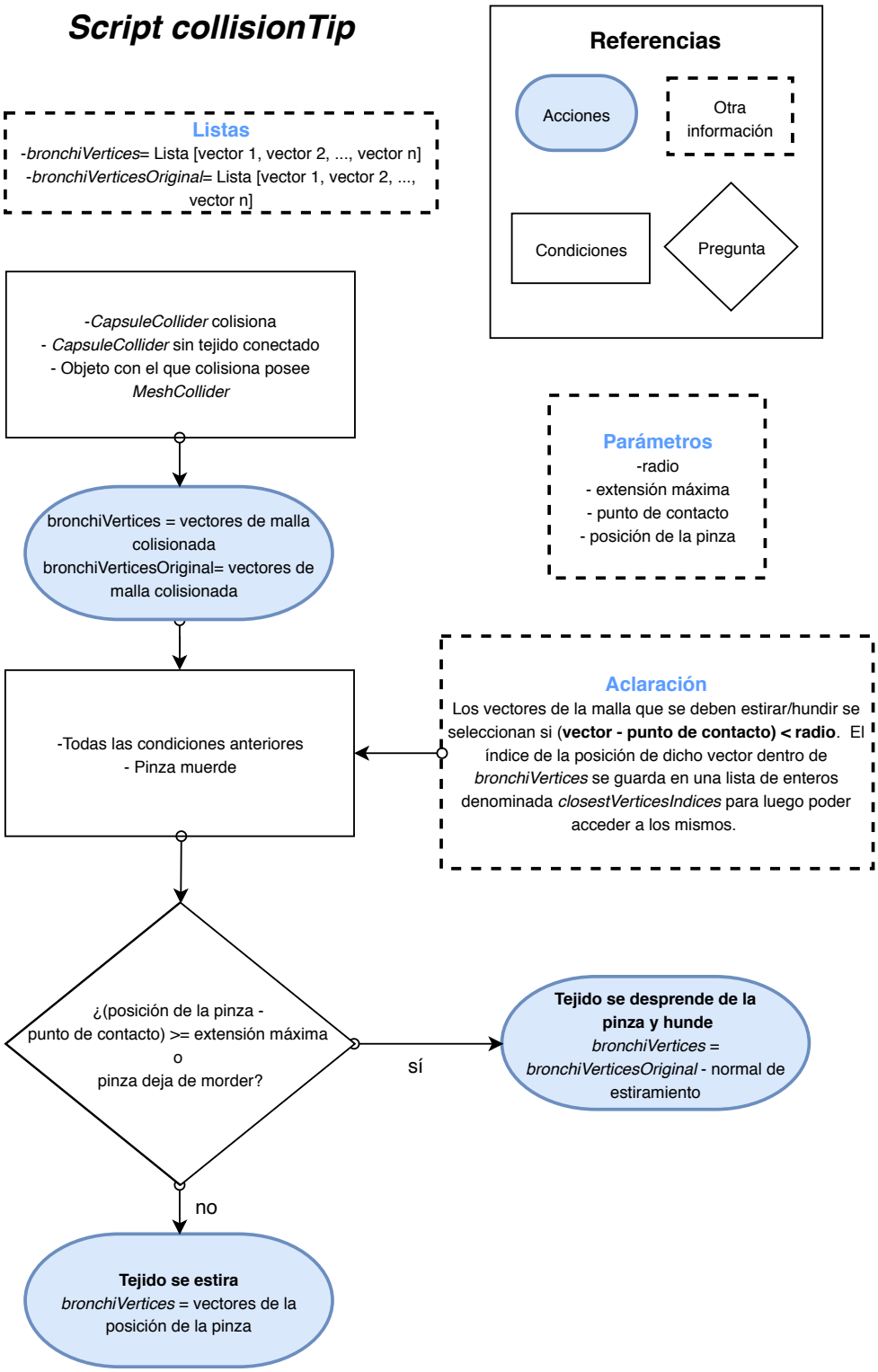
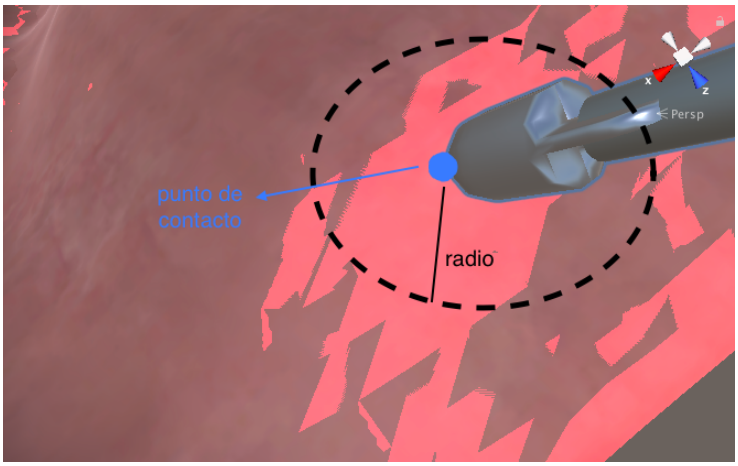


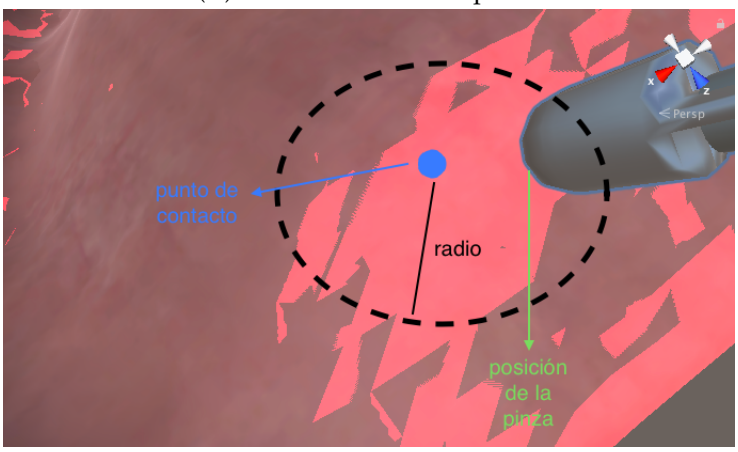
FIGURA 3.40: Diagrama explicativo del *script collisionTip*

vectores del punto de contacto y la normal al mismo en coordenadas locales. Se decide usar una distancia local ya que es más estable porque no depende de la escala del mundo.

Si se está colisionando, se muerde, y no hay tejido ya conectado a la misma, se seleccionan los vértices más cercanos al punto de contacto. Esto se logra buscando la distancia entre cada vector dentro de la lista *bronchiVertices* al punto de contacto. Si la misma es menor al radio configurado



(A) Al colisionar con la pared



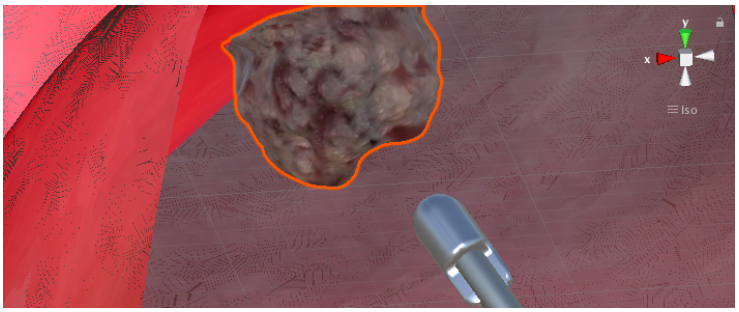
(B) Al alejarse de la pared

FIGURA 3.41: Colisión de la pinza

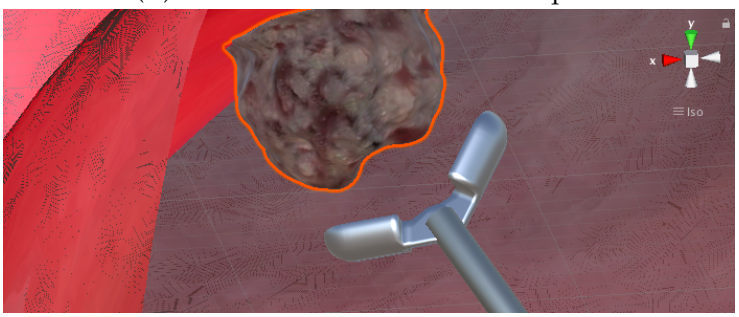
inicialmente, se agrega el índice de la posición de dicho vector de la lista *bronchiVertices* a una nueva lista denominada *closestVerticesIndices*. Estos serán los vértices que se estirarán y seguirán la posición de la punta de la pinza.

Mientras siga colisionando, para que dichos vectores se estiren, se reemplazan los vectores de *bronchiVertices* identificados por los índices de la lista *closestVerticesIndices* por la posición de las puntas de la pinza. Mientras la distancia entre el punto de contacto inicial y la posición de la pinza sea menor a la extensión máxima del tejido establecida inicialmente, los vectores de la lista *bronchiVertices* mantienen su nueva posición la cual es igual a la posición de las puntas de la pinza, con su *offset* correspondiente para que se genere el efecto de tejido elástico siendo mordido. De manera contraria, cuando la distancia pasa a ser igual o mayor a la extensión máxima, o la pinza deja de tomar una muestra, se les asigna una posición con una profundidad mayor que la original para representar la carencia del tejido biopsiado que se retira. Dicha hendidura se obtiene restándole la normal de contacto a los vectores de las posiciones originales (guardados inicialmente en *bronchiVerticesOriginal*) y reemplazando los vectores correspondientes de *bronchiVertices* con dichos nuevos valores.

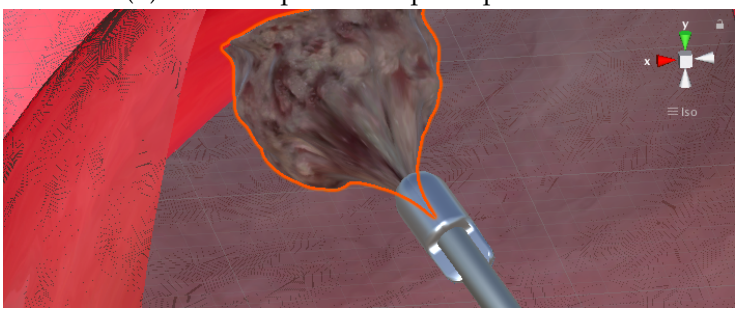
La Figura 3.42 muestra el proceso de una mordida de la pinza.



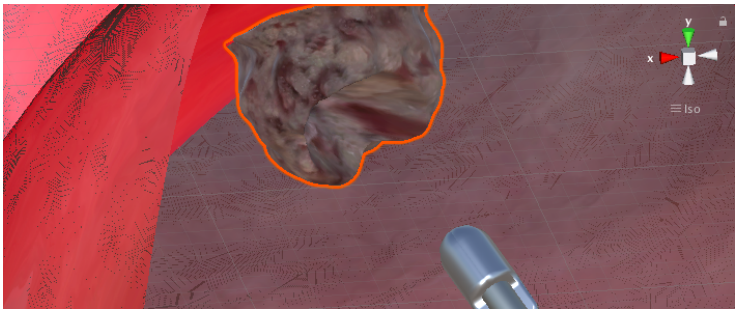
(A) Pinza cerca de lesión endobronquial



(B) Tecla de apertura de pinza presionada



(C) Mordida de lesión



(D) Hendidura de región biopsiada en la lesión

FIGURA 3.42: Pasos de una biopsia

3.4.5. Escena de Resultados

En la escena de resultados se muestra el rendimiento durante la intervención. En caso de querer comenzar desde cero la simulación en la que se ensaya inicialmente se debe presionar la opción *Restart*. En caso de querer seleccionar otro nivel de dificultad, en vez de *Restart*, se selecciona *SelectDifficulty*.

El rendimiento muestra:

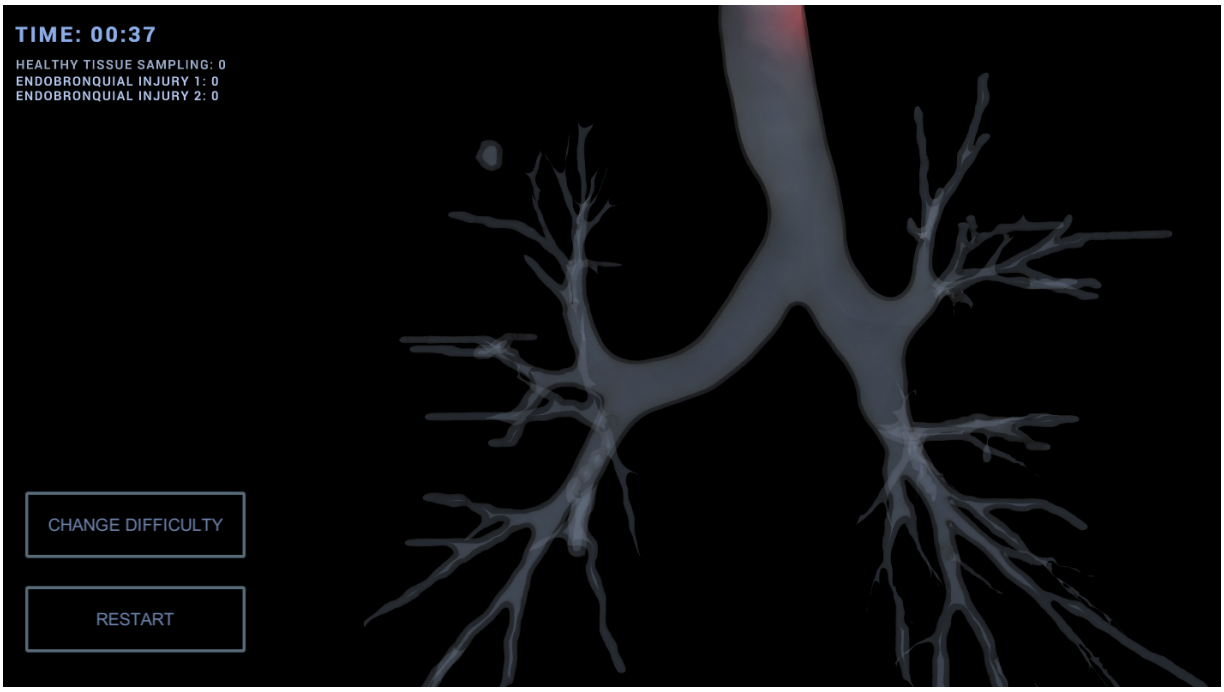


FIGURA 3.43: Escena de Resultados

- **Tiempo de desarrollo del estudio**
- **Mapa de calor**
- **Obtención de muestras en el lugar indicado**, destacando el hecho de que las mismas no deben realizarse en las paredes sanas si no en los granulomas:
    1. **Mapa de número de muestras en lesiones endobronquiales**, distinguidas por lesión.
    2. **Número de muestras en pared normal**

El tiempo de desarrollo de un estudio típico es entre 30 y 60 minutos. En este caso debería durar menos ya que la intubación está contemplada dentro de ese tiempo, acción que no se incorpora a la simulación. Tampoco se contempla el período de adaptación a la anestesia, y otros posibles procedimientos como la colocación de un balón de dilatación.

El porcentaje examinado mostrado en el mapa de calor debería ser lo más alto posible porque, más allá de tener lesiones endobronquiales o no, el médico debería tratar de recorrer toda la vía aérea ya que, a pesar de que en las tomografías aparezcan anormalidades, la visualización del tejido sólo se puede lograr mediante una broncoscopía.

Generalmente se muestrea la lesión más de una vez para una biopsia, el número de muestras en cada lesión debería ser mayor a 1. Por el contrario, el número de muestras en pared normal debería ser 0. Ninguna lesión debería contener 0 biopsias ya que significa que el usuario no ha encontrado la lesión durante la intervención.

3.4.6. *Scripts*

La Figura 3.44 muestra una descripción de qué clases usa cada objeto, las cuales se encuentran en el Anexo. En Unity generalmente no se realizan diagramas de arquitectura debido a que las clases se relacionan con otras clases y también con objetos, por lo que hay veces en que las clases directamente no se relacionan y no tiene sentido mostrar sólo las variables y métodos sin comunicarlas entre ellas.

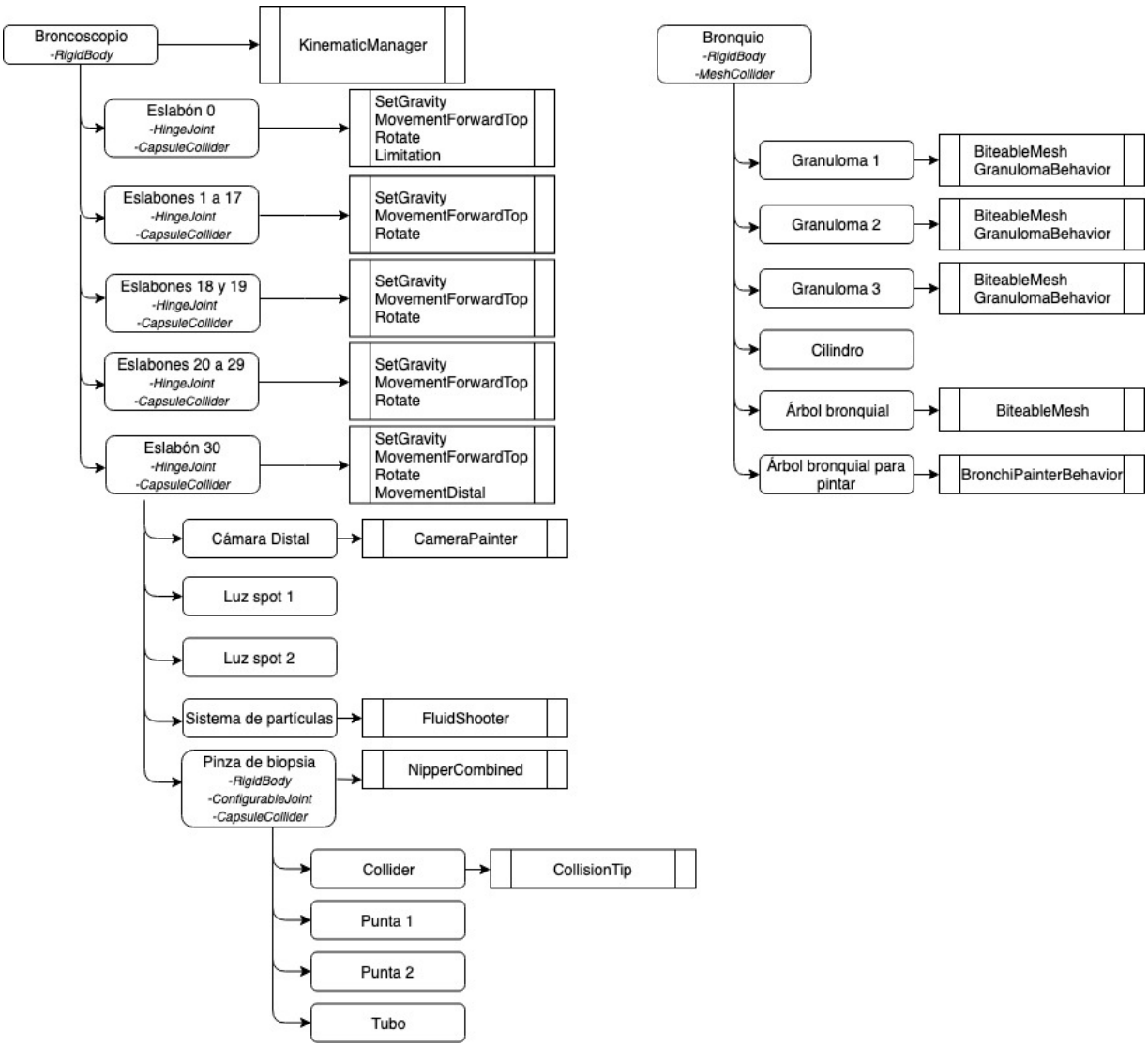


FIGURA 3.44: Clases de cada objeto

3.5. Mejoras a futuro

La elaboración de este trabajo permitió adentrarse en el mundo de la simulación, entendiendo que, para poder representar una situación virtualmente, es necesario comprender peculiaridades de la realidad. Plasmarla, no siempre es sencillo. En primera instancia, porque el motor de físicas tiene limitaciones. A modo de ejemplo, la adición de otro líquido además del agua, como lo puede ser la sangre, implica la incorporación de otro sistema de partículas. El costo computacional de tener dos sistemas de partículas trabajando en simultáneo supera ampliamente los

beneficios de obtener un mayor realismo. Estas relaciones de compromiso se presentan habitualmente en el transcurso del desarrollo de un software.

En muchos casos se suele subestimar la complejidad de recrear un aspecto de la realidad. Las propiedades físicas de un objeto, su estética, su interacción con el medio, sus limitaciones, son algunos de los factores que hay que tener en consideración para plasmarlo virtualmente. Es por ello que no se incorporaron otras patologías aparte de las lesiones endobronquiales, u otros procedimientos, aparte de la biopsia. Se prefirió priorizar la calidad de un aspecto, que abarcar con menor detalle varias situaciones. Sin embargo, fueron sugerencias válidas tanto del Dr. Gonzalo como del Dr. Samolski.

Adicionalmente, no se debe perder de vista que, pese a que en el siguiente trabajo no se presenta el acople del software con un hardware, se considera que a futuro es la mejora que mayores beneficios podría acarrear. En el caso de que se sopese esta alternativa, se superarán las limitaciones del control debido a la carencia *joystick* y así, representar mejor el movimiento de la punta distal del broncoscopio.

Para lograr la integración con un *joystick*, se podría modificar el *script KinematicManager*. A partir de los movimientos individuales destacados en la Tabla 3.4 del Capítulo 3, en donde se muestra las características que varían dependiendo del movimiento deseado, se idea el árbol que ilustra la Figura 3.45 para identificar qué combinaciones resultan en las mismas características deseadas, denominadas 1, 2, 3, 4 y 5. Luego de este esquema, se detallan en la Tabla 3.5 las posibles características que cada opción contiene.

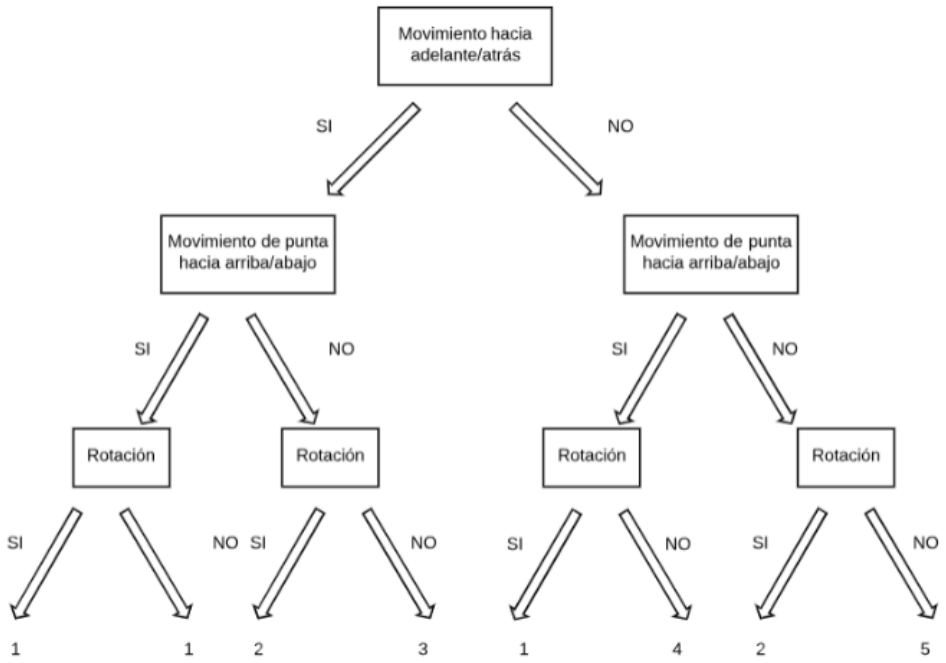


FIGURA 3.45: Esquema de los movimientos que controla *KinematicManager*

	Opción				
	1	**2**	**3**	**4**	**5**
Rotación congelada	No	No	Sí	Sí	Sí
Ángulos de los eslabones distales en 0 grados	No	Sí	Sí	Sí	No
***isKinematic* activo**	No	No	No	No	Sí

CUADRO 3.5: Resumen de lo que debe tener activo cada opción que devenga del *input* de un usuario

Asimismo, una mejora que podría considerarse con esta incorporación es acoplar el hardware a un sistema de devolución háptica que permita al usuario tener una experiencia sensorial semejante a la realidad. Como se desarrolla únicamente el software, no se contempla este aspecto. Sin embargo, aportaría un diferencial que posicionaría al desarrollo al nivel de las últimas tecnologías en torno a la simulación.

Otro aspecto que no se considera, pero que puede ser relevante en el entrenamiento, es la práctica de la intubación (que se corresponde con uno de los primeros pasos a realizar durante una broncoscopía). Este momento suele ser riesgoso y requiere de *expertise* que permita desarrollarla sin dificultades, por lo que ahondar en su entrenamiento implicaría un avance importante en el software.

Por último, el software podría adaptarse como herramienta de planeamiento pre-quirúrgico, repitiendo el ejercicio de edición de tomografías y reconstrucción en 3DSlicer. Cabe destacar que el segmentado de la vía aérea no es tarea sencilla y requeriría de cierta automatización para que el profesional de la salud no tenga que enfrentarse con la tarea de segmentarla.

Capítulo 4

Conclusiones

El siguiente Proyecto Final de grado logra un desarrollo capaz de conceptualizar una práctica médica compleja mediante la profundización tecnica y conocimientoémedico, gracias al trabajo constante en las fronteras del area medico y el área ingenieril y la comunicación satisfactoria interdisciplinaria. Además, se personaliza el resultado mediante la implementación creativa en la gamificación y el diseno final de la interface. Considerando que el proyecto final que se presenta requirió un desarrollo de distintos niveles consecutivos de trabajo, se concluye el cumplimiento de los objetivos respetando el orden de estos niveles, ya que, por ejemplo, sería imposible contar con un sistema de *scoring* sin antes haber sido capaces de incorporar los modelos anatómicos al software.

En primer lugar, se logra alcanzar el objetivo de recrear un escenario anatómico por medio de la simulación. Esto se cumplimenta incorporando modelos anatómicos segmentados de imágenes tomográficas reales, y recreando texturas de los mismos con el fin de lograr una terminación realista. Es por ello que, a simple vista, el software desarrollado logra plasmar la imagen que se observa en una cámara de videobroncoscopía en cercanía con la realidad. Este hecho fue destacado por el Dr. Samolski, quién gentilmente hizo uso del simulador de broncoscopía a modo de prueba del mismo.

Asimismo, otro aspecto igualmente relevante a ponderar es el análisis de los distintos movimientos del broncoscopio, con el fin de encontrar las herramientas necesarias en el motor de físicas que permitieran replicarlos. Se considera el movimiento general del broncoscopio es logrado. Teniendo en consideración que, debido a las limitaciones de los movimientos controlados por medio del teclado en contraposición con los provenientes de un joystick, se debe destacar que el movimiento del extremo distal no es reflejado de manera análoga a la realidad, siendo que podría realizarse ante la incorporación de un hardware de control apropiado.

En el desarrollo de las pinzas de extracción, sus movimientos, su introducción por el canal de trabajo, su apariencia estética, los pasos asociados a una biopsia y demás características constituyentes fueron considerados. Los resultados son realistas, aunque para lograrlos, se acudió a herramientas computacionales ficticias. Un ejemplo es la salida de la pinza de biopsia por el canal de trabajo. El cuerpo de la pinza no se modela con eslabones o como un cuerpo de tamaño fijo con presencia constante dentro del canal de trabajo. Al contrario, se apela al escalado de un tubo y su posterior corrimiento, con el fin de generar en el usuario el efecto de salida desde el canal de trabajo.

Otro aspecto a considerar es el análisis de las colisiones de los objetos partícipes en la escena planteada. Es clara la necesidad de colisiones, con el fin de representar la interacción entre los distintos objetos. Con este fin, se contemplaron distintos tipos del componentes Collider para los GameObjects de Unity, seleccionando cada uno en concordancia con el objeto que acompañan (bronquio, junta de broncoscopio, pinza, lesión endobronquial, etc.).

Con el fin de que el simulador fuese atractivo al usuario, se apeló a la incorporación de diversos elementos. La residente en neumonología, Dra. María Paz Menéndez, comentó que uno de los aspectos iniciales en el aprendizaje de la técnica broncoscópica es el entendimiento de los distintos segmentos de la vía aérea. Es por ello que se incorpora una herramienta pedagógica que permite la visualización de una imagen con referencias que se puede contrastar con la vista frontal de la vía aérea, y, en consecuencia, identificar rápidamente la porción de la misma en la que se ubica el broncoscopio. Asimismo, esta vista frontal posee un mapa de calor que indica la zona en la que el broncoscopio se detuvo, permitiendo trazar el recorrido del mismo. Estas incorporaciones son adicionales a la práctica broncoscópica, por lo que aportan jugabilidad y herramientas educativas adicionales, que distinguen a un simulador educativo controlado de una broncoscopía real.

En cuanto al scoring, se sopesó la posibilidad de construir un índice global de performance que permita comparar las distintas instancias de práctica, tanto para evaluar la evolución individual como para comparar el desempeño entre distintos operarios. Sin embargo, esta idea se descartó, puesto que se prefirió no subjetivizar los parámetros relativos al desempeño como lo son el tiempo de juego, el porcentaje de la vía aérea explorada, logro de los objetivos, al ponderar su aporte en un índice global. Por el contrario, se decidió mostrar al usuario toda la información relativa a su instancia de práctica: área de la vía aérea explorada, mediante un mapa de calor; cantidad de biopsias a lesiones endobronquiales; cantidad de biopsias a tejidos sanos; tiempo de juego.

Los objetivos de máxima planteados, al ser concebidos en un momento inicial, resultaron ser muy ambiciosos. Se optó por un único tipo de lesión endobronquial para poder así profundizar en el procedimiento de biopsia en mayor medida. Además, en primera instancia se consideró la incorporación de Realidad Virtual. Esto se descartó, puesto que no se simula un ambiente en su completitud sino que se plasma lo que se podría observar en una torre de endoscopía (con adicionales de jugabilidad), por lo que no se consideró ni necesario ni relevante al trabajo sumar esta tecnología. Por último, la integración del control de endoscopio con un hardware está contemplada en el código pero no se desarrolló el dispositivo para su acople.

Concluyendo, consideramos que a lo largo del trabajo nuestros conocimientos y habilidades de pensamiento crítico se pusieron en práctica a través de la necesidad de identificar distintas posibles soluciones a un mismo problema, ponerlas en práctica y definir la mejor según diversas variables con distinto peso cada una. A su vez, la búsqueda, los experimentos de prueba y error, y la perseverancia estuvieron constantemente latentes. Fue una experiencia muy enriquecedora no sólo porque aprendimos un lenguaje de programación nuevo y utilizamos una plataforma nunca antes experimentada, y profundizamos en la anatomía respiratoria del cuerpo humano, si no que al interactuar con profesionales de distintas áreas para lograr estas capacidades mejoramos habilidades de comunicación asertiva, comprensión y empatía, contribuyendo a la conexión del conocimiento de carreras disímiles.

Bibliografía

[1] F. Netter, *Atlas de anatomia humana (6a. ed.* Barcelona: Elsevier Health Sciences Spain, 2015.

[2] y. J.-L.-H. E.P. Horvath Jr., S.M. Brooks, *Espirometría en Medicina Ocupacional. Departamento de Salud y Servicios Humanos de los Estados Unidos*, 1981.

[3] I. gaseoso Fisiología. (2015) Vías aéreas bajas. [Online]. Available: https://sites.google.com/site/intercambiogaseosofisiologia/vias-aereas-bajas

[4] J. D. C. J.F. Patiño, E. Celis Rodríguez, *Gases Sanguíneos, Fisiología de la respiración e Insuficiencia respiratoria aguda*. Panamericana, 2015.

[5] F. H. A. Ernst, *PPrinciples and Practice of Interventional Pulmonologys*. New York: Springer, 2013.

[6] M. B. A.B.Verger-Kuhnke, M.A. Reuter, "La biografía de philipp bozzini (1773-1809) un idealista de la endoscopia," *Actas urológicas españolas*, vol. 31, no. 5, pp. 437–444, 2007.

[7] A. M. A.S. Panchabhai, "Historical perspectives of bronchoscopy. connecting the dots," *Annals of the American Thoracic Society*, vol. 12, no. 5, pp. 631–641, 2015.

[8] L. D. M. S. J.Diaz-Mendoza, A.R. Peralta, "Rigid bronchoscopy," *Seminars in Respiratory and Critical Care Medicine*, vol. 39, no. 6, pp. 674–684, 2018.

[9] T. G. H. D. of Anesthesia. (2012) Module 1: The bronchoscope. [Online]. Available: http://pie.med.utoronto.ca/BI/BI_content/BI_module1.html

[10] J. R. G.A. Wilbert, "Planificación de trayectorias en broncoscopías virtuales usando algoritmos rrts," *VII Congreso de Ciencia y Tecnología ESPE 2012*, 2012.

[11] A. Ernst and F. Herth, *Principles and Practice of Interventional Pulmonology*. Springer New York, 2013.

[12] E. F. A. M.-S. F.-B. G. Labarca, I. Caviedes, "Utilidad del ultrasonido endobronquial para el diagnóstico y etapificación del cáncer pulmonar," *Revista médica de Chile*, vol. 145, no. 9, 2017.

[13] A. Ernst, *Introduction to Bronchoscopy*. Cambridge University Press, 2009.

[14] M. W. S. L.-A. C.-G. B. M. J.L. Bacon, M.P. Wilde, "The diagnosis of large airway pathology and the role of rigid bronchoscopy," *Current Respiratory Medicine Reviews*, vol. 9, pp. 11–25, 2013.

[15] PulmonX. (2019) Zephyr® endobronchial valve. [Online]. Available: https://www.pulmonx.com/en/ous/professionals/products/chartis-system/

[16] C. Healthcare. (2019) Cae endovr. [Online]. Available: https://caehealthcare.com/surgical-simulation/endovr/

[17] D. Systems. (2019) Gi mentor. [Online]. Available: https://simbionix.com/simulators/gi-mentor/

[18] O. R. Simulation. (2019) Orsim - features. [Online]. Available: https://www.orsim.co.nz/features

[19] U. H. Network. (2007) Bronchoscopy simulator. [Online]. Available: http://www.thoracic-anesthesia.com/?page_id=2

[20] GooglePlay. (2019) Bronchoguide. [Online]. Available: https://play.google.com/store/apps/details?id=air.androidbronchosimstarter&hl=es_AR

[21] C. Hock-Chuan. (2012) 3d graphics with opengl. [Online]. Available: https://www.ntu.edu.sg/home/ehchua/programming/opengl/cg_basicstheory.html

[22] V. Enfermeria. (2019) Posiciones anatómicas. [Online]. Available: http://mural.uv.es/vipenfer/tecnicas_cuidados_posicionesAnatomicas.html

[23] F. M. USA. (2019) How it works: The angulation system - fibertech medical usa. [Online]. Available: http://fibertechmedical.com/support/how-it-works/how-it-works-the-angulation-system/

[24] Sensus3D. (2019) Sensus3d. [Online]. Available: https://www.sensus3d.com/

[25] C. Ramón and J. Pablo, "Manejo avanzado de la vía aérea," *Revista Médica Clínica Las Condes*, vol. 22, no. 3, pp. 270–279, 2011.

[26] U. de Alcalá. (2019) Aparato respiratorio. [Online]. Available: https://portal.uah.es/portal/page/portal/epd2_asignaturas/asig650019/informacion_academica/Respiratorio.pdf

[27] Anónimo. (2019) Sistema respiratorio. [Online]. Available: https://www.ecured.cu/Sistema_respiratorio

[28] J. F.-T. Hernández, *Fisiología Humana 4ta Ed.* McGraw-Hill, 2010.

[29] R. educativa Partesdel.com. (2017) Partes de la laringe. [Online]. Available: https://www.partesdel.com/partes_de_la_laringe.html.

[30] Leretiel. (2018) Órganos cervicales. [Online]. Available: https://es.slideshare.net/Leretiel/rganos-cervicales-4-presentation

[31] J. R. Palacios, *Sistema respiratorio: anatomía*. Colegio Oficial de enfermeras y enfermeros de Barcelona.

[32] A. J. P. Lindman. (2018) Tracheostomy. [Online]. Available: https://emedicine.medscape.com/article/865068-overview

[33] A. B. B. W. B. M. K. C. C. L. G. G. M. G. P. e. a. M.J. Acuña Navas, E. Arce Rodríguez, "Embriología del desarrollo de los bronquios y el parénquima pulmonar," *Medicina Legal de Costa Rica*, vol. 27, no. 1, 2010.

[34] D. M. Duncker, *Fisiología Pulmonar Introducción*. Facultad de Medicina - Universidad Autónoma del Estado de Morelos, 2017.

[35] A. M. M. M. M. S. e. a. F. Barranco Ruiz, J. Blasco Morilla, *Principios de urgencias, emergencias y cuidados criticos*. Alhulia, 1999.

[36] R. L. L. J.P. Díaz-Jimenez, "Benign tracheal and bronchial stenosis," *Interventions in Pulmonary Medicine*, vol. 1, no. 11, pp. 165–184, 2012.

[37] E. V. R. K.-J. M. M. D.E. Wood, Y. H. Liu, "Airway stenting for malignant and benign tracheobronchial stenosis," *Interventions in Pulmonary Medicine*, vol. 76, no. 1, pp. 167–172, 2003.

[38] A. B. H. C.G. Pantoja Hernández, M.E. Mora Santos, "Traqueotomía: indicaciones, técnica y complicaciones. intubación," *Libro virtual de formación en ORL*, vol. 110, 2018.

[39] J. B. P. Soler, "Physiology of granulomas," *Revue de pneumologie clinique*, vol. 46, no. 6, pp. 257–261, 1993.

[40] O. S. P. J. Korta Murua, "Cuerpos extraños en la vía respiratoria," *Protocolos diagnóstico-terapéuticos de Urgencias Pediátricas SEUP-AEP*, vol. 52, no. 7, pp. 65–73, 2013.

[41] S. F.-B. F. H.-A. E. A. Majida, L. Fernández, "Traqueobroncomalacia," *Archivos de Bronconeumonología*, vol. 46, no. 4, pp. 157–212, 2010.

[42] E. F. J. F. A. A. M. D.B. Casasa, S. Fernández-Bussyc, "Patología obstructiva no maligna de la vía aérea central," *Archivos de Bronconeumonología*, vol. 50, no. 8, pp. 311–372, 2014.

[43] N. Dorland, *Dorland's Medical Dictionary*. Elsevier Health Sciences, 2011.

[44] W. C. G. T. J.R. Biggs, D.-ErZhang, *Molecular Pathology, Chapter 16 - Molecular Basis of Lymphoid and Myeloid Diseases*. Academic Press, 2009.

[45] Anónimo. (2002) Obstrucción bronquial recurrente. [Online]. Available: https://analesdepediatria.org/es-obstruccion-bronquial-recurrente-ii--articulo-13033325

[46] M. Clinic. (2017) Enfermedad pulmonar obstructiva crónica (epoc). [Online]. Available: https://www.mayoclinic.org/es-es/diseases-conditions/copd/symptoms-causes/syc-20353679

[47] A. N. A. F. C. P. I. H. O. R. J. M. R. Cordovilla, E.B. de Miguelb, "Diagnóstico y tratamiento de la hemoptisis," *Archivos de Bronconeumonología*, vol. 52, no. 7, pp. 345–406, 2016.

[48] C. Foundation. (2017) Bronquiectasia. [Online]. Available: https://foundation.chestnet.org/patient-education-resources/bronquiectasia/

[49] O. M. de la Salud. (2016) ¿qué es la tuberculosis y cómo se trata? [Online]. Available: https://www.who.int/features/qa/08/es/

[50] K. S. R. Y. S.-A. T. M. H. F. B. G. M.P. Romero Marín, S.K. Romero Rondon, "Secuelas estructurales y funcionales de tuberculosis pulmonar: una revisión de tema," *Revista Americana de Medicina Respiratoria*, vol. 2, pp. 163–169, 2016.

[51] F. P. C. F. Núñez Miranda, "Embriología del desarrollo de los bronquios y el parénquima pulmonar," *Revista de Actualización Clínica Investiga*, vol. 12, no. 1, pp. 608–610, 2011.

[52] Cancer.net. (2017) Types of endoscopy. [Online]. Available: https://www.cancer.net/navigating-cancer-care/diagnosing-cancer/tests-and-procedures/types-endoscopy

[53] H. S. Ángel Inn. (2017) ¿cuáles son los tipos de endoscopía? [Online]. Available: https://blog.hospitalsanangelinn.mx/tipos-de-endoscopia

[54] A. T. Fernández, "Broncoscopía: Perspectivas de presente y futuro," *Medicina respiratoria*, vol. 1, no. 1, pp. 48–56, 2008.

[55] A. M. P. Lee, E. Kupeli E, "Airway stents," *Clin Chest Med*, vol. 31, no. 1, pp. 141–50, 2010.

[56] J. S. A. T. A.F. Maffey, A. Berlinski, "Broncoscopia flexible en un servicio de neumonología infantil," *Archivos argentinos de pediatría*, vol. 106, no. 1, pp. 19–25, 2008.

[57] A. R. J.P. Díaz-Jimenez, *Interventions in Pulmonary Medicine, Second Edition*. Springer Switzerland, 2018.

[58] A. C. M. J. Pérez Frías, E. Pérez Ruiz, *Fibrobroncoscopía pediátrica*. Ergon, 2006.

[59] S. G. R. T. V. N.-K. T. C. Dooms C, L. Seijo, "Diagnostic bronchoscopy: state of the art." *European Respiratory Society*, vol. 19, no. 117, pp. 229–236, 2010.

[60] P. M. C.T. Bolliger, "Interventional bronchoscopy," *Progress in Respiratory Research*, vol. 30, 1999.

[61] E. d. M. B. E. I. I.-A. S. N. A. U. A. J. P.-I. I. Barredo, I. Zabalza, "Punción aspiración con aguja fina transbronquial guiada por ecografía endobronquial (ebus-tbna) en el hospital galdakao-usansolo," *Póster presentado en la XXXIII Reunión Anual SEAP*, no. 115, p. 8, 2010.

[62] J. F. Aldeyturriagaay, "El lavado broncoalveolar: un procedimiento sencillo que aporta mucha información," *Revista de Patología Respiratoria*, vol. 14, no. 2, pp. 41–65, 2011.

[63] N. Neupane. (2015) Endobronchial ultrasound. [Online]. Available: https://emedicine.medscape.com/article/1970392-overview

[64] G. L. E. F. A. M.-S. F.-B. F. Montufar, L. Del Moral, "Utilidad de la crio-tecnología para uso diagnóstico y terapéutico en neumología intervencionista: crio-biopsia pulmonar transbronquial y crioterapia," *Revista médica de Chile*, vol. 146, no. 9, 2018.

[65] D. V. H. D. E. B. P. S. S.-P. G. A. Alonso, S. Gando, "Nuevos procedimientos broncoscópicos en argentina: Reducción volumétrica pulmonar endoscópica con válvulas endobronquiales," *Revista Americana de Medicina Respiratoria*, vol. 3, pp. 258–268, 2016.

[66] D. J. A. Ernst, G.A. Silvestri, "Interventional pulmonary procedures: Guidelines from the american college of chest physicians," *American College of Chest Physicians*, vol. 123, pp. 1693–1717, 2003.

[67] J. López and L. Spirko, "Simulación, herramienta para la educación médica," *Salud Uninorte*, vol. 23, no. 1, pp. 79–95, 2007.

[68] S. W. V Gough, "Universal tyre test machine," *Proc.9th Int. Technology Congr. FISITA*, pp. 117–137, 1962.

[69] M. S. R. L. C. H. G. R. H. S. Miguel A. Mendoza E., Jhonattan D. Castro P., "Rol de la broncoscopía en el paciente pediátrico," *Neumología Pediátrica*, vol. 13, no. 1, pp. 15–20, 2018.

[70] M. James M. Steckelberg. (2019) Granuloma: ¿qué significa? [Online]. Available: https://www.mayoclinic.org/es-es/granuloma/expert-answers/faq-20057838

[71] Amazon. (2019) Digital image processing medical applications. [Online]. Available: https://www.amazon.com/Digital-Image-Processing-Medical-Applications/dp/0521860857

[72] bronchoscopy. (2019) Segmental anatomy. [Online]. Available: https://www.youtube.com/watch?v=ThYHLG50pH0&t=14s

example.bib

Apéndice A

Scripts del programa en Unity3D

La siguiente tabla muestra los scripts utilizados en el programa y la función que cumplen:

ID	Nombre del Script	Función
1	*BiteableMesh*	Determina el área de mordida, la máxima extensión del tejido al estirarse y lo máximo que se hunde luego.
2	*BoxFriction*	Controla las características de fricción de un material. Por fuera de la vía respiratoria, se coloca una caja rectangular que hace que el broncoscopio se mantenga en posición horizontal en su eje de entrada, por lo que la misma debe tener las características inicializadas en este script.
3	*BronchiPainterBehaviour*	Controla el mapa de calor (*HeatMap*). Fuente: Sensus3D.
4	*CameraPainter*	Pinta los bronquios. En el editor pueden cambiar el **brushSize**. También se puede modificar el **brushDistance**, que es la distancia que considera para pintar desde la cámara del broncoscopio a los bronquios.
5	*collisionTip*	Controla la colisión y mordida de la pinza
6	*explorationManager*	Controla los modelos utilizados en la escena principal, consecuencia del nivel de dificultad elegido
7	*FluidShooter*	Controla la expulsión del agua
8	*GranulomaBehaviour*	Cada lesión contiene este *script* y se utiliza para controlar si se ha realizado una biopsia o no
9	*KinematicManager*	Controla todo el movimiento del broncoscopio
10	*ExtendedFlyCam*	Controla el movimiento de la cámara en la escena de exploración. Fuente: Sensus3D.
11	*Limitation*	Limita el movimiento del broncoscopio hacia atrás. La vía respiratoria posee un cilindro transparente en su apertura, si el eslabón que contiene este **script** toca el cilindro, el broncoscopio no puede seguir yendo hacia atrás. De esta manera el juego se hace más realista; de manera contraria el broncoscopio podría ir hacia atrás a distancia infinita y se alejaría mucho de las vías respiratorias.
12	*MovementDistal*	Controla el movimiento de la porción distal del broncoscopio hacia arriba y abajo
13	*MovementForwardTop*	Controla el movimiento del broncoscopio hacia adelante y hacia atrás
14	*nipperCombined*	Controla los movimientos de la pinza
15	*Rotate*	Controla la rotación del broncoscopio
16	*ScoreManager*	Controla el rendimiento del juego

17	*SetGravity*	Determina la gravedad de los eslabones cuando entran en la vía.
18	*ChangeScene*	Se usa únicamente en la escena de bienvenida porque no tiene información de nivel, y gestiona el cambio de escena entre la esta y la escena de selección de dificultad.
19	*DestroyLevel*	Destruye la variable del nivel seleccionado para cuando se reinicia el juego
20	*GetLevel*	Obtiene el nivel elegido dependiendo del botón seleccionado
21	*SelectCase*	Se utiliza en la escena principal. Mediante la información seleccionada en las escenas anteriores, se activa el caso de bronquio correspondiente y los textos correspondientes a sus granulomas
22	*Timer*	Graba el tiempo
23	*SetDifficulty*	Esta escena previene que el *GameObject* denominado *Level* sea destruido ante cambios de escena. También contiene una función que recibe el nombre de una escena y re-direcciona la cámara principal de la escena actual a dicha escena.
24	*DropdownChoosing*	Este *script* acompaña al botón *Dropdown* y permite que el mismo genere cambios en los videos que se activan según la opción seleccionada del corte tomográfico

CUADRO A.1: *Scripts* del programa

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BiteableMesh : MonoBehaviour
{

    //This script determines the area of a bite, the maximum extension
the tissue stretches when biopsed and the distance the tissue sinks
afterwards


        //this float saves the distance from the center which determines
the circunference to be bitten
    public float verticesDistanceRadius;
    // this float determines the maximum extension of the bite
    public float maxBiteExtension;
    //this float determines the depth up to which the tissue is hollowed
after the biopsy
    public float depthDivider = 10.0f;

    // Start is called before the first frame update
    void Start()
    {
        //the parameters of the biting are set initially

        verticesDistanceRadius = 0.1f;
        maxBiteExtension = 0.35f;
        depthDivider = 4;

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

FIGURA A.1: *BiteableMesh*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BoxFriction : MonoBehaviour {

    //this script describes friction characteristics

    public float dynFriction;
    public float statFriction;
    public float bounce;

    Collider coll;

      // Use this for initialization
      void Start () {

         coll = GetComponent<Collider>();

         dynFriction = 0.1f;
         statFriction = 0.1f;
         bounce = 0;


         coll.material.dynamicFriction = dynFriction;
         coll.material.staticFriction = statFriction;
         coll.material.bounciness = bounce;

         //coll.material = (PhysicMaterial)Resources.Load();

      }

      // Update is called once per frame
      void Update () {

      }
}
```

FIGURA A.2: *BoxFriction*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BronchiPainterBehavior : MonoBehaviour
{

    void Start()
    {
        ScoreManager.Instance.bronchiPainter = gameObject;
    }


}
```

FIGURA A.3: *BronchiPainterBehaviour*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraPainter : MonoBehaviour {

    public int bronchiPainterLayerIndex = 11;
    public GameObject brushCursor, brushContainer; //The cursor that
overlaps the model and our container for the brushes painted
    public Camera canvasCam;  //The camera that looks at the canvas and
The camera that looks at the model
    public Sprite cursorPaint; // Cursor
    public RenderTexture canvasTexture; // Render Texture that looks at
our Base Texture and the painted brushes
    public Material baseMaterial; // The material of our base texture
(Were we will save the painted texture)
    public float brushSize = 6f; // 0.05f; //1.0f; //The size of our brush
    public float distance = 6f; //0.01f; //estaba en 6.5f

  //  Painter_BrushMode mode; //Our painter mode (Paint brushes or decals)
    Color brushColor; //The selected color
    int brushCounter = 0, MAX_BRUSH_COUNT = 1000; //To avoid having
millions of brushes
    bool saving = false; //Flag to check if we are saving the texture

    private void Start()
    {
        RenderTexture.active = canvasTexture;
        Texture2D tex = new Texture2D(canvasTexture.width,
canvasTexture.height, TextureFormat.RGB24, false);
        tex.ReadPixels(new Rect(0, 0, canvasTexture.width,
canvasTexture.height), 0, 0);
        for (int i = 0; i < canvasTexture.width; i++)
            for (int f = 0; f < canvasTexture.height; f++)
                tex.SetPixel(i, f, Color.clear);
        tex.Apply();
        RenderTexture.active = null;
        baseMaterial.SetFloat("_Metallic", 30);


        baseMaterial.mainTexture = tex;
        brushColor = Color.red;  //Updates our painted color with the
selected color
    }

    void Update()
    {
        DoAction();
      /*  if (Input.GetKeyDown(KeyCode.P))
            computePercentage = true;*/
    }

    //The main action, instantiates a brush or decal entity at the clicked
position on the UV map
```

FIGURA A.4: *CameraPainter*

```
    void DoAction()
    {
        if (saving)
            return;
        Vector3 uvWorldPosition = Vector3.zero;
        if (HitTestUVPosition(ref uvWorldPosition))
        {
            GameObject brushObj;
            brushObj =
(GameObject)Instantiate(Resources.Load("CameraPainter/BrushEntity"));
//Paint a brush
            brushObj.GetComponent<SpriteRenderer>().color = brushColor;
//Set the brush color
            brushColor.a = brushSize * 2.0f; // Brushes have alpha to have
a merging effect when painted over.
            brushObj.transform.parent = brushContainer.transform; //Add
the brush to our container to be wiped later
            brushObj.transform.localPosition = uvWorldPosition; //The
position of the brush (in the UVMap)
            brushObj.transform.localScale = Vector3.one * brushSize;//The
size of the brush
        }
        brushCounter++; //Add to the max brushes
        if (brushCounter >= MAX_BRUSH_COUNT)
        { //If we reach the max brushes available, flatten the texture and
clear the brushes
            brushCursor.SetActive(false);
            saving = true;
            Invoke("SaveTexture", 0.1f);

        }
    }
    //To update at realtime the painting cursor on the mesh
    void UpdateBrushCursor()
    {
        Vector3 uvWorldPosition = Vector3.zero;
        if (HitTestUVPosition(ref uvWorldPosition) && !saving)
        {
            brushCursor.SetActive(true);
            brushCursor.transform.position = uvWorldPosition +
brushContainer.transform.position;
        }
        else
        {
            brushCursor.SetActive(false);
        }
    }

    public void ComputePercentage()
    {
        RenderTexture.active = canvasTexture;
        Texture2D tex = new Texture2D(canvasTexture.width,
canvasTexture.height);
```

FIGURA A.5: *CameraPainter*

```
        tex.ReadPixels(new Rect(0, 0, canvasTexture.width,
canvasTexture.height), 0, 0);
        tex.Apply();
        float percentageCovered = 0;

        for (int i = 0; i < canvasTexture.width; i++)
            for (int f = 0; f < canvasTexture.height; f++)
                percentageCovered += tex.GetPixel(i, f).r;

        RenderTexture.active = null;

        int percentage = (int)((percentageCovered / (canvasTexture.width *
canvasTexture.height)) * 100);

        ScoreManager.Instance.ShowPercentage(percentage < 100 ? percentage
: 100);
    }

    //Returns the position on the texuremap according to a hit in the mesh
collider
    bool HitTestUVPosition(ref Vector3 uvWorldPosition)
    {
        RaycastHit[] hits = Physics.RaycastAll(transform.position,
transform.forward, distance);
        foreach(RaycastHit hit in hits)
        {
            if (hit.collider.gameObject.layer == bronchiPainterLayerIndex)
            {
                MeshCollider meshCollider = hit.collider as MeshCollider;
                if (meshCollider == null || meshCollider.sharedMesh ==
null)
                    return false;
                Vector2 pixelUV = new Vector2(hit.textureCoord.x,
hit.textureCoord.y);
                uvWorldPosition.x = pixelUV.x -
canvasCam.orthographicSize;//To center the UV on X
                uvWorldPosition.y = pixelUV.y -
canvasCam.orthographicSize;//To center the UV on Y
                uvWorldPosition.z = 0.0f;
                return true;
            }
        }
        return false;
    }

    //Sets the base material with a our canvas texture, then removes all
our brushes
    void SaveTexture()
    {
        brushCounter = 0;
        System.DateTime date = System.DateTime.Now;
        RenderTexture.active = canvasTexture;
        Texture2D tex = new Texture2D(canvasTexture.width,
canvasTexture.height, TextureFormat.RGB24, false);
```

FIGURA A.6: *CameraPainter*

```
        tex.ReadPixels(new Rect(0, 0, canvasTexture.width,
canvasTexture.height), 0, 0);
        tex.Apply();
        RenderTexture.active = null;
        baseMaterial.mainTexture = tex; //Put the painted texture as the
base
        foreach (Transform child in brushContainer.transform)
        {//Clear brushes
            Destroy(child.gameObject);
        }
        //StartCoroutine ("SaveTextureToFile"); //Do you want to save the
texture? This is your method!
        Invoke("ShowCursor", 0.1f);
    }



    //Show again the user cursor (To avoid saving it to the texture)
    void ShowCursor()
    {
        saving = false;
        brushCursor.SetActive(true);

    }
}
```

FIGURA A.7: *CameraPainter*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class collisionTip : MonoBehaviour {

    //This script controls the collision and bite of the clamp

   // flag which is false when a collision is entered and stayed, and is
false when a collision is exited
   //it is used by the script nipperCombined
    public bool flag;

    //flag which states wether the collider has a vertex attached or not.
False means it has no vertex attached.
    //True means it has vertex attached, therefore cannot bite again.
    public static bool vertexAttached;



    // Vectors for global and local point of contact between the collider
and the bronchi wall, and its normal
    public Vector3 contactPointGlobal;
    public Vector3 contactPointLocal;
    public Vector3 contactNormal;


    //list which will contain the indices of the vertices closest to the
contact point
    public List<int> closestVerticesIndices;

    //list which will contain the vectors for vertices
    public Vector3[] biteableVertices;

    //list which will contain the original vectors of the bronchi wall
previous to be bitten
    public Vector3[] biteableVerticesOriginal;

    // keycode for bite
    public KeyCode bite;

    //MeshFilter which can be bitten
    private MeshFilter biteableMesh;

    // Area which can be bitten, determined by wether it contains the
script BiteableMesh or not
    private BiteableMesh biteableObject;



     public static collisionTip Instance;

     void Awake()
     {
         if (!Instance)
         {
             Instance = this;
         }
     }
```

FIGURA A.8: *collisionTip*

```
    // Use this for initialization
    void Start () {

       // initialization of the flags
        flag = true;

        //initialization of the flag vertexAttached false
        vertexAttached = false;

        //initialization of the list
        closestVerticesIndices = new List<int>();

       // initialization of the keycode for bite
        bite = KeyCode.K;


    }

      // Update is called once per frame
      void Update () {



      }

    //when the collider enters a collision
    void OnCollisionEnter(Collision collision)
    {


        // if the gameObject it collides with has a MeshCollider (is the
bronchi) and is has no vertex attached,
        //the lists are completed with the vectors of the mesh
        if (collision.gameObject.GetComponent<MeshCollider>() &&
!vertexAttached)
        {
            //the mesh which is collided and can be bitten is obtained
            biteableMesh =
collision.gameObject.GetComponent<MeshFilter>();

            //The list biteableVertices is completed with the vertices of
the biteableMesh
            biteableVertices = biteableMesh.mesh.vertices;

            //The list biteableVerticesOriginal is completed with the
vertices of the biteableMesh
            biteableVerticesOriginal = biteableMesh.mesh.vertices;

            //The object which can be bitten is obtained
            biteableObject =
collision.gameObject.GetComponent<BiteableMesh>();

            //since the colliderentered a collision, the movement of the
bronchoscope is unabled so that
            // it does not continue sliding
            KinematicManager.Instance.cantMove();

            //
            flag = false;
```

FIGURA A.9: *collisionTip*

```
        }

    }

    //when the collider stays in a collision
    private void OnCollisionStay(Collision collision)
    {

        // if the gameObject it collides with has a MeshCollider (is the
bronchi) and is has no vertex attached,
        //the lists are completed with the vectors of the mesh
        if (collision.gameObject.GetComponent<MeshCollider>() &&
!vertexAttached)
        {

            //if it collides with the bronchi the bronchoscope movement
is unabled so that is does not continue sliding
            KinematicManager.Instance.cantMove();


        //if is collides with something biteable (bronchi or lesion) and
does not have anything attached to it,
            //it looks for the contact point vector
        if (biteableObject && !vertexAttached)
        {

                //the mesh which is collided and can be bitten is
obtained
            biteableMesh = biteableObject.GetComponent<MeshFilter>();

            //the first global contact point is saved
            contactPointGlobal = collision.contacts[0].point;

            //the first global contact point is transformed to local
            contactPointLocal =
collision.gameObject.transform.InverseTransformPoint(collision.contacts[0
].point);


            // the local axis of the bronchi are not aligned with the
world so the normal vector is obtained
            Vector3 contactNormalAvg = Vector3.zero;

            //each contact in the collision is transformed to a normal
vector
            foreach(ContactPoint point in collision.contacts)
            contactNormalAvg += point.normal;

            contactNormal =
collision.gameObject.transform.InverseTransformDirection(collision.contac
ts[0].normal);

          //if the key to bite is pressed and there are no vertex
attached
            if (Input.GetKeyUp(KeyCode.K)&& vertexAttached == false)
            {

            //The flag vertexAttached is set true
            vertexAttached = true;
```

FIGURA A.10: *collisionTip*

```
            // The courutine DeformVertices is called
             StartCoroutine("DeformVertices");

             }



             }
        }
    }

    //when the collider exits a collision
    private void OnCollisionExit(Collision collision)
    {
        // There are several objects to bite so an exit of the object
where it is has to be done
        // The reason is because if one is touching the lesion and
touches as well the bronqui, when one bites,
        // the OnCollisionExit of the lesion cannot step/overlap the
OnCollisionEnter of the bronchi

        //if the object it exists is biteable and there is no vertex
attached, everything is set null
        // the bronchoscope is enabled to move
        if (biteableObject && biteableObject ==
collision.gameObject.GetComponent<BiteableMesh>() && !vertexAttached)
        {
            biteableMesh = null;
            biteableVertices = null;
            biteableVerticesOriginal = null;
            biteableObject = null;
            KinematicManager.Instance.canMove();
        }

        flag = true;

    }


    //Coroutine DeformVertices - the benefit of  a coroutine is to stop
the execution until sometime or certain condition is met
    IEnumerator DeformVertices()
    {


        // the index of the vectors near the contactpoint are added to
the list closestVerticesIndces
        // if the distance of the vector is less than the value stated,
the index is saved in the list biteableVertices
        for (int i = 0; i < biteableVertices.Length; i++)
        {
            float distance = Vector3.Distance(contactPointLocal,
biteableVertices[i]);
            if (distance < biteableObject.verticesDistanceRadius)
                closestVerticesIndices.Add(i);
        }

        //while the vertices of the colliding object are attached to the
collider, the vertices of the colliding object are streched and moved
        //to the position of the collider
```

FIGURA A.11: *collisionTip*

```
        while (vertexAttached)
        {

            Vector3 localNipperTipPosition =
biteableMesh.transform.InverseTransformPoint(transform.position +
transform.up * transform.localScale.y);

            float sampleDistanceLocal =
Vector3.Distance(contactPointLocal, localNipperTipPosition);

            //  the vertices are streched as long as the extension is
less than a certain value
            if (sampleDistanceLocal <= biteableObject.maxBiteExtension)
            {
                foreach (int vertex in closestVerticesIndices)
                    biteableVertices[vertex] =
biteableVerticesOriginal[vertex] + (localNipperTipPosition -
biteableVerticesOriginal[vertex]).normalized *
                     Vector3.Distance(localNipperTipPosition,
biteableVerticesOriginal[vertex]) *
                    (1.0f -
Vector3.Distance(biteableVerticesOriginal[vertex], contactPointLocal) /
biteableObject.verticesDistanceRadius);
            }


            // if the tissue is stretched more than a certain value, or
the user presses the open key, the tissue is no longer attached
            // the values of the vectors of the bitten area are obtained
by substracting the original vector by a certain value so as to show
depth
            // the vectors of the area which is not bitten remains the
same
            if (sampleDistanceLocal > biteableObject.maxBiteExtension ||
Input.GetKey(KeyCode.K))
            {

                foreach (int vertex in closestVerticesIndices)
                    biteableVertices[vertex] =
biteableVerticesOriginal[vertex] - (contactNormal /
biteableObject.depthDivider) *
                    (1.0f -
Vector3.Distance(biteableVerticesOriginal[vertex], contactPointLocal) /
biteableObject.verticesDistanceRadius);

                // since the tissue is no longer attached, the flag
vertexAttached is set false
                vertexAttached = false;

                flag = true;
            }

            biteableMesh.mesh.vertices = biteableVertices;
            biteableMesh.mesh.RecalculateNormals();

            yield return null;
        }


        // if a lesion is bitten, the  ScoreManager is notified
```

FIGURA A.12: *collisionTip*

```
        GranulomaBehavior granuloma =
biteableObject.GetComponent<GranulomaBehavior>();
        if (granuloma)
        {
            ScoreManager.Instance.ComputeBiopsy(granuloma.granulomaId);
            granuloma.biopsed = true;
        }


        // if healthy tissue is bitten, the  ScoreManager is notified
        if (!granuloma )
        {
            ScoreManager.Instance.ComputeHealthy();
        }


        //the list closestVerticesIndices is cleared
        closestVerticesIndices.Clear();

        //the Coroutine is stopped
        StopCoroutine("DeformVertices");



    }

    // this code can be used in a future to place a scar in the place
where the tissue is bitten
    /*
    public void placeScar()
    {
        GameObject temporaryScar;
        temporaryScar = Instantiate(scar.gameObject , contactPointGlobal-
transform.up/2, Quaternion.Euler(contactNormal)) as GameObject;


    }
    */

}
```

FIGURA A.13: *collisionTip*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class explorationManager : MonoBehaviour
{
    //this Script controls the UI texts which are displayed as well as
the models and the videos to be shown

    //string that stores the ammount of biopses done
    public string biopse;
    // string that stores the difficulty level (low, medium or high)
    public string difficulty;
    //string that saves the value obtained from the GameObject level
which stores the level information along the scenes
    public string difficultyLevel;
    //UI text which displays biopse information
    public Text biopseText;
    //UI text which displays difficulty information
    public Text difficultText;


    //GameObject for the undestructable GameObject "level" to be assigned
    public GameObject levelObject;

    //GameObjects for the models to be attached
    public GameObject case1Low;
    public GameObject case2Low;
    public GameObject case3Low;
    public GameObject case1Med;
    public GameObject case2Med;
    public GameObject case3Med;
    public GameObject case1High;
    public GameObject case2High;
    public GameObject case3High;

    //GameObjects containing the different CT videos of the four cases
    public GameObject videosCase1;
    public GameObject videosCase10;
    public GameObject videosCase20;
    public GameObject videosCase60;

    //list of models
    public List<GameObject> objectPool;
    //index to reference a model in the list
    private int currentIndex = 0;


    void Start()
    {

        // find the GameObject whose tag is "level" and assigns it to the
GameObject levelObject
        levelObject = GameObject.FindGameObjectWithTag("level");
```

FIGURA A.14: *explorationManager*

```
        //gets the difficulty level string from the Level game object and
assigns it to the variable difficultyLevel
        difficultyLevel = levelObject.GetComponent<GetLevel>().level;

        //sets initially false all the groups of CT videos
        videosCase1.SetActive(false);
        videosCase10.SetActive(false);
        videosCase20.SetActive(false);
        videosCase60.SetActive(false);

        //if the level information corresponds to "low"
        if (difficultyLevel == "low")
        {
            //add low difficulty models to the objectPool list
            objectPool.Add(case1Low);
            objectPool.Add(case2Low);
            objectPool.Add(case3Low);

            //set initially false all models from the list
            objectPool[1].SetActive(false);
            objectPool[2].SetActive(false);
            objectPool[0].SetActive(false);

            //select a number from 0 to 2 randonly and save it into
newIndex variable
            int newIndex = Random.Range(0, 2);
            //saves newIndex value into currentIndex variable
            currentIndex = newIndex;
            //sets the object whose list index is currentIndex to true
            objectPool[currentIndex].SetActive(true);
            //saves the information of the case chosen randomly in a
variable of the Level GameObject
            levelObject.GetComponent<GetLevel>().caseIndex =
currentIndex;

            //depending on the case chosen randomly, the set of videos
that are set active
            if (currentIndex==0)
            {
                videosCase1.SetActive(true);
            }
            else if (currentIndex == 1)
            {
                videosCase10.SetActive(true);
            }
            else if (currentIndex == 2)
            {
                videosCase60.SetActive(true);
            }

            //set biopse string to "1" and difficulty string to "HIGH"
            biopse = "1";
            difficulty = "LOW";
        }


        //if the level information corresponds to "low"
        if (difficultyLevel == "medium")
        {
            //add medium difficulty models to the objectPool list
```

FIGURA A.15: *explorationManager*

```
            objectPool.Add(case1Med);
            objectPool.Add(case2Med);
            objectPool.Add(case3Med);

            //set initially false all models from the list
            objectPool[1].SetActive(false);
            objectPool[2].SetActive(false);
            objectPool[0].SetActive(false);

            //select a number from 0 to 2 randonly and save it into
newIndex variable
            int newIndex = Random.Range(0, 2);
            //saves newIndex value into currentIndex variable
            currentIndex = newIndex;
            //sets the object whose list index is currentIndex to true
            objectPool[currentIndex].SetActive(true);
            //saves the information of the case chosen randomly in a
variable of the Level GameObject
            levelObject.GetComponent<GetLevel>().caseIndex =
currentIndex;

            //depending on the case chosen randomly, the set of videos
that are set active
            if (currentIndex == 0)
            {
                videosCase1.SetActive(true);
            }
            else if (currentIndex == 1)
            {
                videosCase10.SetActive(true);
            }
            else if (currentIndex == 2)
            {
                videosCase20.SetActive(true);
            }

            //set biopse string to "2" and difficulty string to "MEDIUM"
            biopse = "2";
            difficulty = "MEDIUM";
        }


        //if the level information corresponds to "high"
        if (difficultyLevel == "high")
        {
            //add medium difficulty models to the objectPool list
            objectPool.Add(case1High);
            objectPool.Add(case2High);
            objectPool.Add(case3High);
            //set initially false all models from the list
            objectPool[1].SetActive(false);
            objectPool[2].SetActive(false);
            objectPool[0].SetActive(false);
            //select a number from 0 to 2 randonly and save it into
newIndex variable
            int newIndex = Random.Range(0, 2);
            //saves newIndex value into currentIndex variable
            currentIndex = newIndex;
            //sets the object whose list index is currentIndex to true
            objectPool[currentIndex].SetActive(true);
```

FIGURA A.16: *explorationManager*

```
            //saves the information of the case chosen randomly in a
variable of the Level GameObject
            levelObject.GetComponent<GetLevel>().caseIndex =
currentIndex;


            //depending on the case chosen randomly, the set of videos
that are set active and the strings to be stored in the variables
difficulty and biopses
            if (currentIndex == 1)
            {
                biopse = "2";
                videosCase20.SetActive(true);
            }

            difficulty = "HIGH";

            if (currentIndex == 0)
            {
                videosCase10.SetActive(true);
                biopse = "3";
            }
            else if (currentIndex == 2)
            {
                videosCase60.SetActive(true);
                biopse = "3";
            }

        }

        //sets the UI text biopseText and difficultyText
      //  biopseText.text = "BIOPSES TO BE PERFORMED: " + biopse;
        difficultText.text = "DIFFICULTY: " + difficulty;



    }

}
```

FIGURA A.17: *explorationManager*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FluidShooter : MonoBehaviour {

    //This script controls the fluid shooting

    //particle system to be used
    private ParticleSystem ps;


      void Start () {
        //Fetch the particle system component from the GameObject
        ps = GetComponent<ParticleSystem>();
        //assigns the variable emission to the emission behavior of the
particle system
        var emission = ps.emission;
        //sets the emission in false
        emission.enabled = false;
      }

      // Update is called once per frame
      void Update () {

        //if the L key is pressed
        if (Input.GetKeyDown(KeyCode.L))
        {
           //emission of the particle system is true
            var emission = ps.emission;
            emission.enabled = true;
        }
        //if the L key is pressed up
        if (Input.GetKeyUp(KeyCode.L))
        {
            //emission of the particle system is false
            var emission = ps.emission;
            emission.enabled = false;
        }

      }
}
```

FIGURA A.18: *FluidShooter*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GranulomaBehavior : MonoBehaviour
{

    //Each lesion contains this script and is used to control whether it
has been biopsed or not

    //initial parameters of the granuloma that has attached this script
    public int granulomaId = 0;
    public bool biopsed = false;

    void Start()
    {
        //Adds into ScoreManager the granuloma that has attached this
script as an instance
        ScoreManager.Instance.granulomas.Add(this);
    }


}
```

FIGURA A.19: *GranulomaBehaviour*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class KinematicManager : MonoBehaviour {

    //controls the whole movement of broncoscope

    //variables for the keys that will control each movement of the
bronchoscope
    public KeyCode forward;
    public KeyCode backwards;
    public KeyCode rotateRight;
    public KeyCode rotateLeft;
    public KeyCode moveUp;
    public KeyCode moveDown;
    public KeyCode nipperOut;
    public KeyCode nipperIn;

   //GameObject used as a threshold to determine whether a piece of the
bronchoscope needs to have gravity or not
    public GameObject umbral;


    public static KinematicManager Instance;

    //Awake is always called before any Start functions
    void Awake()
    {
        //If instance already exists and it's not this:
        if (!Instance)
        {
            //set instance to this
            Instance = this;
        }
    }

    //variables to save each piece ob bronchoscope's position
    public double positionrb;
    public double positionrb1;
    public double positionrb2;
    public double positionrb3;
    public double positionrb4;
    public double positionrb5;
    public double positionrb6;
    public double positionrb7;
    public double positionrb8;
    public double positionrb9;
    public double positionrb10;
    public double positionrb11;
    public double positionrb12;
    public double positionrb13;
    public double positionrb14;
    public double positionrb15;
    public double positionrb16;
    public double positionrb17;
    public double positionrb18;
    public double positionrb19;
    public double positionrb20;
    public double positionrb21;
```

FIGURA A.20: *KinematicManager*

```
    public double positionrb22;
    public double positionrb23;
    public double positionrb24;
    public double positionrb25;
    public double positionrb26;
    public double positionrb27;
    public double positionrb28;
    public double positionrb29;
    public double positionrb30;
    public double positionumbral;

    public List<GameObject> outsideList; //List with joints outside the
airway
    public List<GameObject> insideList; //List with joints inside the
airway
    public List<GameObject> distalList; //List with joints in the distal
portion
    public List<GameObject> proximalList; //List with joints in the
proximal portion

    //GameObjects for all the shackles/pieces ob bronchoscope to be
attached
    public GameObject gameob;
    public GameObject gameob1;
    public GameObject gameob2;
    public GameObject gameob3;
    public GameObject gameob4;
    public GameObject gameob5;
    public GameObject gameob6;
    public GameObject gameob7;
    public GameObject gameob8;
    public GameObject gameob9;
    public GameObject gameob10;
    public GameObject gameob11;
    public GameObject gameob12;
    public GameObject gameob13;
    public GameObject gameob14;
    public GameObject gameob15;
    public GameObject gameob16;
    public GameObject gameob17;
    public GameObject gameob18;
    public GameObject gameob19;
    public GameObject gameob20;
    public GameObject gameob21;
    public GameObject gameob22;
    public GameObject gameob23;
    public GameObject gameob24;
    public GameObject gameob25;
    public GameObject gameob26;
    public GameObject gameob27;
    public GameObject gameob28;
    public GameObject gameob29;
    public GameObject gameob30;

    //rigidbodies of every shackle GameObject
    public Rigidbody rb;
    public Rigidbody rb1;
    public Rigidbody rb2;
    public Rigidbody rb3;
    public Rigidbody rb4;
```

FIGURA A.21: *KinematicManager*

```
public Rigidbody rb5;
public Rigidbody rb6;
public Rigidbody rb7;
public Rigidbody rb8;
public Rigidbody rb9;
public Rigidbody rb10;
public Rigidbody rb11;
public Rigidbody rb12;
public Rigidbody rb13;
public Rigidbody rb14;
public Rigidbody rb15;
public Rigidbody rb16;
public Rigidbody rb17;
public Rigidbody rb18;
public Rigidbody rb19;
public Rigidbody rb20;
public Rigidbody rb21;
public Rigidbody rb22;
public Rigidbody rb23;
public Rigidbody rb24;
public Rigidbody rb25;
public Rigidbody rb26;
public Rigidbody rb27;
public Rigidbody rb28;
public Rigidbody rb29;
public Rigidbody rb30;

//HingeJoints of every shackle GameObject
public HingeJoint hj;
public HingeJoint hj1;
public HingeJoint hj2;
public HingeJoint hj3;
public HingeJoint hj4;
public HingeJoint hj5;
public HingeJoint hj6;
public HingeJoint hj7;
public HingeJoint hj8;
public HingeJoint hj9;
public HingeJoint hj10;
public HingeJoint hj11;
public HingeJoint hj12;
public HingeJoint hj13;
public HingeJoint hj14;
public HingeJoint hj15;
public HingeJoint hj16;
public HingeJoint hj17;
public HingeJoint hj18;
public HingeJoint hj19;
public HingeJoint hj20;
public HingeJoint hj21;
public HingeJoint hj22;
public HingeJoint hj23;
public HingeJoint hj24;
public HingeJoint hj25;
public HingeJoint hj26;
public HingeJoint hj27;
public HingeJoint hj28;
public HingeJoint hj29;
public HingeJoint hj30;
```

FIGURA A.22: *KinematicManager*

```
void Start()
{

    //assigns keys to variables

    forward = KeyCode.Space;
    backwards = KeyCode.B;
    rotateRight = KeyCode.D;
    rotateLeft = KeyCode.A;
    moveUp = KeyCode.UpArrow;
    moveDown = KeyCode.DownArrow;
    nipperIn = KeyCode.I;
    nipperOut = KeyCode.O;


    //Fetch the HingeJoint component from the GameObjects

    hj = gameob.GetComponent<HingeJoint>();
    hj1 = gameob1.GetComponent<HingeJoint>();
    hj2 = gameob2.GetComponent<HingeJoint>();
    hj3 = gameob3.GetComponent<HingeJoint>();
    hj4 = gameob4.GetComponent<HingeJoint>();
    hj5 = gameob5.GetComponent<HingeJoint>();
    hj6 = gameob6.GetComponent<HingeJoint>();
    hj7 = gameob7.GetComponent<HingeJoint>();
    hj8 = gameob8.GetComponent<HingeJoint>();
    hj9 = gameob9.GetComponent<HingeJoint>();
    hj10 = gameob10.GetComponent<HingeJoint>();
    hj11 = gameob11.GetComponent<HingeJoint>();
    hj12 = gameob12.GetComponent<HingeJoint>();
    hj13 = gameob13.GetComponent<HingeJoint>();
    hj14 = gameob14.GetComponent<HingeJoint>();
    hj15 = gameob15.GetComponent<HingeJoint>();
    hj16 = gameob16.GetComponent<HingeJoint>();
    hj17 = gameob17.GetComponent<HingeJoint>();
    hj18 = gameob18.GetComponent<HingeJoint>();
    hj19 = gameob19.GetComponent<HingeJoint>();
    hj20 = gameob20.GetComponent<HingeJoint>();
    hj21 = gameob21.GetComponent<HingeJoint>();
    hj22 = gameob22.GetComponent<HingeJoint>();
    hj23 = gameob23.GetComponent<HingeJoint>();
    hj24 = gameob24.GetComponent<HingeJoint>();
    hj25 = gameob25.GetComponent<HingeJoint>();
    hj26 = gameob26.GetComponent<HingeJoint>();
    hj27 = gameob27.GetComponent<HingeJoint>();
    hj28 = gameob28.GetComponent<HingeJoint>();
    hj29 = gameob29.GetComponent<HingeJoint>();
    hj30 = gameob30.GetComponent<HingeJoint>();

    //Fetch the RigidBody component from the GameObjects

    rb = gameob.GetComponent<Rigidbody>();
    rb1 = gameob1.GetComponent<Rigidbody>();
    rb2 = gameob2.GetComponent<Rigidbody>();
    rb3 = gameob3.GetComponent<Rigidbody>();
```

FIGURA A.23: *KinematicManager*

```
        rb4 = gameob4.GetComponent<Rigidbody>();
        rb5 = gameob5.GetComponent<Rigidbody>();
        rb6 = gameob6.GetComponent<Rigidbody>();
        rb7 = gameob7.GetComponent<Rigidbody>();
        rb8 = gameob8.GetComponent<Rigidbody>();
        rb9 = gameob9.GetComponent<Rigidbody>();
        rb10 = gameob10.GetComponent<Rigidbody>();
        rb11 = gameob11.GetComponent<Rigidbody>();
        rb12 = gameob12.GetComponent<Rigidbody>();
        rb13 = gameob13.GetComponent<Rigidbody>();
        rb14 = gameob14.GetComponent<Rigidbody>();
        rb15 = gameob15.GetComponent<Rigidbody>();
        rb16 = gameob16.GetComponent<Rigidbody>();
        rb17 = gameob17.GetComponent<Rigidbody>();
        rb18 = gameob18.GetComponent<Rigidbody>();
        rb19 = gameob19.GetComponent<Rigidbody>();
        rb20 = gameob20.GetComponent<Rigidbody>();
        rb21 = gameob21.GetComponent<Rigidbody>();
        rb22 = gameob22.GetComponent<Rigidbody>();
        rb23 = gameob23.GetComponent<Rigidbody>();
        rb24 = gameob24.GetComponent<Rigidbody>();
        rb25 = gameob25.GetComponent<Rigidbody>();
        rb26 = gameob26.GetComponent<Rigidbody>();
        rb27 = gameob27.GetComponent<Rigidbody>();
        rb28 = gameob28.GetComponent<Rigidbody>();
        rb29 = gameob29.GetComponent<Rigidbody>();
        rb30 = gameob30.GetComponent<Rigidbody>();

        //obtains the z position of the rigidbodies and saves them into
their corresponding variables
        positionrb = rb.position.z;
        positionrb1 = rb1.position.z;
        positionrb2 = rb2.position.z;
        positionrb3 = rb3.position.z;
        positionrb4 = rb4.position.z;
        positionrb5 = rb5.position.z;
        positionrb6 = rb6.position.z;
        positionrb7 = rb7.position.z;
        positionrb8 = rb8.position.z;
        positionrb9 = rb9.position.z;
        positionrb10 = rb10.position.z;
        positionrb11 = rb11.position.z;
        positionrb12 = rb12.position.z;
        positionrb13 = rb13.position.z;
        positionrb14 = rb14.position.z;
        positionrb15 = rb15.position.z;
        positionrb16 = rb16.position.z;
        positionrb17 = rb17.position.z;
        positionrb18 = rb18.position.z;
        positionrb19 = rb19.position.z;
        positionrb20 = rb20.position.z;
        positionrb21 = rb21.position.z;
        positionrb22 = rb22.position.z;
        positionrb23 = rb23.position.z;
        positionrb24 = rb24.position.z;
        positionrb25 = rb25.position.z;
        positionrb26 = rb26.position.z;
        positionrb27 = rb27.position.z;
        positionrb28 = rb28.position.z;
        positionrb29 = rb29.position.z;
```

FIGURA A.24: *KinematicManager*

```
        positionrb30 = rb30.position.z;


        //adds to the outside list all the GameObjects as they start
initially outside the airway
        outsideList.Add(gameob);
        outsideList.Add(gameob1);
        outsideList.Add(gameob2);
        outsideList.Add(gameob3);
        outsideList.Add(gameob4);
        outsideList.Add(gameob5);
        outsideList.Add(gameob6);
        outsideList.Add(gameob7);
        outsideList.Add(gameob8);
        outsideList.Add(gameob9);
        outsideList.Add(gameob10);
        outsideList.Add(gameob11);
        outsideList.Add(gameob12);
        outsideList.Add(gameob13);
        outsideList.Add(gameob14);
        outsideList.Add(gameob15);
        outsideList.Add(gameob16);
        outsideList.Add(gameob17);
        outsideList.Add(gameob18);
        outsideList.Add(gameob19);
        outsideList.Add(gameob20);
        outsideList.Add(gameob21);
        outsideList.Add(gameob22);
        outsideList.Add(gameob23);
        outsideList.Add(gameob24);
        outsideList.Add(gameob25);
        outsideList.Add(gameob26);
        outsideList.Add(gameob27);
        outsideList.Add(gameob28);
        outsideList.Add(gameob29);
        outsideList.Add(gameob30);


        //adds to the proximal list the proximal GameObjects
        proximalList.Add(gameob);
        proximalList.Add(gameob1);
        proximalList.Add(gameob2);
        proximalList.Add(gameob3);
        proximalList.Add(gameob4);
        proximalList.Add(gameob5);
        proximalList.Add(gameob6);
        proximalList.Add(gameob7);
        proximalList.Add(gameob8);
        proximalList.Add(gameob9);
        proximalList.Add(gameob10);
        proximalList.Add(gameob11);
        proximalList.Add(gameob12);
        proximalList.Add(gameob13);
        proximalList.Add(gameob14);
        proximalList.Add(gameob15);
        proximalList.Add(gameob16);
        proximalList.Add(gameob17);
        proximalList.Add(gameob18);

        //adds to the distal list the distal GameObjects
```

FIGURA A.25: *KinematicManager*

```
        distalList.Add(gameob19);
        distalList.Add(gameob20);
        distalList.Add(gameob21);
        distalList.Add(gameob22);
        distalList.Add(gameob23);
        distalList.Add(gameob24);
        distalList.Add(gameob25);
        distalList.Add(gameob26);
        distalList.Add(gameob27);
        distalList.Add(gameob28);
        distalList.Add(gameob29);
        distalList.Add(gameob30);

        //sets initially a hinge angle for the elements of each list
       setAngle(-2, 2, proximalList);
       setAngle(-10, 15, distalList);


    }



    private void FixedUpdate()
    {
        // if one wants to make any combination of movements
        if (((Input.GetKey(forward) || Input.GetKey(backwards)) &&
(Input.GetKey(moveUp) || Input.GetKey(moveDown)) ||
(Input.GetKey(rotateLeft) || Input.GetKey(rotateRight))) ||
(!(Input.GetKey(forward) || Input.GetKey(backwards)) &&
(Input.GetKey(moveUp) || Input.GetKey(moveDown)) &&
!(Input.GetKey(rotateLeft) || Input.GetKey(rotateRight))))
        {
            //allow the GameObjects to move
            canMove();

            //make the elements of the proximal and list non kinematic to
allow them to move
            KinematicOff(proximalList);
            KinematicOff(distalList);
            rb30.isKinematic = false;

           //set the distal minimum and maximum angles to -10 and 15 and
the proximal angles to -2 and 2
            setAngle(-2, 2, proximalList);
            setAngle(-10, 15, distalList);


        }

        // if one wants only wants to move forwards or backwards

        else if ((Input.GetKey(forward) || Input.GetKey(backwards)) &&
!(Input.GetKey(moveUp) || Input.GetKey(moveDown)) &&
!(Input.GetKey(rotateLeft) || Input.GetKey(rotateRight)))
        {
            //allow the GameObjects to move
            canMove();
```

FIGURA A.26: *KinematicManager*

```
            //make the elements of the proximal and list non kinematic to
allow them to move
            KinematicOff(proximalList);
            KinematicOff(distalList);
            rb30.isKinematic = false;

            //set the distal minimum and maximum angles to -10 and 15 and
the proximal angles to -0 and 0 for the proximal part of the
            // bronchoscope not to be deformed
            setAngle(0, 0, proximalList);
            setAngle(-10, 15, distalList);

        }

        //if the bronchoscope is left still
        else
        {
            //prevent the GameObjects from moving
            cantMove();
            //make the elements from proximal and distal lists kinematic
while leaving the 30th RigidBody non kinematic
            KinematicOn(proximalList);
            KinematicOn(distalList);
            rb30.isKinematic = false;


        }
    }


    //function to inmobilize the bronchoscope
    public void cantMove()
    {


        Rigidbody joint;

        foreach (GameObject go in distalList)
        {
            joint = go.GetComponent<Rigidbody>();
            joint.GetComponent<MovementForwardTop>().enabled = false;
            joint.freezeRotation=true;
        }


        foreach (GameObject go in proximalList)
        {
            joint = go.GetComponent<Rigidbody>();
            joint.GetComponent<MovementForwardTop>().enabled = false;
            joint.freezeRotation = true;
        }



    }

    //function to allow movement of the bronchoscope
    public void canMove()
    {
```

FIGURA A.27: *KinematicManager*

```
        Rigidbody joint;
        foreach (GameObject go in distalList)
        {
            joint = go.GetComponent<Rigidbody>();
            joint.GetComponent<MovementForwardTop>().enabled = true;
            joint.freezeRotation = false;
        }

        foreach (GameObject go in proximalList)
        {
            joint = go.GetComponent<Rigidbody>();
            joint.GetComponent<MovementForwardTop>().enabled = true;
            joint.freezeRotation = false;
        }

    }


    //function to make all of the gameobjects in a list kinematic
    public void KinematicOn(List<GameObject> List)
    {
        Rigidbody joint;
        foreach (GameObject go in List)
        {
            joint = go.GetComponent<Rigidbody>();
            joint.isKinematic = true;
        }

    }

    //function to make all of the gameobjects in a list kinematic off

    public void KinematicOff(List<GameObject> List)
    {
        Rigidbody joint;
        foreach (GameObject go in List)
        {
            joint = go.GetComponent<Rigidbody>();
            joint.isKinematic = false;
        }

    }

    //function to change the angles of  the gameobjects in a list

    public void setAngle(float minLimit, float maxLimit, List<GameObject>
List)
    {
        HingeJoint joint;

        foreach (GameObject go in List)
        {
            joint = go.GetComponent<HingeJoint>();
            JointLimits limits = joint.limits;
            limits.min = minLimit;
            limits.max = maxLimit;
            joint.limits = limits;
            joint.useLimits = true;

        }
```

FIGURA A.28: *KinematicManager*

```
using UnityEngine;
using System.Collections;

public class ExtendedFlycam : MonoBehaviour
{

    //This script controls the movement of the camera in the Exploration
    public bool dragOnClick = true;
    public float cameraSensitivity = 90;
    public float climbSpeed = 4;
    public float normalMoveSpeed = 10;
    public float fastMoveFactor = 3;
    public KeyCode dragBtn = KeyCode.Mouse1;

    public float rotationX = 0.0f;
    public float rotationY = 0.0f;


    private bool drag = false;
    void Start()
    {

        Cursor.visible = true;
        this.rotationX = transform.localRotation.eulerAngles.y;

    }

    void Update()
    {
        if (dragOnClick)
        {
            if (Input.GetKeyDown(this.dragBtn))
                this.drag = true;

            if (Input.GetKeyUp(this.dragBtn))
                this.drag = false;
        }
        else
            drag = true;

        // Screen.lockCursor =  this.drag;
        //Cursor.visible = this.drag;
        //   if(this.d)


        if (this.drag)
        {
            rotationX += Input.GetAxis("Mouse X") * cameraSensitivity *
Time.deltaTime;
            rotationY += Input.GetAxis("Mouse Y") * cameraSensitivity *
Time.deltaTime;
            rotationY = Mathf.Clamp(rotationY, -90, 90);

            transform.localRotation = Quaternion.AngleAxis(rotationX,
Vector3.up);
            transform.localRotation *= Quaternion.AngleAxis(rotationY,
Vector3.left);

            //Cursor.lockState = CursorLockMode.Locked;
            Cursor.visible = false;
```

FIGURA A.29: *ExtendedFlyCam*

```
        }
        else
        {
            Cursor.lockState = CursorLockMode.None;
            Cursor.visible = true;
        }

        if (Input.GetKey(KeyCode.LeftShift) ||
Input.GetKey(KeyCode.RightShift))
        {
            if (Input.GetKey(KeyCode.I)) { transform.position +=
transform.forward * normalMoveSpeed * fastMoveFactor * Time.deltaTime; }
            if (Input.GetKey(KeyCode.O)) { transform.position -=
transform.forward * normalMoveSpeed * fastMoveFactor * Time.deltaTime; }
            if (Input.GetKey(KeyCode.D)) { transform.position +=
transform.right * normalMoveSpeed   * fastMoveFactor * Time.deltaTime; }
            if (Input.GetKey(KeyCode.A)) { transform.position -=
transform.right * normalMoveSpeed   * fastMoveFactor * Time.deltaTime; }
            if (Input.GetKey(KeyCode.W)) { transform.position +=
transform.up * climbSpeed * fastMoveFactor * Time.deltaTime; }
            if (Input.GetKey(KeyCode.S)) { transform.position -=
transform.up * climbSpeed * fastMoveFactor * Time.deltaTime; }
        }
        else
        {
            if (Input.GetKey(KeyCode.I)) { transform.position +=
transform.forward * normalMoveSpeed * Time.deltaTime; }
            if (Input.GetKey(KeyCode.O)) { transform.position -=
transform.forward * normalMoveSpeed * Time.deltaTime; }
            if (Input.GetKey(KeyCode.D)) { transform.position +=
transform.right * normalMoveSpeed * Time.deltaTime; }
            if (Input.GetKey(KeyCode.A)) { transform.position -=
transform.right * normalMoveSpeed * Time.deltaTime;}
            if (Input.GetKey(KeyCode.W)) { transform.position +=
transform.up * climbSpeed * Time.deltaTime; }
            if (Input.GetKey(KeyCode.S)) { transform.position -=
transform.up * climbSpeed * Time.deltaTime; }

        }

        if (Input.GetKeyDown(KeyCode.End))
        {
         //   Screen.lockCursor = (Screen.lockCursor == false) ? true :
false;
        }
    }
}
```

FIGURA A.30: *ExtendedFlyCam*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Limitation : MonoBehaviour
{
    //This script limits the movement of the bronchoscope backwards
    //The respiratory tract has a transparent cylinder at its opening
    //If the joint which contains this script (the one furthest the
distal list is touching that cylinder
    //The bronchoscope is unabled to continue moving backwards so that is
does not go very far away from the bronchi
    //and makes the game more realistic

    Rigidbody rb;

   //flag that states wether the bronchoscope will be able to move
backwards or not
   //when false, the bronchoscope is not allowed to move backwards, when
true, it is allowed
    public static bool test;


    // Use this for initialization
    void Start()
    {

        rb = GetComponent<Rigidbody>();


    }

    // Update is called once per frame
    void Update()
    {

    }


    //when the object that has this script enters the area of another
object
    private void OnTriggerEnter(Collider other)
    {
        //if the other object has a tag named "umbral", the flag test is
set false
        //therefore when the bronchoscope touches the cylinder at the
opening of the respiratory tract,
        //it cannot move backwards any further
        if (other.gameObject.CompareTag("umbral"))
        {
            test = false;
        }

        //if the other object has no tag named "umbral", the flag test is
set true
        //therefore when the bronchoscope is touching any other object
than the cylinder at the opening
        //of the respiratory tract, it is allowed to move backwards
        if (other.gameObject.CompareTag("umbral")==false)
        {
```

FIGURA A.31: *Limitation*

```
                test = true;
            }
        }

    }
```

FIGURA A.32: *Limitation*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MovementDistal : MonoBehaviour {

    //This script controls the movement of the portion part of the
bronchoscope up and down

    //variables to assign the keys which will move the endoscope up an
down
    public KeyCode moveUp;
    public KeyCode moveDown;


    //this is the float that stores the speed of the movement
    public float speed;

    //public rigidbodies for the Rigid to be attached to. They are the
shackles of the bronchoscope.
    public Rigidbody rb;
    public Rigidbody rb1;
    public Rigidbody rb2;
    public Rigidbody rb3;
    public Rigidbody rb4;
    public Rigidbody rb5;
    public Rigidbody rb6;
    public Rigidbody rb7;
    public Rigidbody rb8;
    public Rigidbody rb9;
    public Rigidbody rb10;
    public Rigidbody rb11;
    public Rigidbody rb12;
    public Rigidbody rb13;
    public Rigidbody rb14;
    public Rigidbody rb15;
    public Rigidbody rb16;
    public Rigidbody rb17;
    public Rigidbody rb18;
    public Rigidbody rb19;


   //public HingeJoints fot the joints between shackles to be attached
    public HingeJoint hj19;
    public HingeJoint hj20;
    public HingeJoint hj21;
    public HingeJoint hj22;
    public HingeJoint hj23;
    public HingeJoint hj24;
    public HingeJoint hj25;
    public HingeJoint hj26;
    public HingeJoint hj27;
    public HingeJoint hj28;
    public HingeJoint hj29;
    public HingeJoint hj30;
```

FIGURA A.33: *MovementDistal*

```
   // flag that turns true in case the bronchoscope is moving its distal
segment and false if it's not
    public bool flagDistal;

    //maximum and minimum limits for the movement of the distal segment
    public float limitMin;
    public float limitMax;


      void Start () {
         //Fetch the Rigidbody component from the GameObject
         rb = GetComponent<Rigidbody>();
      }


      void FixedUpdate () {
         //sets direction of movement in (0,0,0) when no key is pressed in
a fixed rate
         Vector3 directionOfMovement = Vector3.zero;

         //if the key pressed is moveUp
         if (Input.GetKey(moveUp))
         {
         //the vector direction of movement turns to the Y-axis
coordinates
             directionOfMovement = transform.up;
       //the speed turns to 2
             speed = 2;
         //flagDistal turns true because there is intention of moving the
distal segment
             flagDistal = true;
       //Rigidbody's velocity is the speed (2) pointing to the Y-axis
direction
             rb.velocity = (directionOfMovement * speed);
         }

         //if the key pressed is moveDown
         else if (Input.GetKey(moveDown))
         {
             //the vector direction of movement turns to the -(Y-axis)
coordinates
             directionOfMovement = -transform.up;
             //the speed turns to 2
             speed = 2;
             //flagDistal turns true because there is intention of moving
the distal segment
             flagDistal = true;
             //Rigidbody's velocity is the speed (2) pointing to the -(Y-
axis) direction
             rb.velocity = (directionOfMovement * speed);
             }

         else
         {
             //flagDistal turns true because there is no intention of
moving the distal segment
             flagDistal = false;

         }
```

FIGURA A.34: *MovementDistal*

```
     }

    public void KinematicAtivator(bool flagDistal)
    {
        //if there is intention of moving the distal segment
        if (flagDistal == true)
        {
            //all shackles that belong to the proximal segment are
kinematic
            rb1.isKinematic = true;
            rb2.isKinematic = true;
            rb3.isKinematic = true;
            rb4.isKinematic = true;
            rb5.isKinematic = true;
            rb6.isKinematic = true;
            rb7.isKinematic = true;
            rb8.isKinematic = true;
            rb9.isKinematic = true;
            rb10.isKinematic = true;
            rb11.isKinematic = true;
            rb12.isKinematic = true;
            rb13.isKinematic = true;
            rb14.isKinematic = true;
            rb15.isKinematic = true;
            rb16.isKinematic = true;
            rb17.isKinematic = true;
            rb18.isKinematic = true;
            rb19.isKinematic = true;


            //the angular limits of the distal joints are set in -10 and
15 degrees
            limitMin = -10;
            limitMax = 15;
            SetLimit(hj19);
            SetLimit(hj20);
            SetLimit(hj21);
            SetLimit(hj22);
            SetLimit(hj23);
            SetLimit(hj24);
            SetLimit(hj25);
            SetLimit(hj26);
            SetLimit(hj27);
            SetLimit(hj28);
            SetLimit(hj29);
            SetLimit(hj30);

        }
        else
        {
            //all shackles that belong to the proximal segment are not
kinematic
            rb1.isKinematic = false;
            rb2.isKinematic = false;
            rb3.isKinematic = false;
            rb4.isKinematic = false;
            rb5.isKinematic = false;
            rb6.isKinematic = false;
```

FIGURA A.35: *MovementDistal*

```
            rb7.isKinematic = false;
            rb8.isKinematic = false;
            rb9.isKinematic = false;
            rb10.isKinematic = false;
            rb11.isKinematic = false;
            rb12.isKinematic = false;
            rb13.isKinematic = false;
            rb14.isKinematic = false;
            rb15.isKinematic = false;
            rb16.isKinematic = false;
            rb17.isKinematic = false;
            rb18.isKinematic = false;
            rb19.isKinematic = false;

            //the angular limits of the distal joints are set in 0
degrees
            limitMin = 0;
            limitMax = 0;
            SetLimit(hj19);
            SetLimit(hj20);
            SetLimit(hj21);
            SetLimit(hj22);
            SetLimit(hj23);
            SetLimit(hj24);
            SetLimit(hj25);
            SetLimit(hj26);
            SetLimit(hj27);
            SetLimit(hj28);
            SetLimit(hj29);
            SetLimit(hj30);


        }


}
    // this function sets the upper and lower angular limits of a joint
       private void SetLimit(HingeJoint junta)
    {
        // the variable limit stores junta's upper and lower limits
        JointLimits limit = junta.limits;
        // junta's lower limit is set in limitMin value
        limit.min = limitMin;
        // junta's upper limit is set in limitMax value
        limit.max = limitMax;
        // junta's limits are set to limit
        junta.limits = limit;
        // the parameter useLimits of the joint junta is set true, which
enables the limit to be set
        junta.useLimits = true;

    }
}
```

FIGURA A.36: *MovementDistal*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MovementForwardTop : MonoBehaviour {

    //variables to assign the keys which will move the endoscope forward
and backwards
    KeyCode forward;
    KeyCode backward;

    //float which determines the thrust of the movement
    public float thrust;

    //Attach this script to the GameObject whose movement will be
controlled
    public Rigidbody rb;


      void Start () {

        //Fetch the Rigidbody component from the GameObject
        rb = GetComponent<Rigidbody>();

        //set the keys forward and backward
        forward = KeyCode.Space;
        backward = KeyCode.B;

    }

      void Update () {

        //if the key pressed is the one assigned to forward variable
        if (Input.GetKey(forward))
        {
            //turn the variable thrust into -2
            thrust = -2;

        }

        //if the key pressed is the one assigned to backward variable and
junta30 is colliding with the threshold GameObject (Limitation.test =
true)
        else if (Input.GetKey(backward) && Limitation.test == true)
//Limitation.backAllowed == true)
        {
            //turn the variable thrust into 2
            thrust = 2;

        }

        //if none of these keys is pressed
        else

        {   //turn the variable thrust into 2
            thrust = 0;
        }

      // the velocity of the rigidbody component of the GameObject has a
speed of "thust" and directs forward
```

FIGURA A.37: *MovementForwardTop*

```
            rb.velocity = (Vector3.forward * thrust);
        }
    }
```

FIGURA A.38: *MovementForwardTop*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class nipperCombined : MonoBehaviour
{

    //This script controls the movement of the nipper

    public TextMeshPro textmeshPro;

    //the nipper is formed by the tube (which is scaled) and 2 tips.
    //the following are the game objects of every tip and the tube that
form the whole nipper
public GameObject tip1;
    public GameObject tip2;
    public GameObject tubo;

    //gameobject of transparent sphere which will detect the collision
    public GameObject nipperCollider;

    //keycodes for inputs: move nipper in, nipper out and bite
    public KeyCode nipperout;
    public KeyCode nipperin;
    public KeyCode bite;


    // The flag flag2 obtains the flag flag from the script collisionTip
    //which is false when a collision is entered and stayed, and is false
when a collision is exited
    public bool flag2;

    //The flag vertexAttached2 obtains the flag vertexAttached from the
script collisionTip
    //This flag is true when the collider has a vertex attached and is
false when it has no vertex attached
    public bool vertexAttached2;

    //vector3 to set anchors in the joints of nipper tips
    //an anchor is used to describe a position and orientation relative to
the world space and one can attach one
    //or more objects to it. The anchor and objects attached to it will
stay where they are placed in the world.

    public Vector3 anchorTip1;
    public Vector3 anchorTip2;
    public Vector3 anchorTipNipperCollider;

    //Configurable joints of the nipper and the tube, the tips and the
nipper, and the nipper with its collider
    private ConfigurableJoint joint;
    private ConfigurableJoint jointTip1;
    private ConfigurableJoint jointTip2;
```

FIGURA A.39: *nipperCombined*

```
    private ConfigurableJoint nipperColliderJoint;

    //nipper max length extension
    private float nipperLength = 0.0f;
    private float unitLength;

    //vectors for local contactpoint and its normal
    private Vector3 contactPointLocal;
    private Vector3 contactNormal;

    //vector for anchor of the tube
    private Vector3 tuboConnectedAnchor;


    void Start()
    {

    //the flag flag2 is set true
        flag2 = true;

        //the configurablejoints are obtained
        jointTip1 = tip1.GetComponent<ConfigurableJoint>();
        jointTip2 = tip2.GetComponent<ConfigurableJoint>();
        joint = tubo.GetComponent<ConfigurableJoint>();
        nipperColliderJoint =
nipperCollider.GetComponent<ConfigurableJoint>();

        //the keycodes are initialized
        nipperout = KeyCode.O;
        nipperin = KeyCode.I;
        bite = KeyCode.K;

        //the function for anchors to be autoconfigured is set false
        jointTip2.autoConfigureConnectedAnchor = false;
        jointTip1.autoConfigureConnectedAnchor = false;
        joint.autoConfigureConnectedAnchor = false;
        nipperColliderJoint.autoConfigureConnectedAnchor = false;

        //the anchors are set in zero
        jointTip1.connectedAnchor = new Vector3(0.0f, 0.0f, 0.0f);
        jointTip2.connectedAnchor = new Vector3(0.0f, 0.0f, 0.0f);
        nipperColliderJoint.connectedAnchor = new Vector3(0.0f, 0.0f,
0.0f);

        //the anchors are initialized
        tuboConnectedAnchor = joint.connectedAnchor;
        anchorTip1 = jointTip1.connectedAnchor;
        anchorTip2 = jointTip2.connectedAnchor;
        anchorTipNipperCollider = nipperColliderJoint.connectedAnchor;


        unitLength = 0.02F;
```

FIGURA A.40: *nipperCombined*

```
    }

    // Update is called once per frame
    void Update()
    {

       // The flags vertexAttached2 and flag2 are obtained from the script
collisionTip
        vertexAttached2 = collisionTip.vertexAttached;
        flag2 = collisionTip.Instance.flag;

        //The if is entered if the keycode to take the nipper out and the
the nipper is not colliding
        if (Input.GetKey(KeyCode.O)  && (flag2 == true))
        {
            //the nipper tube is scaled
            tubo.transform.localScale += new Vector3(0, unitLength, 0);
            tubo.transform.localPosition += new Vector3(0, 0, -
unitLength*tubo.transform.localScale.y);
            nipperLength = nipperLength + unitLength;

            //the anchors of the joints and tube are updated
            joint.connectedAnchor = tuboConnectedAnchor;
            jointTip1.connectedAnchor = new Vector3(0,
tubo.transform.localPosition.y - (50 * unitLength), 0);
            jointTip2.connectedAnchor = new Vector3(0,
tubo.transform.localPosition.y - (50 * unitLength), 0);
            nipperColliderJoint.connectedAnchor = new Vector3(0,
tubo.transform.localPosition.y - (50 * unitLength), 0);
            anchorTip1 = jointTip1.connectedAnchor;
            anchorTip2 = jointTip2.connectedAnchor;
            anchorTipNipperCollider = nipperColliderJoint.connectedAnchor;

        }

        //The if is entered if the keycode to move the nipper inside the
bronchoscope and the nipperlength is positive
else if (Input.GetKey(KeyCode.I) && (tubo.transform.localScale - new
Vector3(0, unitLength, 0)).y > 0)
        {
            //the nipper tube is scaled
            tubo.transform.localScale -= new Vector3(0, unitLength, 0);
            tubo.transform.position += new Vector3(0, 0, unitLength *
tubo.transform.localScale.y);
            nipperLength = nipperLength - unitLength;

            //the anchors of the joints and tube are updated
            joint.connectedAnchor = new Vector3(0,
tubo.transform.position.y, 0);
            jointTip1.connectedAnchor = new Vector3(0,
tubo.transform.localPosition.y - (40 * unitLength), 0);
            jointTip2.connectedAnchor = new Vector3(0,
tubo.transform.localPosition.y - (40 * unitLength), 0);
```

FIGURA A.41: *nipperCombined*

```
            nipperColliderJoint.connectedAnchor = new Vector3(0,
tubo.transform.localPosition.y - (40 * unitLength), 0);
            anchorTip1 = jointTip1.connectedAnchor;
            anchorTip2 = jointTip2.connectedAnchor;
            anchorTipNipperCollider = nipperColliderJoint.connectedAnchor;
            joint.connectedAnchor = tuboConnectedAnchor;

        }

        //when the K keycode to bite is pressed
        if (Input.GetKeyDown(KeyCode.K))
        {
            //the Z angular limit from the tips is set to 60
            SoftJointLimit angularLimit1 = jointTip1.angularZLimit;
            angularLimit1.limit = 60;
            jointTip1.angularZLimit = angularLimit1;
            SoftJointLimit angularLimit2 = jointTip2.angularZLimit;
            angularLimit2.limit = 60;
            jointTip2.angularZLimit = angularLimit2;
        }

       //when the K keycode to stop biting is up
        else if (Input.GetKeyUp(KeyCode.K))
        {
            //the Z angular limit from both tips is set to 0
            SoftJointLimit angularLimit1 = jointTip1.angularZLimit;
            angularLimit1.limit = 0;
            jointTip1.angularZLimit = angularLimit1;
            SoftJointLimit angularLimit2 = jointTip2.angularZLimit;
            angularLimit2.limit = 0;
            jointTip2.angularZLimit = angularLimit2;
        }
    }
}
```

FIGURA A.42: *nipperCombined*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Rotate : MonoBehaviour {

    //This script controls the rotation of the bronchoscope

    //variables to assign the keys which will move the bronchoscope left
and right
    public KeyCode rotateLeft;
    public KeyCode rotateRight;

    //float in which the turn speed of the bronchoscope is saved
    public float turnSpeed;


     // Use this for initialization
     void Start () {

      //assigns the corresponding keys to the variables (D and A)
        rotateLeft = KeyCode.D;
        rotateRight = KeyCode.A;


     }

     // Update is called once per frame
     void Update () {

        // if the key pressed equals rotateLeft (or D)
        if (Input.GetKey(rotateLeft))
        {
            // the variable turnSpeed changes to 100
            turnSpeed = 100;
            //the rotation of the GameObject to wich this script is
attached turns locally around its "-forward" axis
            //by a rate of turnSpeed*the amount of time it took to render
the last frame
            transform.Rotate(-Vector3.forward, turnSpeed *
Time.deltaTime, Space.Self);

        }


        // if the key pressed equals rotateLeft (or A)
        else if (Input.GetKey(rotateRight))
        {
            // the variable turnSpeed changes to 100
            turnSpeed = 100;
            //the rotation of the GameObject to wich this script is
attached turns locally around its "forward" axis
            //by a rate of turnSpeed*the amount of time it took to render
the last frame
            transform.Rotate(Vector3.forward, turnSpeed * Time.deltaTime,
Space.Self);


        }
```

FIGURA A.43: *Rotate*

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class ScoreManager : MonoBehaviour
{
    public static ScoreManager Instance;

    //This script controls the performance

    //variables to count the ammout of times a granuloma/tissue is bitten
    public int healthyCounter;
    public int biopsyCounter1;
    public int biopsyCounter2;
    public int biopsyCounter3;
    public int biopsyCounter;

    //UI texts to show the results of the biopsies
    public TextMeshPro biopsyCountText1;
    public TextMeshPro biopsyCountText2;
    public TextMeshPro biopsyCountText3;
    public TextMeshPro biopsyCountText;
    public TextMeshPro percentageText;
    public TextMeshPro totalScoreText;
    public TextMeshPro timerText;
    public TextMeshPro title;
    public TextMeshPro levelText;
    //UI buttons
    public GameObject endButton;
    public GameObject restartButton;
    public GameObject helpButton;
    public GameObject changeDifficultyButton;
    //list of UItexts
    public List<TextMeshPro> granulomasTexts;

    //camera for the UI
    public Camera UICamera;
    //camera to show the model from a frontal view
    public Camera frontCamera;

    //GameObject which contains all the bronchoscopy objects
    public GameObject broncoscopyObjects;

    //color for the text of the granuloma
    public Color granulomaBiopsedColor;

    //as the percentage tham can be shown is model dependant, the
multiplier allows to adjust it to reality
    public float percentageMultiplier = 3;

    // Make the list GranulomaBehavior adn the GameObject bronchiPainter
not show up in the inspector
    // but be serialized.
    [HideInInspector]
    public List<GranulomaBehavior> granulomas;
    [HideInInspector]
    public GameObject bronchiPainter;
```

FIGURA A.44: *ScoreManager*

```
    //strings to be shown in the UI
    private string biopsyString1;
    private string biopsyString2;
    private string biopsyString3;
    private string biopsyString;
    private string percentageString;
    private string totalScoreString;
    private string timerString;

    //GameObject level which contains the level and case information
which was randmized in previous scenes
    public GameObject level;
    //string which is used to save the level information from the
GameObject Level
    public string difficultyLevel;

    //ints to count the ammount of biopsies, %covered, total score and
time
    private int biopsyCount1;
    private int biopsyCount2;
    private int biopsyCount3;
    private int biopsyCount;
    private int percentageCovered; //not used
    private int totalScore; //not used
    private float theTime;

    //Awake is always called before any Start functions
    private void Awake()
    {
        //If instance already exists and it's not this:
        if (!Instance)
            //set instance to this
            Instance = this;
    }

    // Start is called before the first frame update
    void Start()
    {
        //assign to level GameObject a GameObject whose tag is "Level"
        level = GameObject.Find("Level");
        //get the variable level from the GameObject Level and set
difficultyLevel as such
        difficultyLevel = level.GetComponent<GetLevel>().level;
        //set the UI text levelText as "difficultyLevel LEVEL", e.g.
"HIGH LEVEL"
        levelText.text = difficultyLevel + " LEVEL";
        //set the counters in 0
        healthyCounter = 0;
        biopsyCounter1 = 0;
        biopsyCounter2 = 0;
        biopsyCounter3 = 0;
        //assign strings to UI texts
        biopsyString1 = biopsyCountText1.text;
        biopsyString2 = biopsyCountText2.text;
        biopsyString3 = biopsyCountText3.text;
        biopsyString = biopsyCountText.text;
        percentageString = percentageText.text;
        totalScoreString = totalScoreText.text;
        timerString = timerText.text;
```

FIGURA A.45: *ScoreManager*

```
        //set the UI texts  as "biopsystring + biopsy count" and
"totalScoreString + healthyCounter"
        biopsyCountText1.text = biopsyString1 + biopsyCount1;
        biopsyCountText2.text = biopsyString2 + biopsyCount2;
        biopsyCountText3.text = biopsyString3 + biopsyCount3;
        biopsyCountText.text = biopsyString + biopsyCount;
        totalScoreText.text = totalScoreString + healthyCounter;
        //set percentage covered and total score to 0 initially
        percentageCovered = 0; //not used
        totalScore = 0; //not used

        //set the UI components that will appear at first
        title.gameObject.SetActive(false);
        levelText.gameObject.SetActive(true);
        percentageText.gameObject.SetActive(false);
        totalScoreText.gameObject.SetActive(true);
        restartButton.SetActive(false);
        changeDifficultyButton.SetActive(false);
        timerText.gameObject.SetActive(true);
        broncoscopyObjects.SetActive(true);
        endButton.SetActive(true);
        helpButton.SetActive(true);
        foreach (TextMeshPro objText in granulomasTexts)
        objText.gameObject.SetActive(false);

        //start computing time
        StartCoroutine("ComputeTime");
    }


    //If the lesion is bitten the counter for that lesion is updated
    public void ComputeBiopsy(int granulomaID)
    {
        if (granulomaID ==1)
        {
            biopsyCountText1.text = biopsyString1 + ++biopsyCount1;
            biopsyCountText.text = biopsyString + ++biopsyCount;
            biopsyCounter1 = biopsyCounter1 + 1;
            biopsyCount = biopsyCount + 1;

        }
        if (granulomaID == 2)
        {
            biopsyCountText2.text = biopsyString2 + ++biopsyCount2;
            biopsyCountText.text = biopsyString + ++biopsyCount;
            biopsyCounter2 = biopsyCounter2 + 1;
            biopsyCount = biopsyCount + 1;

        }
        if (granulomaID == 3)
        {
            biopsyCountText3.text = biopsyString3 + ++biopsyCount3;
            biopsyCountText.text = biopsyString + ++biopsyCount;
            biopsyCounter3 = biopsyCounter3 + 1;
            biopsyCount = biopsyCount + 1;

        }

    }
```

FIGURA A.46: *ScoreManager*

```
    //If healthy tissue is bitten its counter is updated
    public void ComputeHealthy()
    {
        totalScoreText.text = totalScoreString + ++healthyCounter;

    }

    //shows the percentage painted influenced by the percentage
multiplier
    public void ShowPercentage(int percentage)
    {
        percentageCovered = (int)(percentage * percentageMultiplier);
        percentageText.text = "PERCENTAGE COVERED " + percentageCovered +
"%";
    }

    //if the simulation ends
    public void EndSimulation()
    {
        //the computing of time stops
        StopCoroutine("ComputeTime");

       //the cameras change their angle to point towards the results
screen
        frontCamera.rect = new Rect(0.3f, 0.0f, 0.7f, 1.0f);
        UICamera.rect = new Rect(0.0f, 0.0f, 0.3f, 1.0f);


        frontCamera.cullingMask = ~(1 <<
LayerMask.NameToLayer("Endoscope"));
        frontCamera.cullingMask =  (1 << LayerMask.NameToLayer("Bronchi
Painter"));



        //bronchoscopy objects are hiden
        broncoscopyObjects.SetActive(false);

        //UI result components are shown and the rest is hiden
        endButton.SetActive(false);
        restartButton.SetActive(true);
        helpButton.SetActive(false);
        changeDifficultyButton.SetActive(true);
        totalScoreText.gameObject.SetActive(true);

        //makes the bronchiPainter GameObject's parent null
        bronchiPainter.transform.parent = null;
        //starts couroutine to rotate the painted bronchi
        StartCoroutine("AnimateBronchiPainter");

        //shows total score text
        totalScoreText.text = totalScoreString + healthyCounter;



    }

    //works as a chronometer which turns time values to string
    IEnumerator ComputeTime()
    {
```

FIGURA A.47: *ScoreManager*

```
        while (true)
        {
            theTime += Time.deltaTime;
            string minutes = Mathf.Floor((theTime % 3600) /
60).ToString("00");
            string seconds = (theTime % 60).ToString("00");
            timerText.text = timerString + minutes + ":" + seconds;
            yield return null;
        }
    }

    //when called, starts rotating bronchiPainter GameObject
    IEnumerator AnimateBronchiPainter()
    {
        while (true)
        {
            bronchiPainter.transform.Rotate(Vector3.up, 20 *
Time.deltaTime, Space.Self);
            yield return null;
        }
    }

    //function to restart the simulation when called
    public void RestartSimulation()
    {

        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }

    //function which directs the player to "Set difficulty" scene
    public void RestartDifficulty()
    {
        SceneManager.LoadScene("SetDifficulty");

    }
}
```

FIGURA A.48: *ScoreManager*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SetGravity : MonoBehaviour {


    //this script sets gravity to the GameObjects which crosses the
threshold GameObject that determines the gravity action

    //Rigidbody component to be obtained from the GameObject to which
this script is attached
    Rigidbody rb;
    // this float counts the number of times that the GameObject crosses
a certain threshold
    public float counter;
    //this float is 0 if counter is even and 1 if it is odd
    public float temporal;


     // Use this for initialization
     void Start () {

        //Fetch the Rigidbody component from the GameObject
        rb = GetComponent<Rigidbody>();

        //set the counter to 0
        counter = 0;

     }


     void Update () {


    // an instance variable that will catch counter's value and give 0 if
it is even and 1 if it is odd.
    temporal = counter - 2*Mathf.Floor(counter / 2);

        //if temporal variable is 0, then the game object is outside the
airway (this is determined by how many times the thrsehold has been
crossed)
    if (temporal == 0f) //ignoramos advertencia
       {
           //the GameObject's gravity is set to false
           rb.useGravity = false;
        }
        //if temporal variable is 1, then the game object is inside the
airway (this is determined by how many times the thrsehold has been
crossed)
        if (temporal == 1)
        {
            //the GameObject's gravity is set to true
            rb.useGravity = true;
        }
    }

    // when the collider of the GameObject collides with the threshold
GameObject "umbral":
    private void OnTriggerEnter(Collider other)
```

FIGURA A.49: *SetGravity*

```
{
    if (other.gameObject.CompareTag("umbral"))
     {
        //the counter is increased by one
        counter = counter + 1;

        }
    }


}
```

FIGURA A.50: *SetGravity*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class ChangeScene : MonoBehaviour
{
   //This script allows scene management, if the function SceneChanging()
is called, the Scene is changed to sceneName

    public void SceneChanging(string sceneName)
    {
        SceneManager.LoadScene(sceneName);

    }
}
```

FIGURA A.51: *ChangeScene*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;


public class DestroyLevel : MonoBehaviour
{

    //This script destroys the variable of the level selected for when
the game is restarted

    //button from the UI corresponding to back action
    public Button back;
    //GameObject which is set not to be destroyed when changing scenes
    public GameObject level;

    // Start is called before the first frame update
    void Start()
    {
        ////Fetch the Button component from the GameObject
        back.GetComponent<Button>();
        //Fetch the GameObject whose tag is "Level"
        level = GameObject.Find("Level");

        //when the button back is clicked, calls Destroy()
     back.onClick.AddListener(() => Destroy());

    }

    //this function destroys the GameObject "Level"
    private void Destroy()
    {
        Destroy(level);
    }
}
```

FIGURA A.52: *DestroyLevel*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class GetLevel : MonoBehaviour

    //This script gets the level chosen depending on the button selected

{
    //string that stores the information of the button clicked
    public string level;
    //possible buttons to be clicked
    public Button lowBut, medBut, highBut;
    //variable for caseIndex to be saved
    public int caseIndex;



    // Start is called before the first frame update
    void Start()
    {
        level = "none";
        //if the button clicked is the high/medium/low button, then call
ButtonClicked("high"/"medium"/"low")
        highBut.onClick.AddListener(() => ButtonClicked("high"));
        medBut.onClick.AddListener(() => ButtonClicked("medium"));
        lowBut.onClick.AddListener(() => ButtonClicked("low"));

    }

    // this function sets level string depending on the button which was
clicked
    void ButtonClicked(string buttonClicked)
    {
        level = buttonClicked;
    }

}
```

FIGURA A.55: *GetLevel*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SelectCase : MonoBehaviour
{

    //This script shows the model that is randomized between the options
according to the level of difficulty chosen

    //variable used to get GameObject containing level information
    public GameObject level;


    //variables used to attach bronchi GameObjects
    public GameObject Case1LD;
    public GameObject Case2LD;
    public GameObject Case3LD;

    public GameObject Case1MD;
    public GameObject Case2MD;
    public GameObject Case3MD;

    public GameObject Case1HD;
    public GameObject Case2HD;
    public GameObject Case3HD;

    //list of bronchi to be randomized
    public List<GameObject> cases;


    //variables to register level and case of the bronchi selected
    public string difficultyLevel;
    public int bronchiCase;

    // Start is called before the first frame update
    void Start()
    {
        //set GameObject level to the variable Level
        level = GameObject.Find("Level");

        //add cases to the list of bronchi cases
        cases.Add(Case1LD);
        cases.Add(Case2LD);
        cases.Add(Case3LD);
        cases.Add(Case1MD);
        cases.Add(Case2MD);
        cases.Add(Case3MD);
        cases.Add(Case1HD);
        cases.Add(Case2HD);
        cases.Add(Case3HD);

        //initially sets all objects in false, for them not to appear in
the scene
        cases[1].SetActive(false);
        cases[2].SetActive(false);
        cases[0].SetActive(false);
        cases[3].SetActive(false);
        cases[4].SetActive(false);
        cases[5].SetActive(false);
```

FIGURA A.56: *SelectCase*

```
        cases[7].SetActive(false);
        cases[8].SetActive(false);
        cases[6].SetActive(false);

        //get level and case information from the GameObject level
        difficultyLevel = level.GetComponent<GetLevel>().level;
        bronchiCase = level.GetComponent<GetLevel>().caseIndex;

        //set active the bronchi level and case that corresponds to the
information brought in GameObject level. Set active their corresponding
UI texts

        if (difficultyLevel == "low")
        {
            if (bronchiCase == 0)
            {
                cases[0].SetActive(true);


ScoreManager.Instance.biopsyCountText1.gameObject.SetActive(true);

ScoreManager.Instance.biopsyCountText2.gameObject.SetActive(false);

ScoreManager.Instance.biopsyCountText3.gameObject.SetActive(false);


            }
            if (bronchiCase == 1)
            {
                cases[1].SetActive(true);


ScoreManager.Instance.biopsyCountText1.gameObject.SetActive(true);

ScoreManager.Instance.biopsyCountText2.gameObject.SetActive(false);

ScoreManager.Instance.biopsyCountText3.gameObject.SetActive(false);

            }
            if (bronchiCase == 2)
            {
                cases[2].SetActive(true);


ScoreManager.Instance.biopsyCountText1.gameObject.SetActive(true);

ScoreManager.Instance.biopsyCountText2.gameObject.SetActive(false);

ScoreManager.Instance.biopsyCountText3.gameObject.SetActive(false);

            }
        }
        if (difficultyLevel == "medium")
        {
            if (bronchiCase == 0)
            {
                cases[3].SetActive(true);


ScoreManager.Instance.biopsyCountText1.gameObject.SetActive(true);
```

FIGURA A.57: *SelectCase*

```
ScoreManager.Instance.biopsyCountText2.gameObject.SetActive(true);

ScoreManager.Instance.biopsyCountText3.gameObject.SetActive(false);

            }
            if (bronchiCase == 1)
            {
                cases[4].SetActive(true);


ScoreManager.Instance.biopsyCountText1.gameObject.SetActive(true);

ScoreManager.Instance.biopsyCountText2.gameObject.SetActive(true);

ScoreManager.Instance.biopsyCountText3.gameObject.SetActive(false);

            }
            if (bronchiCase == 2)
            {
                cases[5].SetActive(true);


ScoreManager.Instance.biopsyCountText1.gameObject.SetActive(true);

ScoreManager.Instance.biopsyCountText2.gameObject.SetActive(true);

ScoreManager.Instance.biopsyCountText3.gameObject.SetActive(false);

            }
        }

        if (difficultyLevel == "high")
        {
            if (bronchiCase == 0)
            {
                cases[6].SetActive(true);


ScoreManager.Instance.biopsyCountText1.gameObject.SetActive(true);

ScoreManager.Instance.biopsyCountText2.gameObject.SetActive(true);

ScoreManager.Instance.biopsyCountText3.gameObject.SetActive(true);

            }
            if (bronchiCase == 1)
            {
                cases[7].SetActive(true);


ScoreManager.Instance.biopsyCountText1.gameObject.SetActive(true);

ScoreManager.Instance.biopsyCountText2.gameObject.SetActive(true);

ScoreManager.Instance.biopsyCountText3.gameObject.SetActive(false);

            }
            if (bronchiCase == 2)
            {
```

FIGURA A.58: *SelectCase*

```
                cases[8].SetActive(true);

ScoreManager.Instance.biopsyCountText1.gameObject.SetActive(true);
ScoreManager.Instance.biopsyCountText2.gameObject.SetActive(true);
ScoreManager.Instance.biopsyCountText3.gameObject.SetActive(true);
            }
        }


    }
}
```

FIGURA A.59: *SelectCase*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class Timer : MonoBehaviour
{

    //This script counts the time

    private bool record;
    private float currentTime;
    private float bestTime;
    private TextMeshPro textmeshPro;


    float theTime;
    public float speed = 1;
    bool playing;



    // Start is called before the first frame update
    void Start()
    {
        // Reset current time
        currentTime = 0.0f;

        // Start record
        textmeshPro = GetComponent<TextMeshPro>();
        textmeshPro.SetText("00:00");

    }

    // Update is called once per frame
    void Update()
    {

        theTime += Time.deltaTime * speed;
     //   string hours = Mathf.Floor((theTime % 216000)
/3600).ToString("00");
        string minutes = Mathf.Floor((theTime % 3600) /
60).ToString("00");
        string seconds = (theTime % 60).ToString("00");
        textmeshPro.SetText(   minutes + ":" + seconds);

    }


    void StartRecord()
    {
        record = true;
    }


}
```

FIGURA A.60: *Timer*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class SetDifficulty : MonoBehaviour {

    //this scene prevents GameObject "level" from being destroyed when
changing scenes.
    //It should be used when clicking the button corresponding to scene
transition, as it also controls that


    public GameObject level; //variable used to be assigned to the level
information containing GameObject



    private void Start()
    {
        //find GameObject named "Level"
        level = GameObject.Find("Level");
        // do not destroy it while changing scenes
        DontDestroyOnLoad(level);
    }


    // this void receives a scene name and re-directs the main camera to
that scene
    public void ChangeDifficutyScene(string sceneName){
        SceneManager.LoadScene(sceneName);

    }
}
```

FIGURA A.61: *SetDifficulty*

```
ï»¿using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class DropdownChoosing : MonoBehaviour
{
    //Attach this script to a Dropdown GameObject
    Dropdown m_Dropdown;

    //This is the index value of the Dropdown
    int m_DropdownValue;

    // public GameObjects which contains the videos of the different
planes and cases
    public GameObject sagital1;
    public GameObject sagital10;
    public GameObject sagital20;
    public GameObject sagital60;

    public GameObject frontal1;
    public GameObject frontal10;
    public GameObject frontal20;
    public GameObject frontal60;

    public GameObject trans1;
    public GameObject trans10;
    public GameObject trans20;
    public GameObject trans60;



    void Start()
    {
        //Fetch the DropDown component from the GameObject
        m_Dropdown = GetComponent<Dropdown>();

    }

    // Update is called once per frame
    void Update()
    {
        //Keep the current index of the Dropdown in a variable
        m_DropdownValue = m_Dropdown.value;

        //depending on wich Dropdown value is selected, the videos whose
SetActive is enabled:

        if (m_DropdownValue == 0) //the user has chosen sagital view
        {
            sagital1.SetActive(true);
            sagital10.SetActive(true);
            sagital20.SetActive(true);
            sagital60.SetActive(true);
```

FIGURA A.62: *DropdownChoosing*

```
            frontal1.SetActive(false);
            frontal10.SetActive(false);
            frontal20.SetActive(false);
            frontal60.SetActive(false);

            trans1.SetActive(false);
            trans10.SetActive(false);
            trans20.SetActive(false);
            trans60.SetActive(false);

        }
        if (m_DropdownValue == 1) // the user has chosen frontal view
        {
            sagital1.SetActive(false);
            sagital10.SetActive(false);
            sagital20.SetActive(false);
            sagital60.SetActive(false);

            frontal1.SetActive(true);
            frontal10.SetActive(true);
            frontal20.SetActive(true);
            frontal60.SetActive(true);

            trans1.SetActive(false);
            trans10.SetActive(false);
            trans20.SetActive(false);
            trans60.SetActive(false);
        }

        if (m_DropdownValue == 2) //the user has chosen transversal view
        {
            sagital1.SetActive(false);
            sagital10.SetActive(false);
            sagital20.SetActive(false);
            sagital60.SetActive(false);

            frontal1.SetActive(false);
            frontal10.SetActive(false);
            frontal20.SetActive(false);
            frontal60.SetActive(false);

            trans1.SetActive(true);
            trans10.SetActive(true);
            trans20.SetActive(true);
            trans60.SetActive(true);
        }

    }
}
```

FIGURA A.63: *DropdownChoosing*