Título: Diseño e implementación de un juego serio de entrenamiento quirúrgico cardíaco.

Autor: Javier Ignacio Pedreira Estabridis

Fecha:

Directora: Daniela Tost Pardell

Departamento del director: Lenguajes y Sistemas Informáticos (LSI)

Titulación: Ingeniería Informática (EI).

Centro: Facultad de Informática de Barcelona (FIB)

Contenido

1 Introducción

1.1 Motivación

Este proyecto se centrará en las patologías cardíacas, el principal motivo se debe a que las enfermedades del corazón conforman la primera causa de muerte actualmente en España y en la mayor parte de países desarrollados del primer mundo. De todas las patologías cardíacas, este proyecto se centra en las que afectan a la válvula mitral, que se encarga de regular el paso de la sangre entre la aurícula y el ventrículo izquierdo. El mecanismo de la válvula es sencillo, cuando la aurícula izquierda se llena de sangre durante el movimiento de diástole, se ejerce presión sobre la válvula, la cual se abre y permite el paso de sangre hacia el ventrículo. Cuando la sangre ha pasado y se inicia el movimiento de sístole la válvula se cierra y unas cuerdas tendinosas impiden que se abra en el sentido contrario.

Las intervenciones quirúrgicas que afectan al corazón, en general se pueden clasificar en dos grandes grupos, las abiertas, que son con el corazón al descubierto y que suelen requerir un postoperatorio más prolongado, y las realizadas por catéter, que son menos invasivas.

Con la edad, las fibras del corazón pierden su elasticidad y pueden producir entre otras cosas que las cuerdas que sujetan los velos de la válvula se rompan o sufran una elongación con la consecuencia de permitir que los velos de la valva permitan el paso de sangre en el sentido contrario conocido como regurgitación de válvula mitral. Entonces se requiere la intervención de un cirujano quien se encarga de reparar las cuerdas de la válvula.

Los cirujanos, en general, durante su formación tienen acceso a gran cantidad de material visual en manuales de anatomía y medicina que tiene por objetivo formarles en lo que será su profesión. Sin embargo, deben acumular una gran cantidad de horas de prácticas con cuerpos y muestras reales antes de poder tener en sus manos vidas a las cuales deberán sanar. Es por eso que este proyecto está pensado para contribuir con el desarrollo de herramientas de simulación que permita a estos profesionales familiarizarse y practicar en un entorno de pruebas, con el objetivo de reducir el tiempo que necesitan para adquirir la experiencia necesaria. Por este motivo se trabaja con la idea del desarrollo de un juego serio, un concepto en auge actualmente, ya que permite a muchos profesionales adquirir experiencia y desarrollar estrategias en un ambiente de “juego” que les permitirá más adelante desenvolverse con mayor soltura en su entorno de trabajo, sobre todo cuando se trata de algo delicado, como es el caso de un cirujano cardiólogo.

1.2 Objetivos

Este apartado describe los objetivos del proyecto, clasificados en dos grandes grupos. En primer lugar, se habla de los objetivos generales del proyecto, que a simple vista, son los *milestones* que queremos conseguir para dar por finalizado este proyecto. A continuación tenemos los objetivos específicos, que dentro del contexto de los objetivos generales, tenemos un desglose de objetivos específicos que han de concretarse los objetivos primarios del proyecto. Este desglose se hace de esta manera, debido a que por lo general para conseguir un objetivo principal, primero fue necesario conseguir una serie de condiciones previas.

1.2.1 Generales

- **Construir un prototipo interactivo de un modelo de válvula mitral:** Se utilizarán técnicas de modelos deformables para conseguir la sensación de estar manipulando un objeto blando.
- **Obtener un modelo visualmente realista de una válvula mitral.**
- **Conseguir un escenario de simulación donde se pueda observar la válvula mitral.**
- **Establecer una relación entre el modelo interactivo y el modelo de simulación:** Este escenario debe permitir estudiar diferentes configuraciones de la válvula mitral.

1.2.2 Específicos

Para poder completar los objetivos principales creamos una serie de objetivos concretos a conseguir.

- **Conocer y entender cómo funciona la válvula mitral:** Se necesita comprender que partes la conforman y que factores influyen en su funcionalidad. Además se ha de estudiar las distintas afecciones cardíacas que pueden afectar a la misma para poder definir el alcance del proyecto.
- **Diseñar un modelo simplificado que represente la funcionalidad básica de una válvula mitral**: Se debe identificar todas sus partes y debe cumplir con su función de abrir, cerrar e impedir físicamente el paso de objetos en uno u otro sentido.

- **Analizar que herramientas existen en el mercado que nos pueda permitir construir la aplicación que estamos buscando:** Existe un amplio abanico de posibilidades ya que la industria del videojuego está muy extendida hoy en día. Sin embargo se han de identificar las herramientas adecuadas para poder construir una simulación de un objeto blando como lo es un corazón o una válvula cardíaca.
- **Diseñar un prototipo simplificado de una válvula mitral:** Este modelo no es visualmente realista pero sí, físicamente lo cual permitirá estudiar su comportamiento.
- **Se introducirá el factor interactivo que tendrá por objetivo permitir la deformación física del modelo y la manipulación de las cuerdas tendinosas.**
- **Conseguir un modelo de visualización interactivo que permita identificar los elementos de escenario con sus contrapartidas reales.**

1.3 Alcance y estructura de la memoria

La memoria aquí presente describe el trabajo llevado a cabo para desarrollar una idea de producto con la intención de sentar las bases para el desarrollo de un juego serio basado en la válvula mitral.

En primer lugar este documento describirá qué es una válvula mitral, qué partes la componen y cuál es su funcionalidad dentro del corazón.

En segundo lugar se detallarán las pruebas realizadas a la hora de escoger una herramienta tecnológica adecuada ya que el desarrollo del modelo de una válvula requiere la utilización de tecnologías orientadas al desarrollo de videojuegos de una forma que no se suele utilizar con mucha frecuencia debido al coste computacional que requiere y a otras limitaciones tecnológicas.

En tercer lugar, se describirá la aplicación conceptual a la que se ha llegado finalmente y se justificará su diseño e implementación.

A continuación se expondrán las conclusiones extraídas del trabajo realizado y finalmente se explicará en qué punto de desarrollo se encuentra el proyecto y que opciones de ampliación existen, que pueden servir como base para futuros proyectos que se basen en este.

Finalmente se encuentran los anexos, que contienen la información técnica del proyecto así como el glosario de acrónimos.

2. Antecedentes

El apartado de antecedentes explica el contexto sobre el que se basa este proyecto final de carrera. En las siguientes líneas se describirá el trabajo previo que sirvió para documentar la anatomía de una válvula mitral, sus partes, su funcionalidad y finalmente que clase de patologías le afectan así como que clase de intervenciones se pueden aplicar sobre la misma.

2.1 La válvula mitral

2.1.1 Funcionalidad

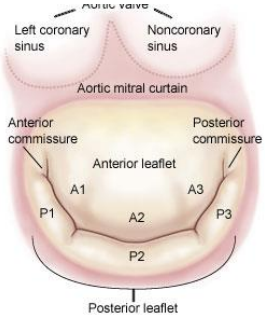
Figura 1: Esquema simple de corazón

La válvula mitral es un componente del corazón que se encuentra alojado entre el ventrículo izquierdo y la aurícula izquierda. También denominada válvula bicúspide, está formada por 2 valvas o velos que se presionan una contra otra para formar lo que se conoce como válvula de regulación de paso de fluidos. Su diseño está pensado para permitir el paso de sangre en un único sentido, que es entre la aurícula izquierda y el ventrículo izquierdo.

Esta válvula interviene dentro del ciclo cardíaco cuando se produce una diferencia de presión entre la aurícula y el ventrículo izquierdo. Mientras la presión auricular sea mayor que la ventricular, la sangre entrará en el ventrículo hasta que la presión de este provoca que la sangre intente regresar cerrando la válvula. En este punto las cuerdas tendinosas evitan que los velos cedan impidiendo que la sangre vuelva a la aurícula.

2.2 Los velos

Figura 2: Válvula mitral, vista vertical



Los dos velos que forman la válvula mitral se conocen como velo anterior y velo posterior. El velo anterior tiene forma semicircular y ocupa la mayor parte de la superficie de la válvula. El tejido del velo anterior está conectado con el tejido de la válvula aórtica que está ubicada también en el ventrículo izquierdo, como se puede ver en la

figura 3, de forma adyacente a la válvula mitral. El velo anterior carece de indentaciones (comisuras), lo cual debe tenerse en cuenta en el caso de un prolapso de la válvula. Este velo está separado en 3 segmentos, identificados en la figura como segmento anterior [A1], segmento medio [A2] y segmento posterior [A3]. ([1])

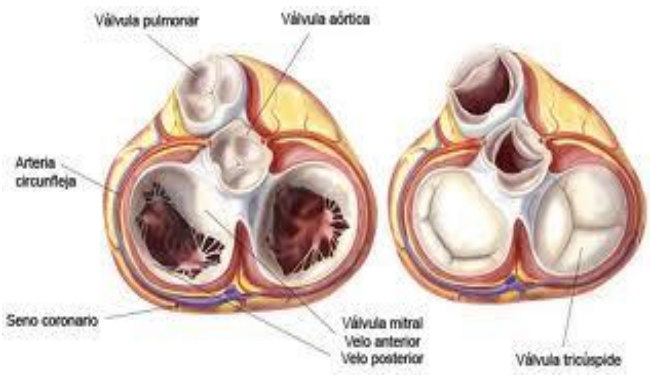


Figura 3: Válvulas ventrículo izquierdo

El velo posterior de la válvula tiene en cambio forma cuadrangular y tiene una superficie similar al velo anterior.A diferencia del velo anterior, este posee 2 indentaciones muy claras que dividen el velo en 3 partes. En la figura 2, las partes se identifican como segmento anterior [P1], segmento medio [P2] y segmento posterior o lateral [P3].

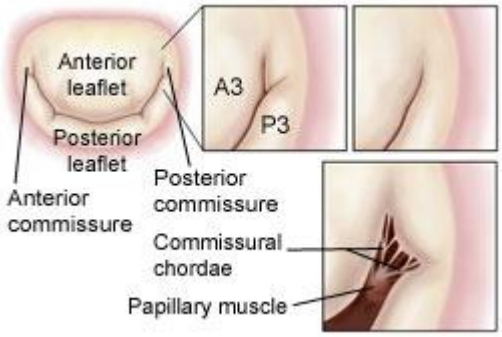


Figura 4: Comisuras

El velo anterior y el velo posterior se unen en las dos comisuras de la válvula dándole este característico aspecto de sonrisa. Los segmentos A1 y P1 y A3 y P3 se juntan respectivamente y forman esta comisura de las cuales nacen las cuerdas tendinosas llamadas comisurales. Esta sección debe tenerse en cuenta a la hora de tratar el prolapso y la regurgitación de sangre. ([1])

2.3 Zona de coaptación

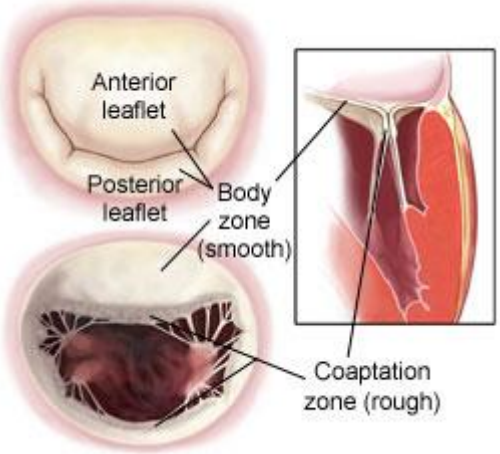


Figura 5: Zona de copatación

La zona de Coaptación es la parte que bien podría definir la funcionalidad misma de la válvula ya que es la parte de los velos que se pegan entre sí para provocar el cierre de la válvula y evitar que la sangre vuelva a la aurícula. De esta área es donde parte de las cuerdas tendinosas tiene su raíz a lo largo del perímetro que forma el orificio de la válvula. ([1])

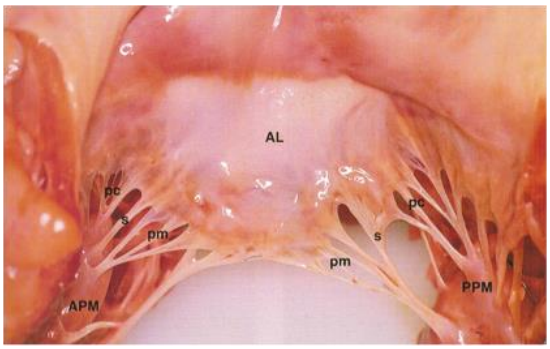


Figura 6: Vista de velo anterior

En esta imagen podemos apreciar el velo anterior [AL] y algunas cuerdas marginales sujetas a los músculos papilares anterior y posterior [APM, PPM]. Se puede apreciar como las cuerdas se reúnen para formar manojos que se arraciman en la cúspide de los músculos. ([1])

2.4 Anillo Mitral

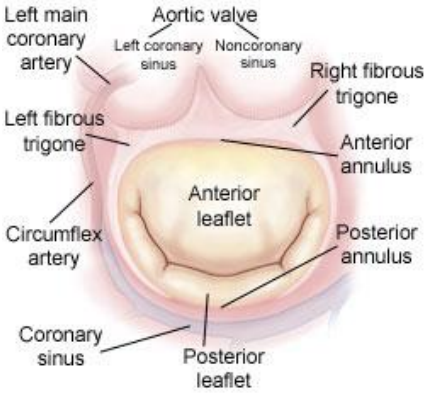


Figura 7: Estructuras vecinas

El anillo mitral constituye la unión entre el ventrículo y la aurícula izquierda y sirve como punto de inserción para el tejido de los velos. El anillo está dividido en dos partes. El anillo anterior, que se corresponden con el tejido del velo anterior y el anillo posterior, que se corresponde con el tejido del velo posterior.

La parte anterior del anillo está mucho más desarrollada que la parte posterior y está anclada a un trozo de tejido que por su figura se denomina trígono fibroso, que es una porción de tejido que separa la válvula mitral de la válvula tricúspide, la válvula aórtica y el septo membranoso. La deformación del trígono fibroso es otro elemento que puede llegar a influir en la regurgitación de sangre en la válvula mitral, pero que no desarrollaremos a lo largo del proyecto.

La parte posterior no se encuentra anclada a ninguna estructura fibrosa. Esta porción de tejido puede dar de sí y permitir una dilatación auricular o ventricular permitiendo la aparición de un aumento en el diámetro del anillo mitral propiciando el prolapso. Esto quiere decir que la regulación del diámetro del anillo mitral es esencial a la hora de reparar la válvula mitral.

El anillo mitral tiene una figura que nos recuerda a una silla de montar, durante la fase sistólica las comisuras mitrales se desplazan apicalmente, mientras que la contracción contribuye al estrechamiento del diámetro de la válvula. Ambos procesos, de dilatación y contracción, contribuyen a una apertura y cierre idóneos, cualquier afección a las zonas adyacentes a la válvula pueden influir de una forma u otra en la forma en que se abre y cierra la misma. ([1])

2.5 Las cuerdas tendinosas

Las valvas están conectadas a las paredes del ventrículo mediante unos manojos de cuerdas tendinosas de número y ramificación variables. Estas desarrollan un rol clave en la anatomía de la válvula, ya que si bien su funcionalidad es simple, son las encargadas de mantener en su sitio los velos de la válvula.

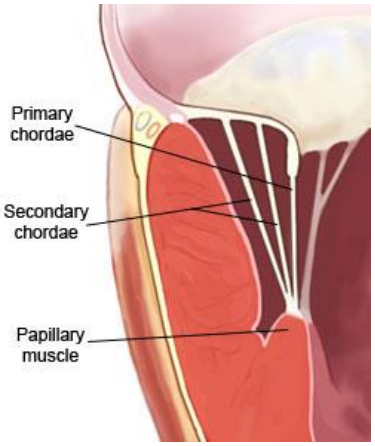


Figura 8: Esquema de inserción de cuerdas

Las cuerdas tienen su origen en alguno de los músculos papilares, que nacen de las paredes del ventrículo. Tomando el punto donde se fijan en el tejido del velo, se pueden clasificar como cuerdas primarias o marginales, cuerdas secundarias o intermedias y cuerdas terciarias o basales.

Las cuerdas primarias o marginales se insertan en el velo libre de la zona de coaptación, y cumplen la función de evitar que el borde del velo se dé la vuelta a causa de la presión intraventricular.

Las cuerdas intermedias no se insertan en el borde libre si no que parten de la superficie inferior del velo y su funcionalidad consiste en liberar el tejido valvular del exceso de tensión, también cumplen la funcionalidad de mantener la integridad estructural de la válvula durante los cambios de presión.

Las cuerdas basales solo están presentes en el velo posterior y su función es la de conectar el anillo y la base velar al musculo papilar. La importancia de este tipo de cuerdas ha salido a la luz cuando se empezaron a realizar intervenciones de reemplazo total de la válvula al descubrir que tras la intervención, su funcionalidad no fuera como antes. Como consecuencia la supervivencia era más bien pobre.

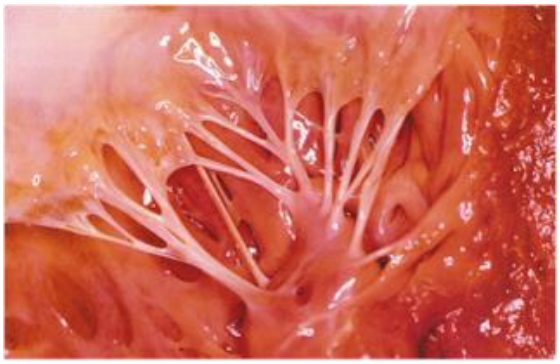


Figura 9: Vista comisural

En esta imagen podemos ver una vista desde la comisura [A3, P3] de la válvula y como las cuerdas se fijan en el músculo papilar correspondiente. ([1])

2.6 Músculos papilares

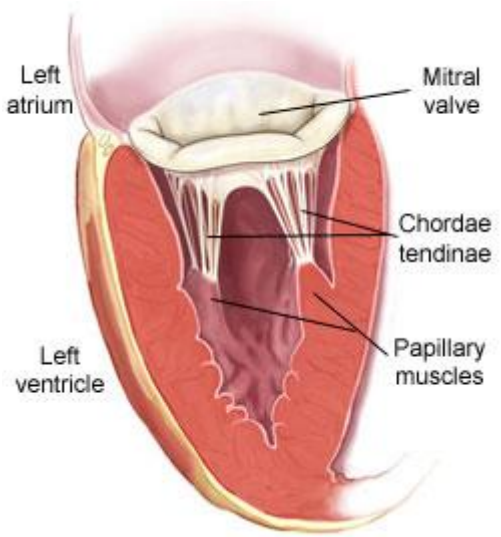


Figura 10: Músculos papilares

Existen dos músculos papilares que están enraizados en las paredes del ventrículo. El musculo anterolateral que se compone de una cabeza o cuerpo y el músculo papilar posteromedial que consta de dos cabezas o cuerpos.

Cada músculo papilar proporciona el anclaje de las cuerdas tendinosas para ambos velos de la válvula. ([1])

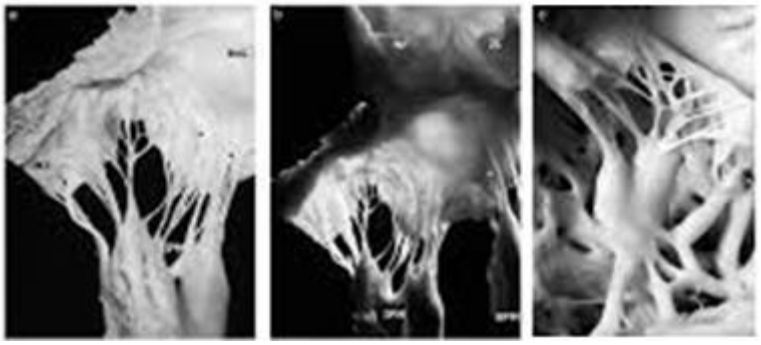


Figura 11: Vista músculos papilares

2.7 Afecciones de la válvula mitral

Existen diversas afecciones que afectan a la válvula mitral. Entre las principales se encuentran las enfermedades degenerativas que consisten en aquellas afecciones que impiden el cierre correcto de la válvula durante la contracción del ventrículo. Dos ejemplos de enfermedades degenerativas son la enfermedad fibroelástica (EFE) y la enfermedad de Barlow (EB). En la siguiente figura se pueden observar las distintas características de estas afecciones.

Características	EFE	EFE avanzada	Forme fruste	Enfermedad de Barlow
				
Edad de diagnóstico	> 60 años	> 60 años	Variable	< 60 años
Historia de IM	< 5 años	< 5 años	Variable	> 10 años
Tejido de las valvas	Normal/translúcido	++	++/+++	+++
Tejido de valva anterior	+	+	++	+++
Tejido de la valva posterior	++	++	++/+++	+++
Segmentos afectados	Único segmento (P2)	Único segmento (P2)	Multisegmento	Multisegmento
Cuerdas tendinosas	Delgadas y rotas	Delgadas y rotas	Variable	Engrosadas y elongadas
Dilatación anular	Ninguna (< 32 mm)	↑ (< 32 mm)	↑↑ (32-36 mm)	↑↑↑ (> 36 mm)
Calcificación	Ninguna	+	–/++	+++

Figura 12: Enfermedades mitrales

2.7.1 Tratamientos

En el caso de la EFE, el tratamiento más habitual implica la reparación de las cuerdas dañadas, o bien estirando y volviendo a atar una cuerda rota o reemplazándola por una cuerda sintética. En caso de que una cuerda esté demasiado elongada, la corrección implica volver a tensarla retirando los tramos de cuerda sobrantes y volviendo a coser la cuerda al músculo papilar correspondiente.

La reparación de una válvula afectada por EB, implica una reparación del prolapso interviniendo directamente sobre los velos, cortando el material sobrante e instalando finalmente un anillo protésico que permite ajustar el diámetro del orificio que regula la válvula. ([3])

Figura 13: Anillo protésico semirígido

3. Análisis de requisitos

El siguiente apartado describe los requerimientos funcionales y no funcionales necesarios para la conclusión del proyecto.

3.1 Requisitos funcionales

Los requisitos funcionales de la aplicación, es decir aquellos que describen el comportamiento que deberá tener el programa son:

1- Seleccionar y deseleccionar una cuerda de la válvula mitral.
2- Mover una cuerda utilizando el teclado.
3- Reestablecer la configuración de cuerdas original.
4- Reestablecer la posición de una cuerda.
5- Borrar configuración guardada.
6- Manipular la cámara, aplicando zoom (con el ratón y el teclado).
7- Inspeccionar la escena desplazando la cámara concéntricamente con la válvula.
8- Cambiar el punto de observación, que puede ser el centro de los velos, el músculo papilar anterior, el músculo papilar posterior y el punto equidistante entre ambos músculos.
9- Grabar una repetición.
10- Iniciar una simulación de la válvula mitral.
11- Guardar o Cargar una configuración de cuerdas determinada.
12- Añadir o eliminar el tabique ventricular en el modo de inspección.
13- Modificar la presión proveniente de la aurícula.
14- Modificar la presión proveniente del ventrículo.

3.2 Requisitos no funcionales

Los requisitos no funcionales son aquellos que describen las limitaciones tecnológicas y condiciones que se deben cumplir para que el proyecto funcione correctamente.

Hardware y SO:

La aplicación funcionará sobre plataforma Windows 7 y sobre Linux.

La aplicación requiere de un equipamiento gráfico potente para poder visualizar correctamente las deformaciones. El proyecto se desarrolló utilizando una tarjeta gráfica modelo Nvida GT540M 2gb presentando latencia gráfica a mayor número de subdivisiones de los modelos.

Periféricos necesarios:

Para poder ejecutar la aplicación es necesario contar con un teclado y un ratón. Además el ratón debe contar con rueda de desplazamiento.

Escalabilidad:

La lógica de la aplicación está gestionada por Blender Game Engine y el estado actual del proyecto permite ampliaciones en diferentes campos, como la disposición de un catálogo de válvulas o la implementación de nuevas funcionalidades en la GUI sin demasiado esfuerzo.

Usabilidad:

La interfaz de la aplicación es sencilla y poco confusa, los iconos utilizados son autodescriptivos y el usuario cuenta con un panel informativo que explica el uso de las teclas de control de la aplicación.

Instalación:

La aplicación no requiere añadir registros en Windows y no hace falta ninguna instalación. El despliegue de la aplicación en producción no representa más complicación que iniciar el ejecutable.

4. Análisis de componentes

Este capítulo describe las pruebas realizadas con diferentes tecnologías en la búsqueda de la herramienta más apropiada para el desarrollo de este proyecto.

4.1 Antecedentes tecnológicos

La válvula mitral ha sido recreada utilizando un modelo deformable interactivo en tiempo real, algo que requiere cierto coste computacional. Se han realizado estudios que evalúan la eficiencia y el coste de los cálculos de las deformaciones de las mallas o meses, ya que en la actualidad en la mayoría de los videojuegos los elementos deformables utilizan animaciones no interactivas.

Tanto para animación como para modelos interactivos, se utilizan implementaciones similares para crear un modelo deformable mediante la física. Describirlas queda fuera del alcance de la memoria. Las siguientes referencias hacen un estudio detallado de su comportamiento así como delo coste que supone su utilización. (Ascher, 2004), (Cotin), (Wong, 2002). Sin embargo, de todas las implementaciones posibles diremos que la implementación final utiliza el modelo masa-muelle.

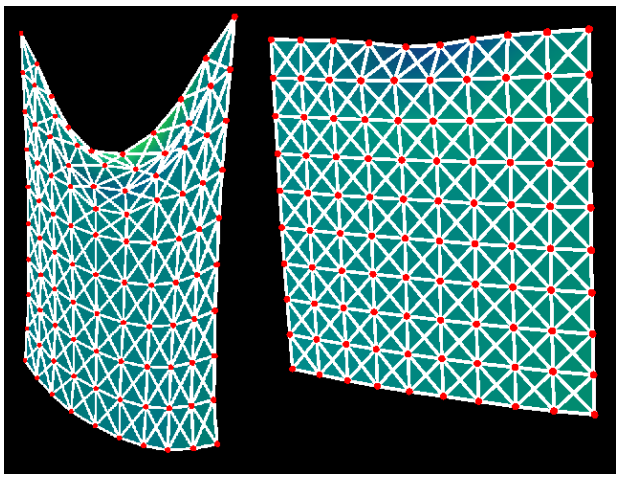


Figura 14: Cloth, modelo masa-muelle

El modelo de deformación masa-muelle consiste en la creación de una red de nodos o masas, interconectadas entre sí mediante muelles, esta red se superpone a una malla o mesh gráfica, asignando conjuntos de vértices a un nodo de esta red deformable. Así pues, cuando un nodo recibe un impulso externo, la fuerza resultante se extiende por los nodos vecinos a través de los muelles. La fuerza aplicada al nodo, lo desplaza y este desplazamiento se traduce en un desplazamiento de posición de los vértices de la *mesh*, lo cual visualmente se traduce en una deformación. Dependiendo de lo que se quiera implementar, la deformación puede ser configurable para obtener deformaciones más o menos rígidas, para que el modelo tienda a mantener su forma original o por el contrario se comporte como un trozo de tela. Esto permite implementar

desde la deformación de un coche al chocar, pasando por una cortina o en nuestro caso, tejidos del cuerpo humano blandos, a esta clase de objetos, a partir de este momento los llamaremos SoftBodies (SB).

Figura 15: Colision modelo mass-spring, implementado en CryEngine

4.2 Tecnologías probadas

Existen muchas implementaciones del modelo masa-muelle y para la realización de este proyecto se han estudiado algunas de ellas antes de escoger la más adecuada. Como generalmente se utilizan para crear animaciones que luego se integran dentro de una aplicación o juego, se ha tenido que valorar no solo la funcionalidad, la documentación y la comunidad sino que además se ha tenido que valorar la interactividad con el usuario en tiempo de ejecución.

4.2.1 VPython y Kinetics Kit

VPython es una variación de lenguaje Python que lleva integrado un módulo gráfico llamado visual. Con *VPython* es posible desarrollar pequeñas aplicaciones gráficas, utilizando geometrías simples como cubos, pirámides, esferas, etc.

Además *VPython* permite la implementación de una interfaz GUI para estas aplicaciones, lo cual facilita la integración de componentes como botones, *radiobuttons*, *sliders*, etc a estas aplicaciones.

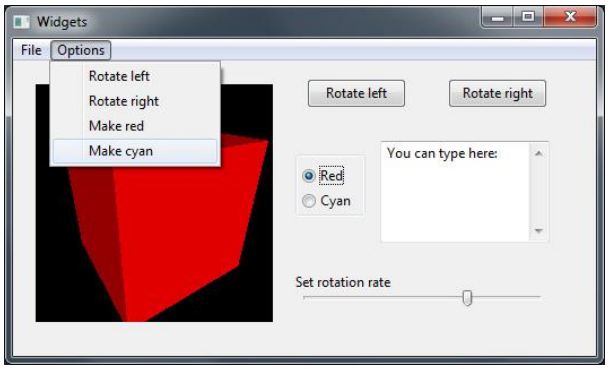


Figura 16: Aplicación VPython

La última versión de *VPython*, que es la versión 6, está basada en la librería *wxpython*, lo cual permite capturar y

gestionar eventos de pantalla en las plataformas Windows, Linux e iOS.

Kinetics Kit por otro lado, es un módulo físico desarrollado para poder crear simulaciones físicas realistas. Se pueden configurar diferentes entornos físicos y crear simulaciones para estudiar el comportamiento de objetos en diferentes escenarios.

KineticsKit incluye en su motor físico la posibilidad de implementar un modelo masa muelle a muy bajo nivel, permitiendo añadir uno a uno los nodos de masas, interconectarlos y simular las deformaciones añadiendo impulsos a los distintos nodos para poder estudiar su comportamiento.

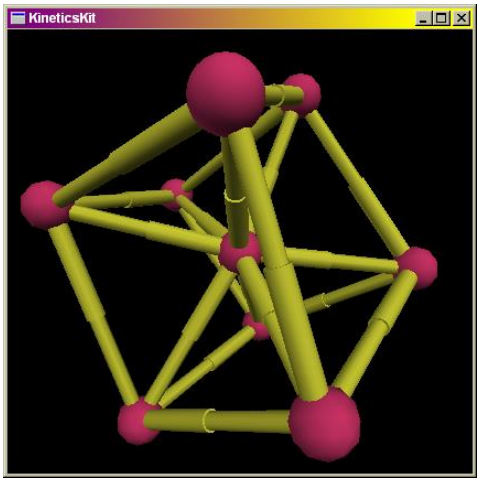


Figura 17: Modelo masa-muelle

Esta implementación a bajo nivel de *KineticsKit* permite identificar y almacenar los datos de los nodos del modelo en la lógica de la aplicación, fijar nodos específicos del modelo al espacio y asignar distintas propiedades físicas a distintas partes del modelo. Es decir se pueden implementar modelos con áreas más elásticas y otras más rígidas.

Además *KineticsKit* funciona sobre *VPython*, esto quiere decir que parte de la interacción con el usuario ya es capturable.

Las posibilidades de *KineticsKit* son altas, pero para conseguir los objetivos del proyecto requeriría la implementación de un sistema masa-muelle que se acoplara con una *mesh* deformable antes de poder empezar con la implementación misma del proyecto. Y esto está fuera del alcance del proyecto. Es posible detectar con código *VPython* la interacción del usuario con los objetos de escena y calcular las colisiones y deformaciones utilizando el motor de físicas, sin embargo cuando el usuario selecciona un objeto que no contempla el motor de colisiones (aquellos que el motor utiliza para representar elementos físicos, tales como tubos y esferas), se producen errores de implementación en el mismo código de la librería ya que estos objetos de *VPython* no tienen los atributos lógicos asignados por *KinecticsKit* tales como el valor de la masa. Es decir que esto supone una limitación sobre los elementos que se pueden utilizar o bien implica que hay modificar la librería para que sea capaz de diferenciar cuales son objetos físicos y cuales son objetos gráficos.

Sin embargo es necesario tenerlo en cuenta dado su potencial para crear un motor de deformaciones basado en el modelo masa-muelle.

Utilizando esta tecnología se desarrollaron dos pruebas. La primera orientada a comprobar si se podía crear una *mesh* y asignar una red simple de masas a dicha *mesh*. Mediante la segunda prueba verificamos la posibilidad de crear conjuntos de filamentos deformables a partir de una ristra de masas y *springs*. Ambas pruebas se encuentran en el anexo 1 de este documento.

4.2.1.3 Conclusiones de las pruebas

Tras ver algunos de los ejemplos de implementación, podemos hacer un resumen de la tecnología probada.

Los enlaces a la documentación de la librería no funcionan siempre, algunos links están rotos. Además no hay una comunidad de desarrolladores o foros de preguntas de la plataforma que de apoyo al proyecto. Sin embargo la configuración del motor físico es elevada. Al ser un lenguaje que trabaja a tan bajo nivel, el abanico de posibilidades es elevado, lo mismo ocurre con los *SoftBodies* que son altamente manipulables.

El uso de la librería es gratuito y sin embargo como ya se ha mencionado con anterioridad, el coste de desarrollar con esta tecnología es muy elevado, ya que aunque *vpython* detecte algunas de las interacciones de ratón y teclado por defecto, habría que construir todo un *framework* alrededor de la librería para poder comenzar a desarrollar el juego en sí mismo.

4.2.2 Panda 3D + bullet

Entramos en el apartado de los *GameEngines* probando Panda 3D. El peso computacional recae sobre el código C++ del *framework* y este ofrece una API en Python con el objetivo de facilitar el trabajo de codificación al programador.

Esta separación del motor en dos partes ofrece al desarrollador las ventajas que da desarrollar en un lenguaje sencillo de aprender como Python y al mismo tiempo conserva el rendimiento del código compilado del núcleo del *framework*.

El motor físico que utiliza Panda 3D es una librería externa llamada *Bullet*. Esta librería de físicas es muy completa y multiplataforma, utilizada para dar soporte físico en herramientas

como *Cinema4D, Blender, 3DMark11*, etc. Lo más importante que cabe destacar de esta librería es que da soporte a la creación y gestión de *SoftBodies*.

Una de las características más interesantes de Panda 3D es la gestión de objetos de escena. A diferencia de otros *engines* que utilizan una estructura de datos tipo lista, Panda 3D dispone de un grafo llamado *SceneGraph*. Este grafo tiene forma de árbol y en cada nodo se almacena un objeto llamado *NodePath*. Cada *NodePath* está asociado a un objeto y tiene la capacidad de anidar otros *NodePaths*(NP). Esto permite acceder a una gran cantidad de objetos relacionados entre sí de forma sencilla o crear objetos compuestos e insertarlos de forma anidada en el *SceneGraph* para obtener una estructura mejor organizada. La raíz de dicho grafo es el objeto *render*.

El objeto *render* no posee una geometría propia, sin embargo cualquier objeto que tenga que ser renderizado debe estar conectado a *render* de forma directa o indirecta. También es posible almacenar nodos que no contienen geometría en *SceneGraph* pero que puede servir para crear atributos compartidos entre distintos tipos de objetos. Por ejemplo, en la siguiente figura podemos ver un ejemplo de *SceneGraph* que representa la implementación de un sistema solar que contiene el fondo o cielo (*sky*), el sol y los primeros cuatro planetas. En el diagrama se puede observar como todos los elementos están conectados con el objeto *render*, además podemos ver que salvo el sol y el cielo, todos los elementos están divididos en dos componentes. La información geométrica de cada objeto está guardada en una hoja del árbol y la información lógica de cada objeto está almacenada en un objeto llamado *self.orbit_root*. Finalmente se puede observar como el objeto tierra anida tanto la información del planeta Tierra como de la luna.

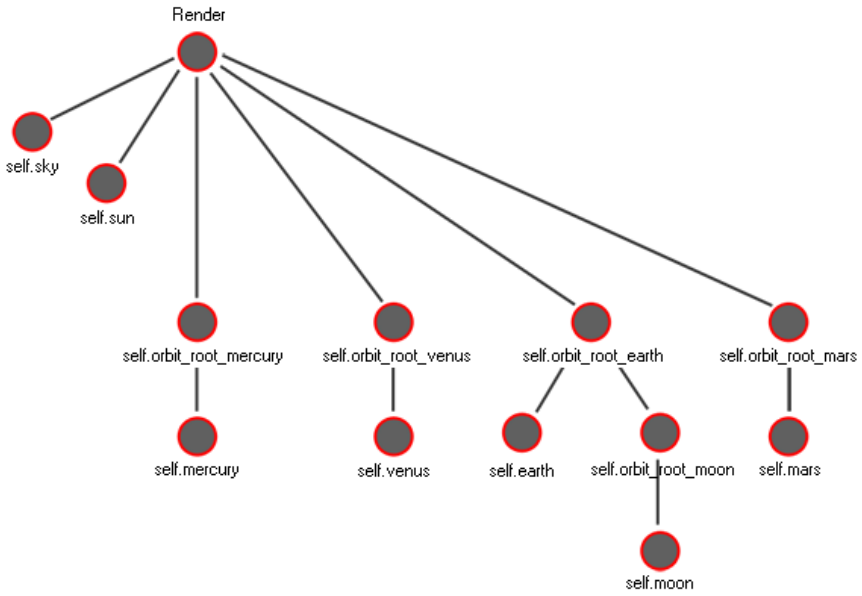


Figura 18: SceneGraph

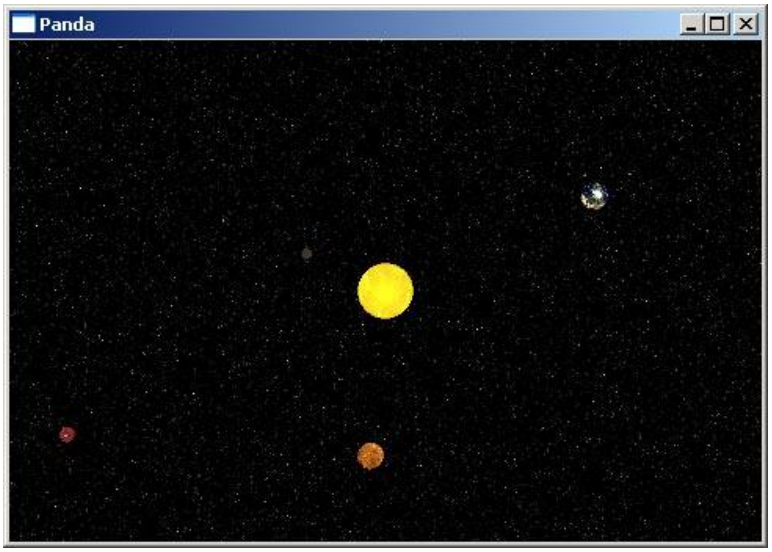


Figura 19: Ejemplo de implementación

Dado que la implementación del modelo masa-muelle recae en la librería externa, podemos centrarnos en el desarrollo de una serie de pruebas que tienen por objetivo conseguir un modelo físico funcional de una compuerta que se abre y se cierra controlada por un filamento, el objetivo de dicha prueba es emular el comportamiento de una válvula mitral.

En el capítulo del Anexo 1 se pueden encontrar las dos implementaciones llevadas a cabo con esta herramienta para probar la viabilidad de su uso en la implementación de este proyecto.

4.2.2.4 Conclusiones de las pruebas

Panda 3D es un *Game Engine* bastante completo que permite el desarrollo de videojuegos a nivel profesional y particular, es una herramienta gratuita con una documentación muy completa y cuenta con una gran comunidad de desarrolladores que impulsan y mantienen activo el proyecto. Además cuenta con el respaldo de haber sido utilizado para el desarrollo de grandes producciones comerciales en títulos como Piratas del Caribe de la factoría Disney.

Como se ha podido observar en las pruebas realizadas implementar un sencillo sistema de clases a partir de las funcionalidades aportadas por la API no es complejo y ha permitido realizar todas las pruebas evitando en gran medida la redundancia de código.

La librería de físicas es altamente configurable y permite un manejo básico de objetos rígidos, lo cual permite crear desplazamientos, objetos que caen y colisiones de forma rápida y sencilla.

Sin embargo durante las pruebas hemos podido apreciar la dificultad de manejar *SoftBodies* con esta herramienta. Al carecer de una herramienta de edición no se pueden manipular los objetos de forma visual y específicamente los *SoftBodies* aíslan la malla para que no sea accesible por el desarrollador ya que esta se reescribe en cada *frame*, lo cual impide identificar qué puntos se desean asignar como posiciones de anclaje. La vaguedad con la que se puede manejar un *SoftBody* con esta herramienta hace que sea inviable el desarrollo del proyecto con Panda 3D. Por otro, aunque esto hubiera sido posible es preocupante la caída de fps con objetos tan simples como una esfera cuando esta es un *SoftBody*.

4.2.3 Unity 3D

Unity 3D también es una herramienta que está volviéndose muy popular para el desarrollo de videojuegos. Este *GameEngine* proporciona una gran cantidad de herramientas para la creación de aplicaciones propias y proporciona facilidades a los equipos de desarrollo para la comercialización de sus productos.

De todas las tecnologías estudiadas esta es la única que no funciona con Python. Los lenguajes que soporta *Unity* 3D son 3, C#, *UnityScript* que no es más que una variante de JavaScript y *Boo*, un lenguaje que guarda ciertas semejanzas con Python.

Unity3D permite crear ejecutables *StandAlone* que funcionan no solo en Windows, Linux o Mac sino que además permite crear ejecutables para plataformas móviles como Android o iOs y consolas como Ps3, Wii o Xbox360, así como *WebPlayer* lo cual permitiría ejecutar la aplicación desde un navegador.

Esta herramienta está diseñada para proporcionar al desarrollador una interfaz de edición y visualización que facilita el trabajo de gestionar los elementos en una escena de forma que el resultado final se puede apreciar antes de poner en marcha la aplicación.

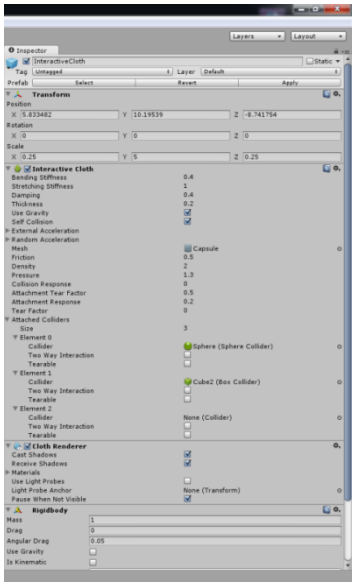


Figura 20: Panel SoftBody

Unity incluye un motor de físicas muy completo basado en cajas de colisión que permite controlar las colisiones de forma sencilla y así cubrir las necesidades de la mayor parte de los desarrolladores en este aspecto.

Todos estos aspectos hacen que haya valido la pena estudiar brevemente sus funcionalidades.En el aspecto de los *SoftBodies*, *Unity* tiene un apartado que permite agregar telas o *cloths* de forma sencilla.

Como se puede observar en la figura 20, hay una gran cantidad de parámetros configurables tal como sus propiedades físicas. Además se pueden añadir de forma visual los anclajes, de tal forma que se puede decidir en donde anclar un *SoftBody* a otro objeto o al espacio.

Sin embargo para implementar modelos *SoftBody* con presión interna se requiere el uso de un script externo que simular el modelo masa-muelle, utilizado para las pruebas.

Para entender cómo funciona mejor esta tecnología el lector puede referirse al anexo 1 donde se describen las pruebas realizadas.

4.2.3.3 Conclusiones de las pruebas

En resumen *Unity* 3D es una de las herramientas probadas más elaboradas que proporciona más facilidades al desarrollador. Sin embargo estas facilidades pueden llegar a esclavizarnos a una serie de técnicas que podrían no ser las más adecuadas para los objetivos del proyecto.

Este entorno de desarrollo puede llegar a ser muy potente en otros contextos ya que es posible conseguir programas funcionales con muy pocas líneas de código como el caso de las pruebas antes mencionadas que no requirieron ninguna línea de código. El único trabajo que supusieron fue la creación del segundo modelo modificado del *Thorus* y agregar las configuraciones necesarias para las pruebas.

Por otro lado, es justo mencionar que es destacable la cantidad de plataformas que son compatibles con los programas que se pueden desarrollar con esta tecnología. Además la comunidad de desarrolladores que dan apoyo al proyecto incluso aportan sus trabajos a una tienda online de modelos, scripts y escenarios que puede facilitar el trabajo al desarrollador.

En el aspecto de los *SoftBodies* a pesar de que inicialmente son muy fáciles de integrar configurar y manipular, estos dan serios problemas visuales que son independientes del modelo.

Unity 3D es una herramienta gratuita pero también existe una versión de pago que tiene muchas más características que no se han probado durante las pruebas realizadas.

4.2.4 Blender Game Engine

Blender es una completa unidad de diseño utilizada en el entorno profesional para la creación y animación de modelos 3D. Esta popular herramienta tiene una alta capacidad de crear imágenes y videos renderizados de muy alta calidad. En general esta herramienta gratuita no tiene nada que envidiar a sus equivalentes de pago.

Lo destacable de esta herramienta es que posee un *Game Engine* integrado que aunque no es tan popular como su apartado de diseño, es una herramienta lo suficientemente potente para crear productos de calidad.

En el apartado de físicas, igual que Panda 3D, BGE (*Blender Game Engine*) y BR (*Blender Render*) utilizan la librería externa *Bullet*, aunque la integración de esta librería está implementada de otra forma.

Las siguientes pruebas están orientadas a comprobar si esta integración de Bullet es viable para el desarrollo del proyecto y permitirá mostrar cómo crear una aplicación con esta herramienta.

4.2.4.1 Desarrollo de aplicaciones con BGE

El desarrollo de aplicaciones gráficas utilizando BGE es similar al de otras herramientas ya descritas. No obstante hay que destacar una serie de peculiaridades de BGE sobre las otras herramientas.

Scripts y Módulos: Para la creación de scripts BGE da soporte a tres formas de integrar código que controle la aplicación.

La primera de ellas son los scripts internos. Estos se crean con un editor propio de Blender, el código no se almacena en ficheros separados si no que se guardan dentro del proyecto con el nombre del script correspondiente.

La segunda alternativa es la creación de ficheros externos que cumplen la misma funcionalidad que los scripts internos con la posibilidad de poder ser editados con cualquier editor de texto convencional.

La tercera opción es la creación de módulos creándolos como clases con métodos propios. Estas clases hacen las veces de controladores, ya que los métodos son llamados desde la capa lógica de BGE utilizando lo que se denomina *Logic bricks* que se describirá a continuación.

Logic bricks: Los logic bricks son un sistema de control visual que permite crear funcionalidades de forma rápida. Existen 3 tipos de logic bricks que se pueden interconectar entre ellos para generar comportamiento.

Los primeros de ellos son los sensores. Como se puede observar en la figura 59 hay diferentes tipos de sensores, que en resumen sirven para detectar inputs externos, tales como colisiones, pulsaciones de teclas o movimientos de ratón.

En segundo lugar tenemos los controladores. Estos pueden ser también de distinto tipo o incluso simplemente booleanos que regulan la interacción entre distintos tipos de sensores. Cuando queremos utilizar un script o un método de un controlador Python es aquí donde se selecciona.

Finalmente están los actuadores que pueden definir un comportamiento a consecuencia del estímulo recibido por un sensor. En la figura 59 podemos ver un ejemplo de asignación de comportamiento donde el programa reinicia la escena en caso de que el usuario presione las teclas W o barra espaciadora.

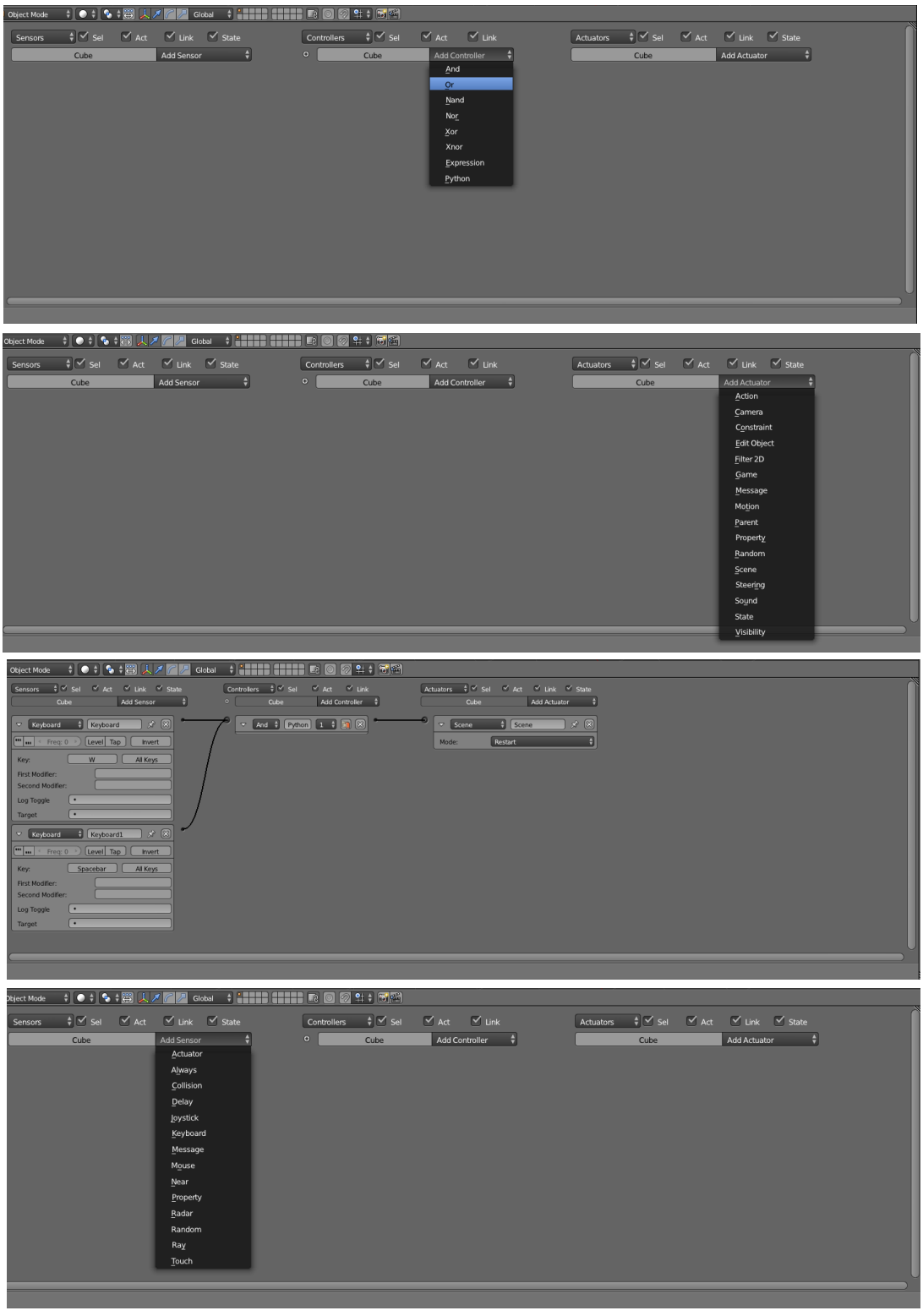


Figura 21: Ejemplo

4.2.4.2 Técnicas de modelado

Independientemente de la tecnología utilizada para el desarrollo de la aplicación, dentro del apartado gráfico, fue necesario adquirir algunas habilidades de creación de modelos en 3D, ya que los modelos ya existentes de válvulas o bien eran demasiado densos en cuanto a geometría y por tanto computacionalmente costosos o bien no eran visualmente realistas.

El apartado de edición y creación de modelos de Blender es muy completo y ofrece todas las opciones que se pudieran requerir para modelado a nivel profesional.

Existen diversas técnicas de creación de modelos en 3D pero el objetivo común a todas las técnicas de modelado es mantener el nivel de polígonos necesario para conseguir la relación entre calidad y realismo óptimos.

Para el desarrollo del proyecto se utilizaron dos técnicas de creación de modelos en 3D. La primera de ellas se conoce como box modelling y la segunda NURB surface modelling.

Box modelling: Esta primera técnica parte de un cubo al que se le aplica la funcionalidad de subdivisión de caras para poder esculpirlo.

Para poder aplicar la subdivisión de caras de un modelo, basta con seleccionar una mesh, en este caso un cubo y pasar al modo de edición pulsando la tecla TAB. Veremos todos elementos de la mesh seleccionados cambiarán de color y el panel izquierdo cambiará. Dentro del apartado Add, encontraremos una opción denominada Subdivide que nos permitirá subdividir un elemento seleccionado, esto se puede aplicar a aristas, vértices y caras.

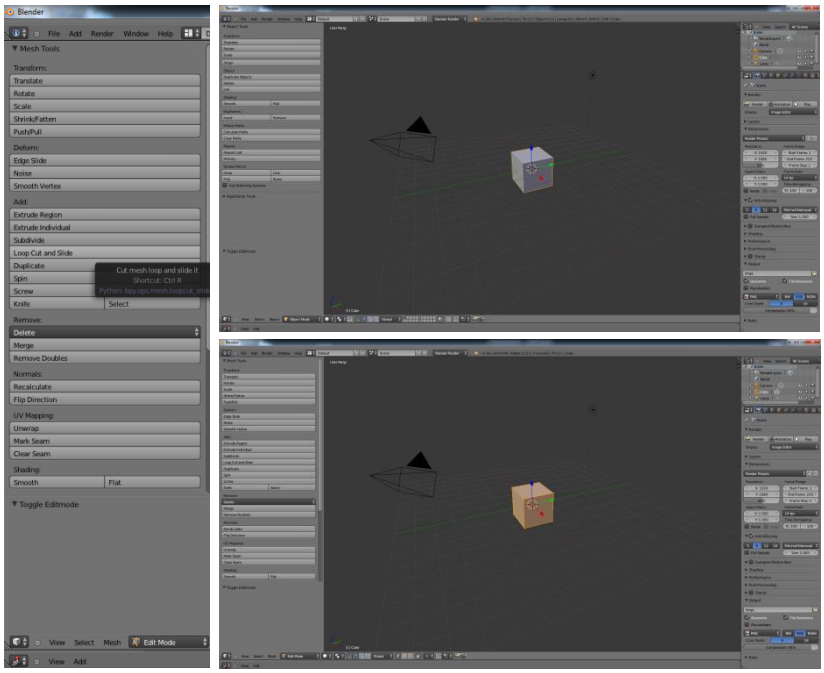


Figura 22: Modo Edición

Cuantas más veces pulsemos el botón Subdivide más elementos nuevos obtendremos, pero hay que tener en cuenta que a mayor número de elementos más costoso será luego de renderizar el modelo y de calcular sus deformaciones, aunque también el modelo será más realista.

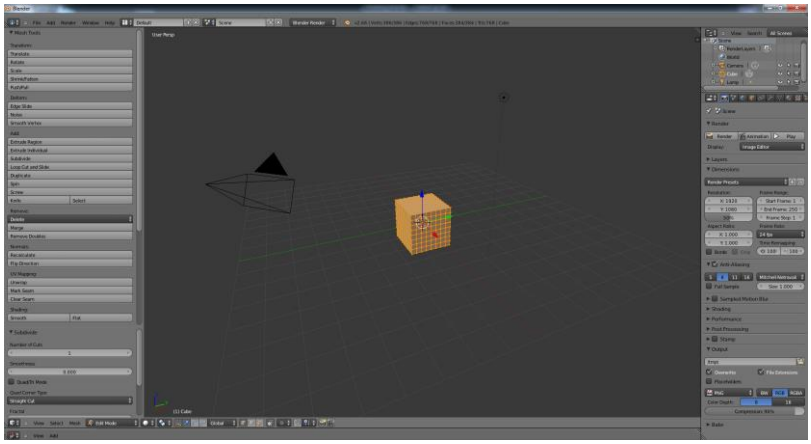


Figura 23: Cubo subdividido

Figura 24: Panel de selección

Como se puede observar en la figura 24, existen 3 tipos de selección de elementos de una malla. El primero permite seleccionar vértices, el segundo aristas y el tercero caras. El último botón de este panel, de color blanco, habilita la visibilidad de elementos que se encuentran ocultos por otros, es decir una especie de visión de “rayos x”.

Para este ejemplo se seleccionó el tercer modo de selección y se pulsará una cara cualquiera con el botón derecho. Una vez hecho, mediante la tecla E del teclado se generará una nueva cara que podremos desplazar con el ratón como se puede ver en la imagen 63 o bien se puede desplazar la cara directamente deformando el modelo.

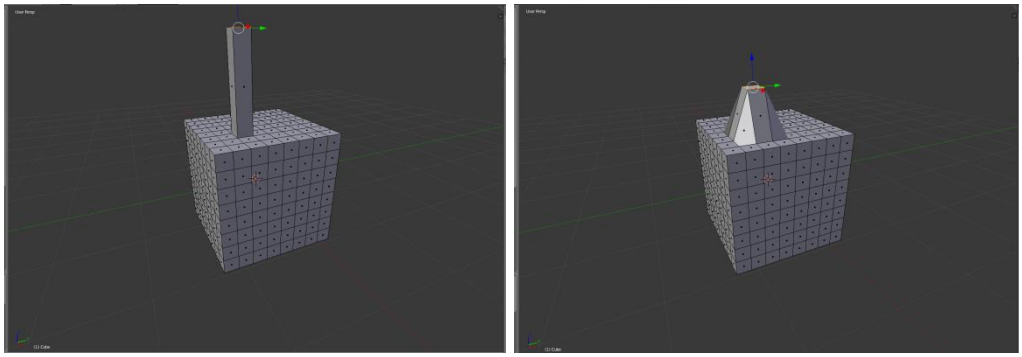


Figura 25: Extrusión vs Deformación

Esta técnica de modelado permite crear modelos sencillos de forma rápida pero para crear objetos curvos puede llegar a resultar muy complicado o incluso impracticable.

NURB surface modelling: Esta técnica de modelado sirve para la creación de modelos que contienen varias curvas complejas que de otro modo serían difíciles de conseguir. El concepto consiste en partir de un objeto curvo que proporciona Blender como base y deformar su curvatura para obtener la forma deseada.

En primer lugar escogemos el tipo de superficie que queremos utilizar como base, en este ejemplo se usa un NURBS Thorus.

Figura 26: Selección de NURBS Thorus

En pantalla aparecerá el Thorus seleccionado, basta con pulsar TAB para entrar en el modo de edición.

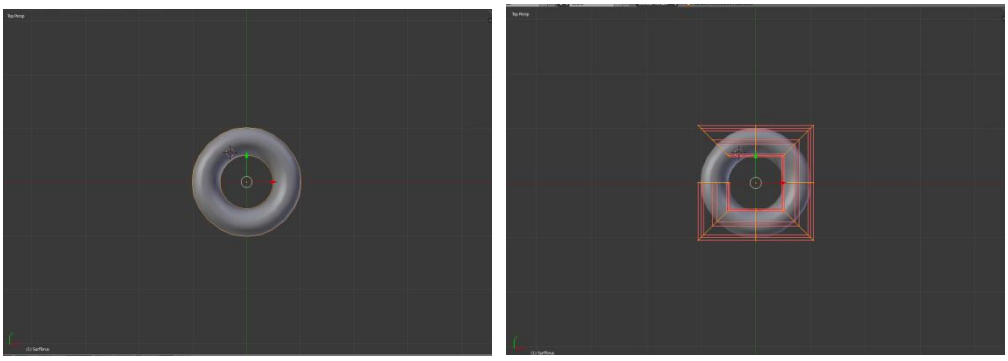


Figura 27: Edición de una NURBS surface

Dentro del modo de edición basta con seleccionar las guías para deformar la superficie NURBS, una vez hemos terminado y la deseamos convertir en una malla, se pulsa ALT+C y se selecciona la opción que permite la conversión a una malla regular.

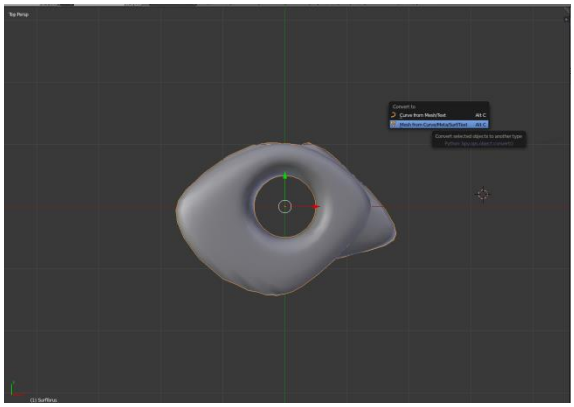


Figura 28: Conversión a mesh

4.2.4.3 Conclusiones de las pruebas

Blender resulta ser una herramienta gráfica potente que cuenta con una documentación muy completa, así como cursos impartidos tanto para diseñadores como para desarrolladores. La comunidad que hay detrás de esta tecnología es muy amplia dado el uso que se le da a nivel profesional.

En cuando al manejo de SBs y a la configuración del entorno físico, encontramos en BGE muchas opciones que permiten la implementación y manipulación de SoftBodies de forma razonable, así como identificar manualmente puntos de fijación y controlarlos fácilmente mediante código.

El coste de la aplicación es gratuito lo cual no deja de sorprender si lo comparamos con herramientas similares cuyas licencias pueden tener un coste elevado.

Finalmente el desarrollo de la aplicación no tiene por qué suponer un coste demasiado elevado gracias al sistema de LBs que descarga al programador de solucionar problemas resueltos ya muchas veces y delega en él la creación de un sistema de controladores para gestionar el comportamiento de los objetos de la aplicación.

4.3 Comparativa de tecnologías

En la tabla que hay a continuación se hallan las características de todas las tecnologías comparadas entre sí. Según se ha podido ver en las pruebas realizadas, cada una de las tecnologías tiene sus puntos a favor y en contra. Sin embargo la característica a la cual se le ha dado más peso ha sido al manejo de SBs. En este sentido la tecnología que más facilidades ha proporcionado es Blender, aunque pierde en ostros aspectos como la generación de ejecutables para diferentes plataformas o la cantidad de lenguajes soportados. A pesar de todo en esta comparativa hemos probado algunas herramientas potentes y populares y bajo otras circunstancias es posible que la decisión final hubiera sido distinta. Sin embargo para este proyecto la usabilidad de los SBs ha sido crucial. Es por este motivo que a pesar de no ser la mejor herramienta en todos los campos se ha escogido la herramienta Blender para la implementación del proyecto.

	Vpython + KineticsKit	Panda 3D	Unity 3D	Blender
Lenguaje base	Python	Python	C#/UnityScript/Boo	Python
Sistema operativo	Windows/Linux	Windows/Linux/ OsX	Windows/Linux/OsX/Android/Web Player/Wii/Ps3/Xbox 360	Nativo SO
Documentación	Incompleta	Completa	Completa	Completa
Comunidad desarrolladores	Inexistente	Elevada	Muy elevada	Muy elevada
Configuración Físicas	Elevada	Elevada	Elevada	Elevada
Manipulación de SoftBodies	Falta desarrollo	Insuficiente	Defectuoso	Aceptable
Tipo de tecnología	Librería de físicas	Game Engine	Game Engine	Game Engine
Coste	Gratuito	Gratuito	Gratuito/De pago	Gratuito
Coste desarrollo	Elevado	Intermedio	Bajo	Intermedio

5. Modelo de deformación y visualización

El modelo final utilizado para la implementación del proyecto consiste en una malla de las siguientes características geométricas:

Vértices	30619
Caras	31813
Triángulos	59978

Estructuralmente la válvula consta de los 2 velos correspondientes y 25 cuerdas tendinosas, donde cada cuerda es la representación de un manojo de cuerdas reales. Esta decisión conceptual es así para prevenir la aparición de latencia gráfica por exceso de cálculos en la deformación. El objeto velo válvula tiene insertadas las cuerdas a lo largo de la zona de coaptación. Cada cuerda del modelo representa tanto cuerdas primarias como secundarias .El modelo de la válvula se encuentra ubicado en el modelo estático de un ventrículo izquierdo que es ajeno a la simulación como se puede ver en la siguiente figura.

Figura 29: Representación de válvula

El modelo de la válvula está representado por un único objeto Soft Body con las siguientes características físicas:

Preservación de forma	0.04
Elasticidad lineal	1.0
Fricción	0.2
Masa	1.0

Las características físicas del modelo son arbitrarias para este modelo conceptual. Sin embargo los modelos reales de válvula también varían de caso a caso según la edad de la persona y estos valores deberían cambiar de modelo a modelo o bien disponiendo de un catálogo de válvulas diferentes o modificando estos valores a un mismo modelo, cosa que en la versión

actual de *bullet* no se puede realizar en *blender*, aunque hay que tener en cuenta que en próximas versiones se podrá hacer y se podrá incorporar como funcionalidad a la aplicación.

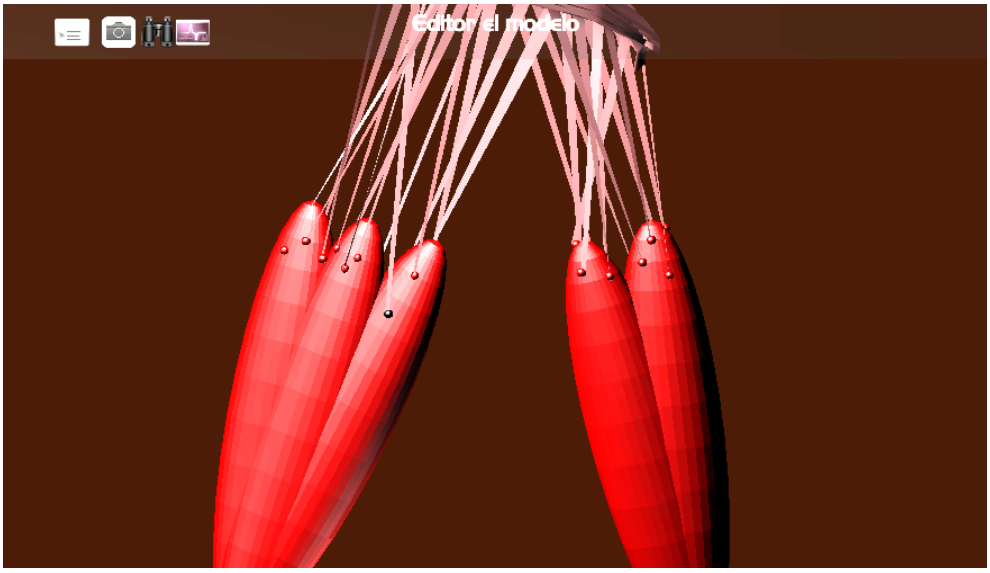


Figura 30: Cuerdas y anclajes

Como se puede observar en la figura 30 cada cuerda tiene asociada un objeto esférico que representa un punto de referencia. Estas esferas son objetos tipo *RigidBody* que no pertenecen realmente al modelo si no que sirven de guía interactiva para el usuario y a la vez están vinculados físicamente con su cuerda de tal forma que el usuario lo que hace es interactuar con la esfera para generar una fuerza que se aplica sobre el objeto *SoftBody* deformando el objeto. Inicialmente se escogió como punto de referencia el extremo de una cuerda pero también es posible escoger otros puntos de referencia dentro de la misma cuerda.

Cuando se estira una de las referencias mencionadas se genera una fuerza en una dirección determinada esta fuerza la aplica el motor de físicas al modelo de tal forma que se va diseminando por la red del modelo masa-muelle, esta fuerza afecta de distintas formas en función del parámetro de elasticidad lineal y de la preservación de la forma. De tal forma que a mayores ratios de Preservación de la forma, se requieren fuerzas mucho más elevadas para provocar una deformación y en caso de que se deforme tiende a recuperar su forma original.

Figura 31: Estiramiento de cuerda

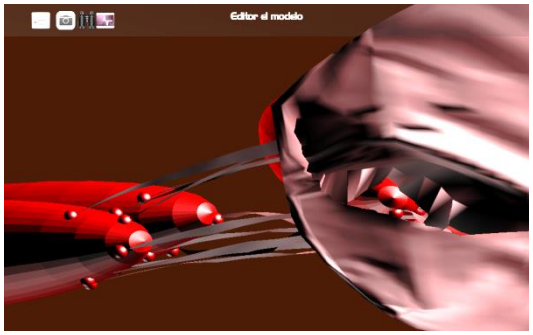


Figura 31: Deformación

Finalmente, el modelo cuenta con un generador de fuerzas que emulan el comportamiento de la sangre dentro del ventrículo y la aurícula. El modo simulador está configurado para producir un cambio en el sentido de la presión cada segundo.

Con el objetivo de simular la presión intraventricular se configuraron 6 focos de fuerza que simulan la presion que recibiría la válvula proveniente desde el ventrículo en distintas direcciones y ejercen una fuerza con una dirección dada mediante el vector calculado entre el centro de la válvula y el foco de generación de fuerza. Del mismo modo se calculan otros 6 focos que simulan la presión proveniente de la aurícula. Dichas fuerzas ejercidas sobre el modelo provocan las deformaciones necesarias para abrir y cerrar la válvula mitral.

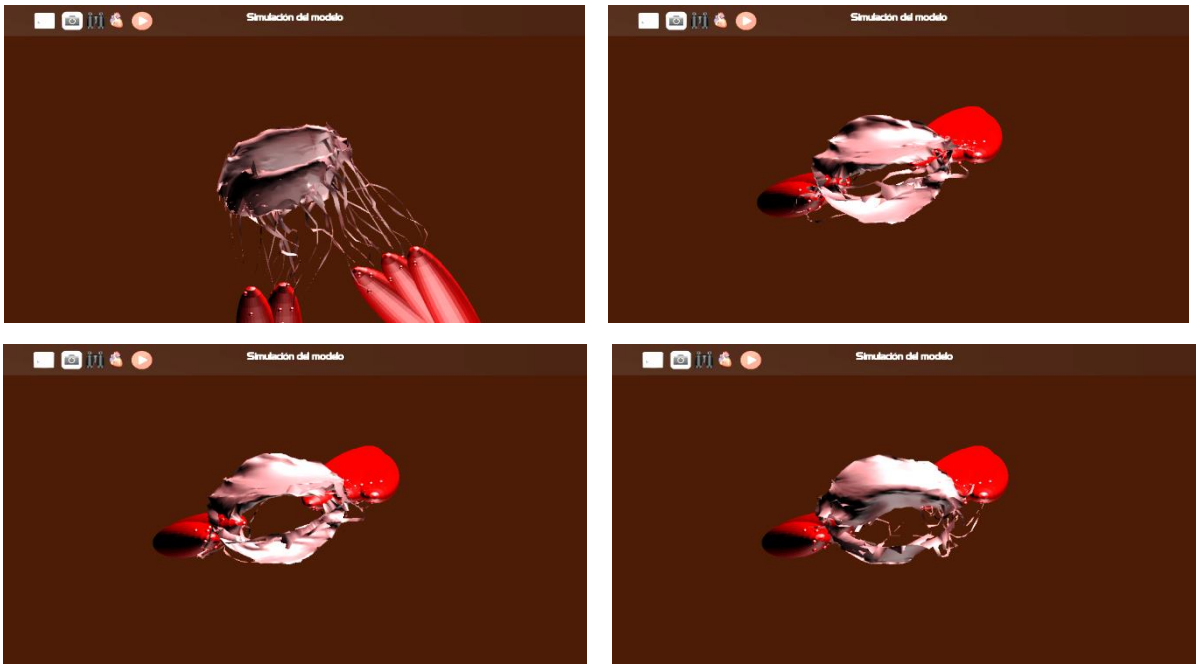


Figura 32: Válvula deformada

6. Arquitectura del Software

El siguiente capítulo detalla la especificación y diseño de la aplicación del proyecto. En primer lugar se detallan los casos de uso de los cuales se obtienen los requisitos funcionales de la aplicación con sus respectivos diagramas de caso de uso. A continuación se detallará el diagrama de clases de la lógica de la aplicación y se justificará su implementación.

6.1 Especificación de la aplicación

Estos son los casos de uso que describen el comportamiento de las funcionalidades de la aplicación. Para cada caso de uso se describirá la relación entre el usuario y el sistema, así como el escenario principal y el diagrama correspondiente.

6.1.2 Casos de uso

1- Grabación de una repetición

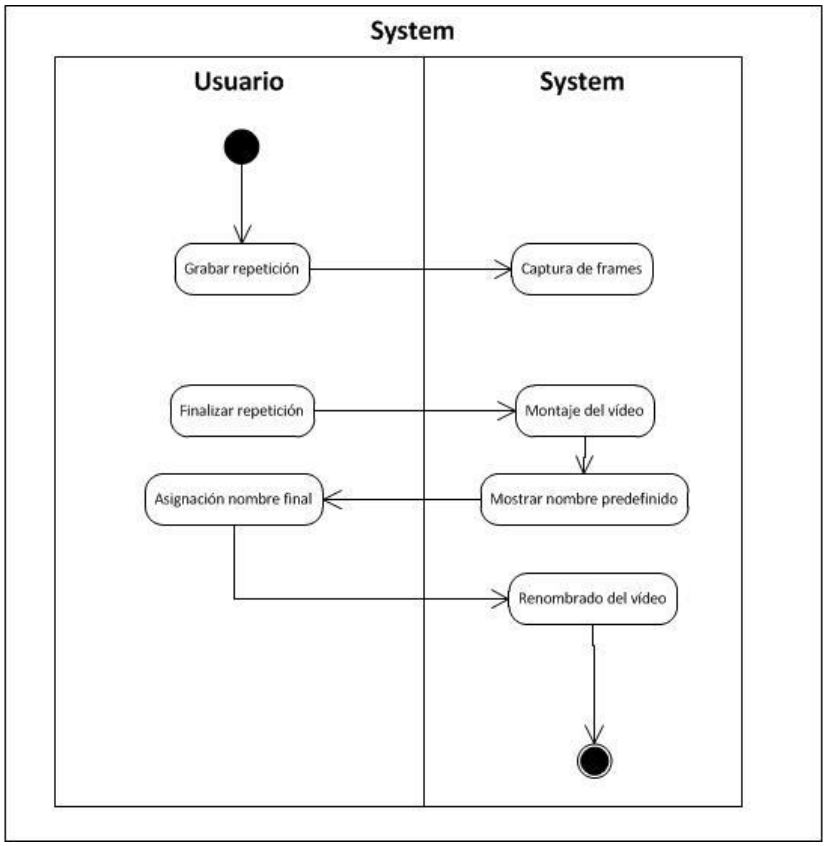
Actor principal: Usuario

Precondiciones: No debe estar grabándose previamente una repetición. El usuario debe estar en el modo de simulación.

Accionador: El usuario desea guardar una repetición de la simulación sobre una configuración determinada.

Escenario principal:
1- El usuario indica que quiere grabar una repetición.
2- El sistema inicia la captura de frames
3- El usuario desea terminar la repetición.
4- El sistema muestra al usuario el nombre de la repetición predefinida.
5- El usuario asigna un nombre al vídeo.
6- El sistema almacena la repetición con el nombre asignado.

Otros: -



2- Mover cuerda

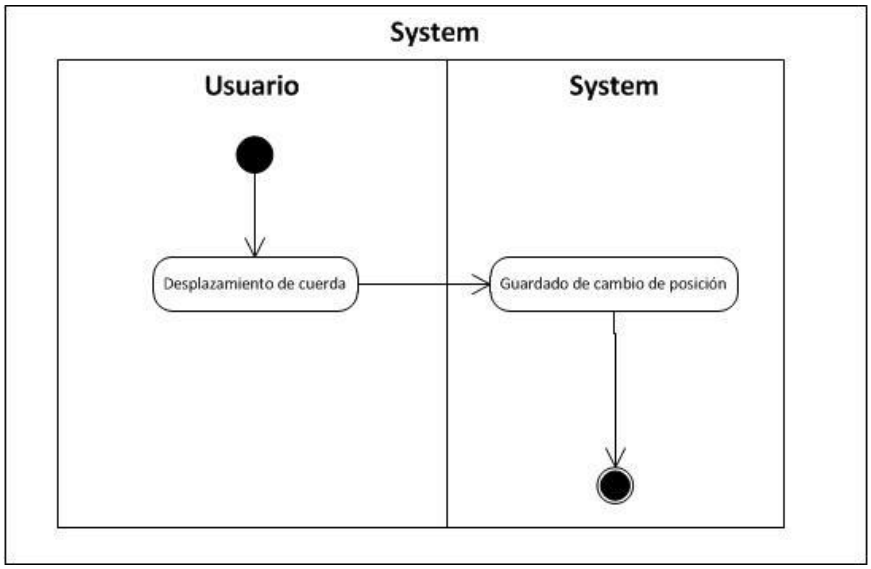
Actor principal: Usuario

Precondiciones: La cuerda debe estar seleccionada

Accionador: El usuario desea desplazar una cuerda seleccionada por el escenario utilizando el ratón o el teclado desplazando el ancla de la cuerda.

Escenario principal:
1- El usuario desplaza el anclaje de una cuerda.
2- El sistema almacena la nueva posición del anclaje de la cuerda.

Otros: -



3- Cambiar punto de estiramiento de una cuerda

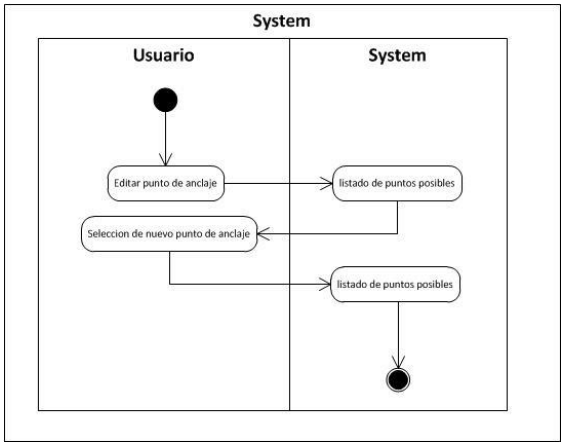
Actor principal: Usuario

Precondiciones: La cuerda debe estar seleccionada.

Accionador: El usuario desea cambiar el punto por el cual estira una cuerda.

Escenario principal:
1- El usuario indica al sistema que quiere editar el anclaje de una cuerda.
2- El sistema indica al usuario que puntos de anclaje puede elegir.
3- El usuario escoge un nuevo punto de anclaje para la cuerda.
4- El sistema almacena el punto de anclaje elegido.

Otros: -



4- Mover cámara.

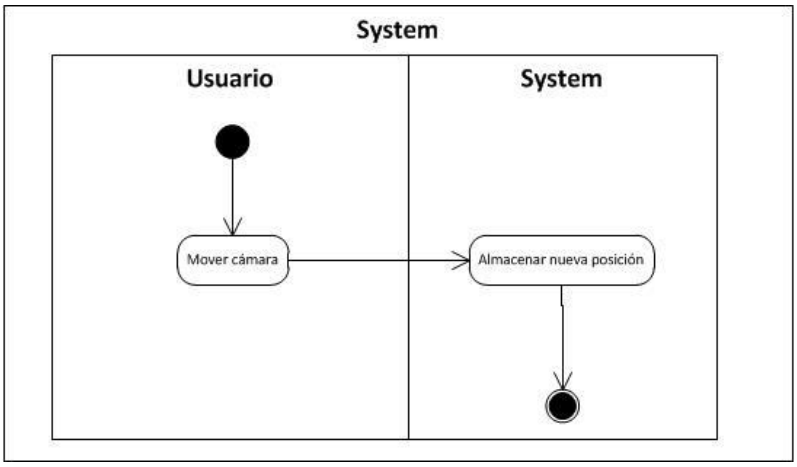
Actor principal: Usuario.

Precondiciones: El usuario debe estar en el modo de edición, simulación o inspección.

Accionador: El usuario desea cambiar el punto de vista para inspeccionar la escena.

Escenario principal:
1- El usuario indica al sistema la nueva posición de la cámara.
2- El sistema almacena la nueva posición de la cámara.

Otros: -



5- Tapar/Destapar el ventrículo izquierdo

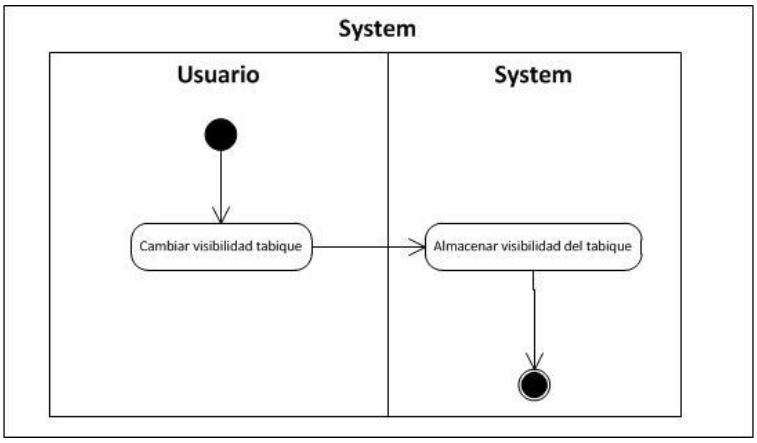
Actor principal: Usuario.

Precondiciones: El usuario debe estar en el modo de inspección.

Accionador: El usuario quiere hacer visible u ocultar el tabique que separa los ventrículos para observar mejor el interior del ventrículo izquierdo.

Escenario principal:
1- El usuario indica al sistema el nuevo estado de visibilidad del tabique.
2- El sistema almacena el nuevo estado de visibilidad del tabique.

Otros: -



6- Seleccionar una cuerda

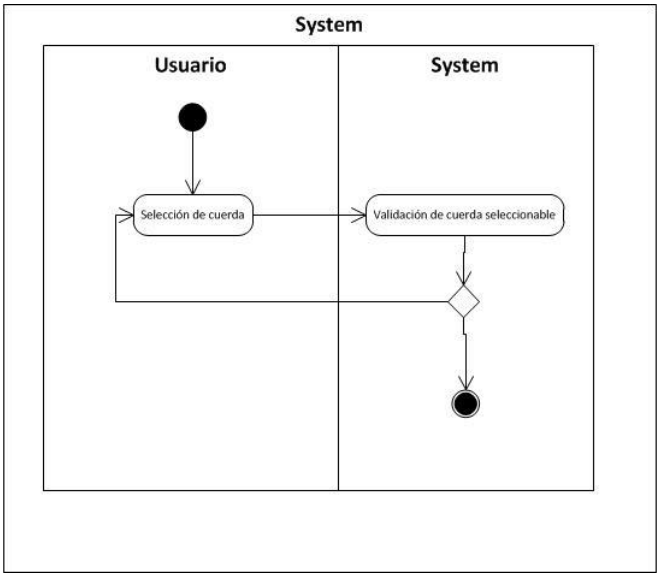
Actor principal: Usuario.

Precondiciones: El usuario debe estar en el modo de edición. No debe haber otras cuerdas seleccionadas o si las hay deben estar vinculadas a un músculo papilar.

Accionador: El usuario desea seleccionar una cuerda determinada.

Escenario principal:
1- El usuario indica al sistema la nueva cuerda seleccionada.
2- El sistema confirma al usuario si puede seleccionar la cuerda.

Otros:
1- El sistema indica al usuario que no puede seleccionar la cuerda.
2- El sistema vuelve al estado anterior.



7- Cambio de punto de referencia

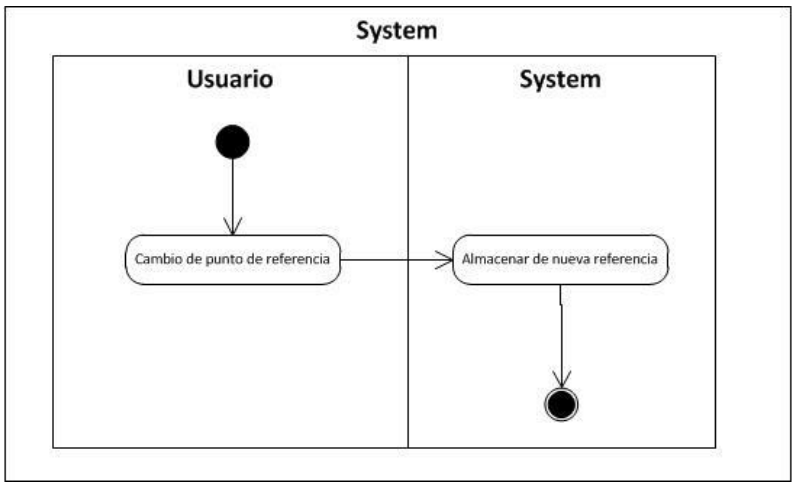
Actor principal: Usuario.

Precondiciones: El usuario debe estar en el modo de edición, simulación o inspección.

Accionador: El usuario desea cambiar el punto de objetivo de la cámara para inspeccionar distintas partes de la válvula mitral.

Escenario principal:
1- El usuario indica al sistema el nuevo punto de referencia.
2- El sistema almacena el nuevo punto de referencia de la cámara.

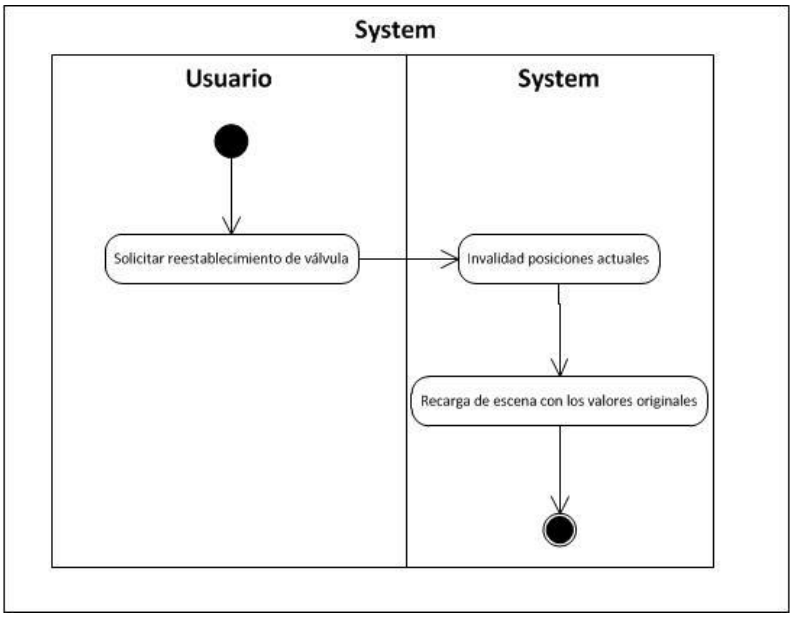
Otros: -



8- Reestablecer configuración original

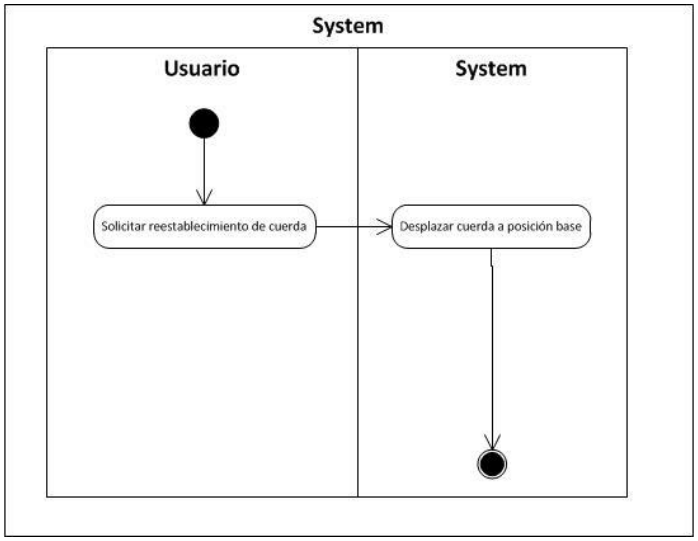
Actor principal: Usuario
Precondiciones: -
Accionador: El usuario desea reestablecer la configuración inicial de la válvula.
Escenario principal:
1- El usuario indica al sistema que desea reestablecer la configuración de la válvula.
2- El sistema falsea las posiciones actuales y carga las originales.

Otros: -



9- Reestablecer posición de una cuerda

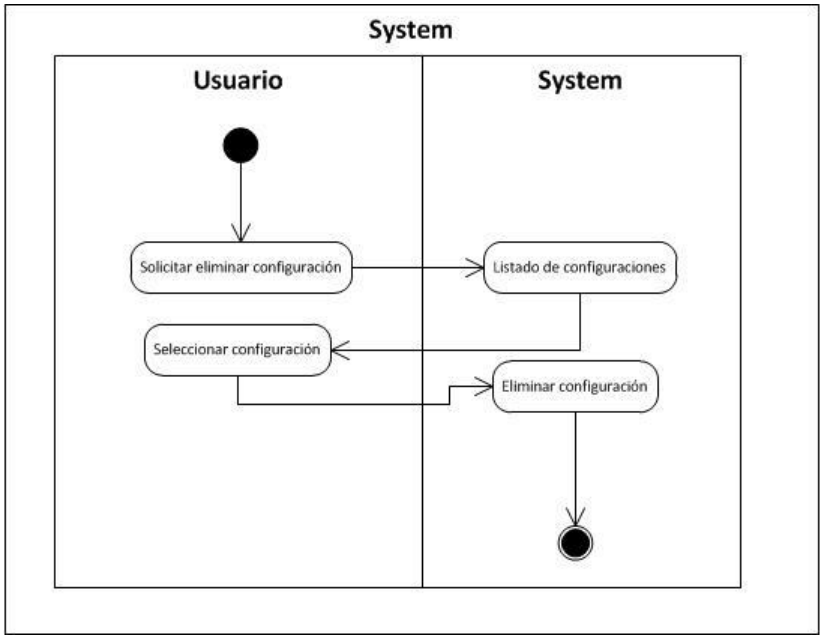
Actor principal: Usuario
Precondiciones: El Usuario tiene una cuerda seleccionada.
Accionador: El usuario desea que la cuerda vuelva a su posición original.
Escenario principal:
1- El usuario indica al sistema que desea reestablecer una cuerda determinada.
2- El sistema desplaza la cuerda a su posición base.
Otros: -



10- Borrar configuración guardada

Actor principal: Usuario
Precondiciones: Existen partidas guardadas
Accionador: El usuario desea eliminar una configuración.
Escenario principal:
1- El usuario indica al sistema que desea eliminar una configuración
2- El sistema indica que configuraciones existen
3- El usuario indica que configuración existente desea eliminar
4- El sistema eliminar la configuración.

Otros:-



11- Guardar configuración

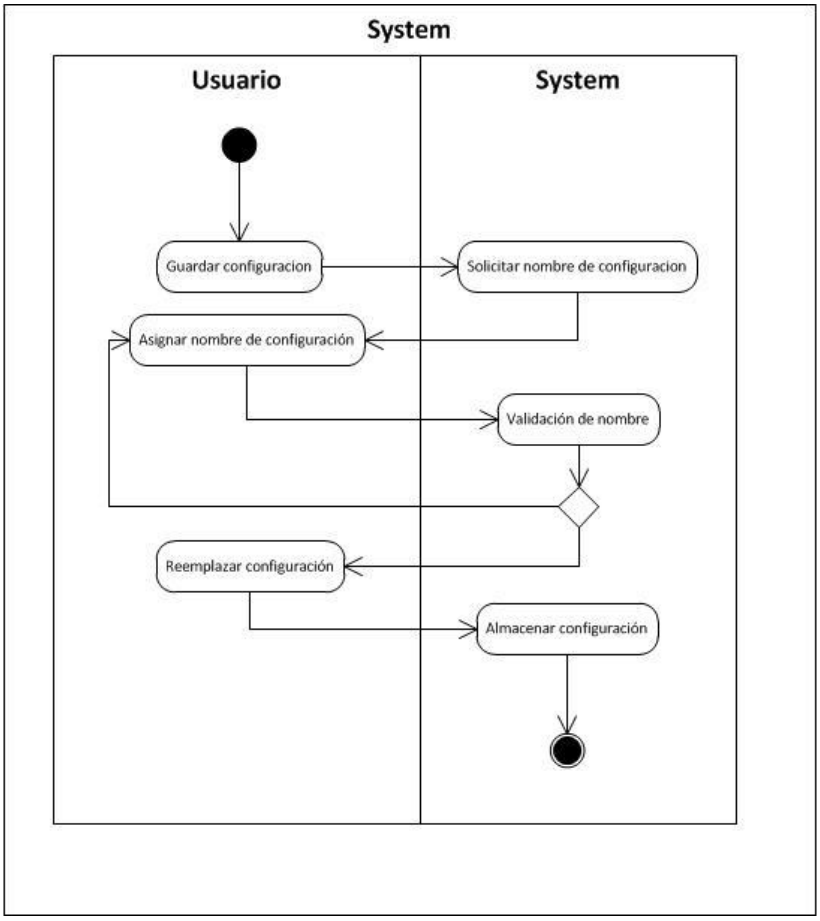
Actor principal: Usuario.

Precondiciones: El usuario debe estar en el modo de edición.

Accionador: El usuario desea guardar una configuración de cuerdas determinadas.

Escenario principal:
1- El usuario indica al sistema la configuración de cuerdas que quiere guardar.
2- El sistema solicita al usuario un nombre de configuración.
3- El usuario indica al sistema el nombre de la configuración de cuerdas.
4- El sistema almacena la configuración con el nombre asignado.

Otros:
5- El sistema indica que ya hay una configuración con ese nombre.
6- El usuario cambia de nombre a la nueva configuración
7- El usuario reemplaza la versión anteriormente guardada.



12- Cargar configuración

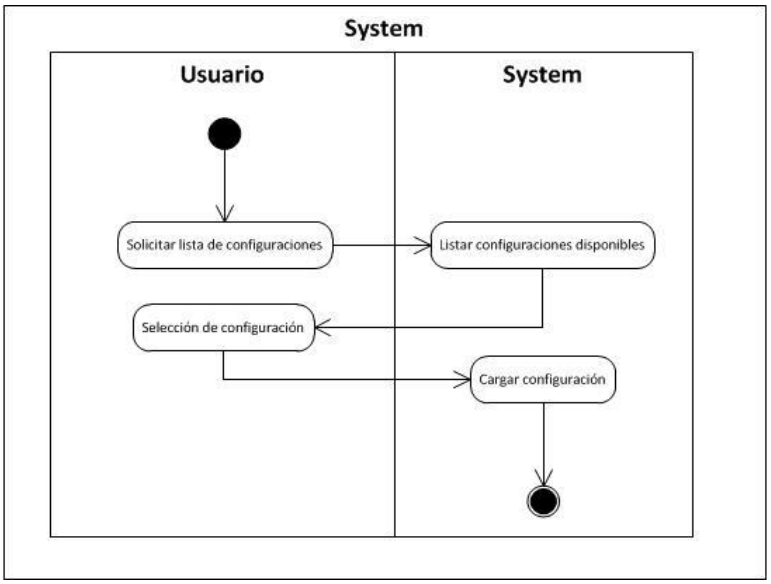
Actor principal: Usuario.

Precondiciones: El usuario debe estar en el modo de edición.

Accionador: El usuario desea cargar una configuración de cuerdas almacenada previamente.

Escenario principal:
1- El usuario solicita la lista de configuraciones almacenadas.
2- El sistema indica al usuario las configuraciones guardadas disponibles.
3- El usuario elige una configuración determinada.
4- El sistema carga la configuración seleccionada.

Otros: -



13- Mostrar panel de controles del juego

Actor principal: Usuario.

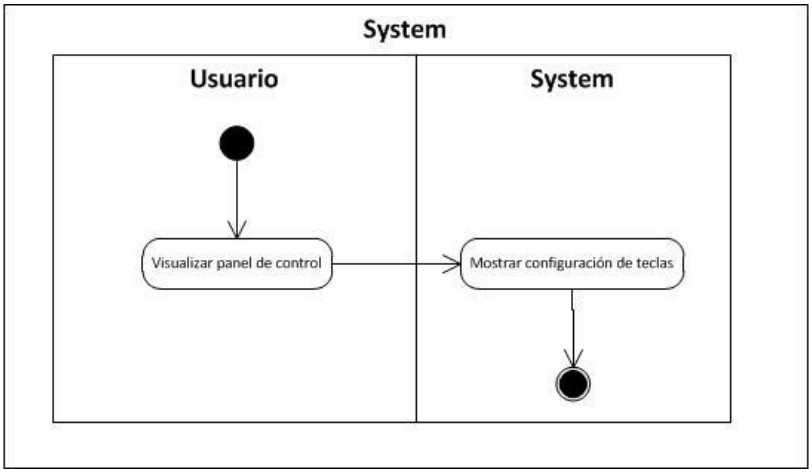
Precondiciones: El usuario debe estar en el modo de edición.

Accionador: El usuario desea informarse de cómo utilizar la aplicación

Escenario principal:

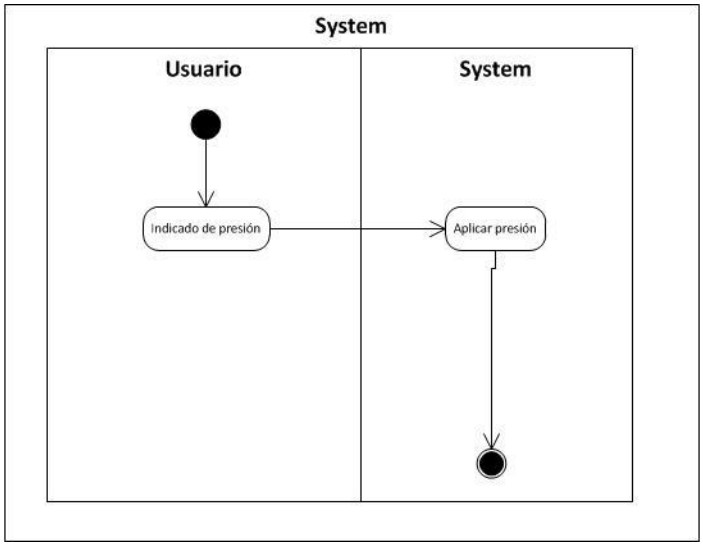
1- El usuario indica al sistema que quiere visualizar las teclas de control de la aplicación.
2- El sistema muestra al usuario la información solicitada.

Otros: -



14- Cambiar presión ventricular

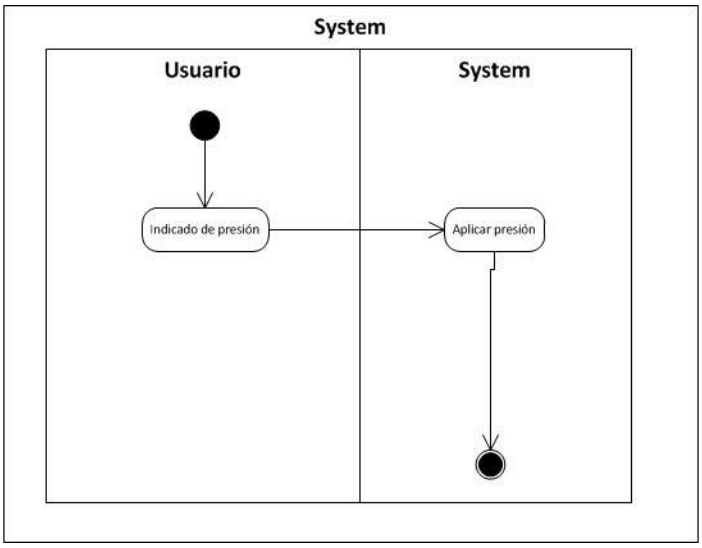
Actor principal: Usuario
Precondiciones: La escena es el modo simulación
Accionador: El usuario desea modificar la presión ventricular
Escenario principal:
1- El usuario indica la presión deseada
2- El sistema aplica la presión indicada
Otros: -



15- Cambiar presión auricular

Actor principal: Usuario
Precondiciones: La escena es el modo simulación
Accionador: El usuario desea modificar la presión auricular
Escenario principal
1- El usuario indica la presión deseada
2- El sistema aplica la presión indicada

Otros: -



6.2 Diseño de la aplicación

Los siguientes apartados describen la estructura interna del proyecto así como el diseño pensado para la implementación del proyecto, la justificación de dicho diseño y que criterios de la ingeniería del software han tenido más prioridad a la hora de la creación del programa.

6.2.1 Integración con BGE

BGE dispone de una serie de componentes que se pueden utilizar para la integración de código de usuario con una aplicación desarrollada bajo esta plataforma. Para este proyecto se han utilizado los módulos de *constrainsts*, que se utiliza para la creación de anclajes físicos entre objetos. Se ha utilizado el módulo *events*, que gestiona las interacciones de teclado y ratón y se ha utilizado el módulo *logic*. Este último componente es el más importante de todos ya que es a través de él como se puede gestionar las escenas de la aplicación y los objetos que las componen. Es posible agregar instancias de objetos a *logic* y utilizar los métodos públicos de los mismos en cualquier momento. Además *logic* tiene un una estructura de datos llamada *globalDict*, que como indica su nombre se trata de un diccionario. La principal propiedad de esta estructura de datos es que es accesible desde el contexto de diversas escenas e incluso lo es tras reiniciar una escena. Esto es porque las escenas se comportan como subprocesos.

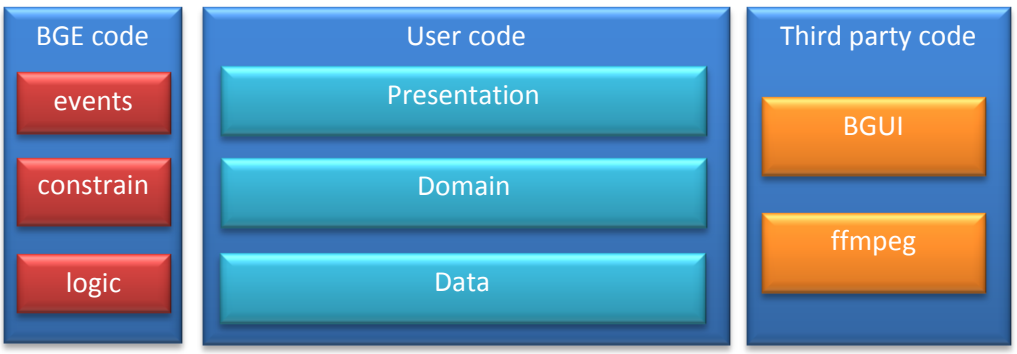


Figura 33 Estructura de componentes

6.2.2 Dominio

La capa de Dominio de la aplicación representa tres componentes diferentes. Todos ellos agregados al componente *logic* de *Blender.* Cada uno de estos componentes representa un objeto distinto de *Blender* como se puede ver en la siguiente figura.

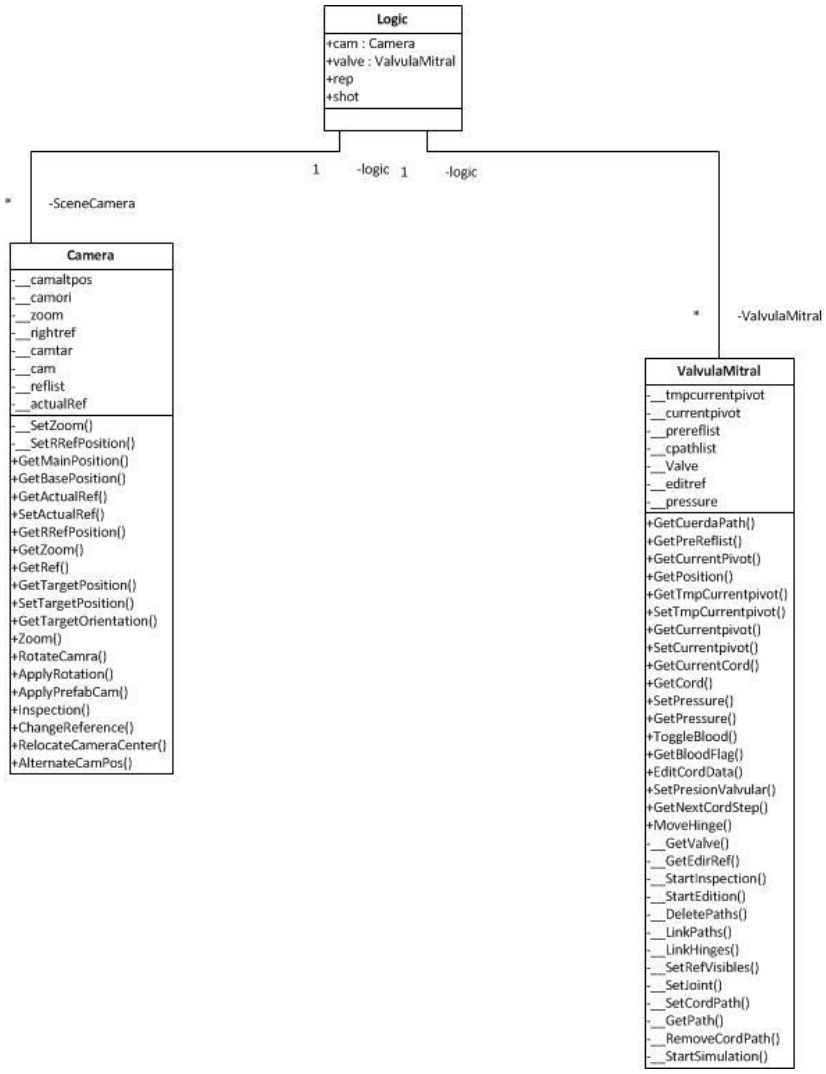


Figura 34: Capa de Dominio

6.2.2.1 Cámara

La cámara es el componente que proporciona la interacción con el usuario. La entidad cámara está representada por el objeto de *Blender Camera,* pero cuyo comportamiento está implementado dentro de la clase Camera de la capa de Dominio.

Se utilizan dos tipos de cámara. La primera es una cámara estática que solo aparece en la escena denominada *MainMenu* que captura la interacción para crear capturas de pantalla y en cuyo viewport se representa el Menú denominado *MainMenu*. El segundo tipo de cámara, es una cámara dividida en tres componentes.

El primer componente es la *cameraTarget*, este representa el punto objetivo de la cámara, además al estar la cámara *parentizada* dentro de este objeto, las acciones realizadas sobre él afectarán al final a la cámara. Esto es esencial para conseguir que la cámara pueda rotar alrededor de un punto central o desplazar el objetivo para que la cámara siga un punto concreto de la escena.

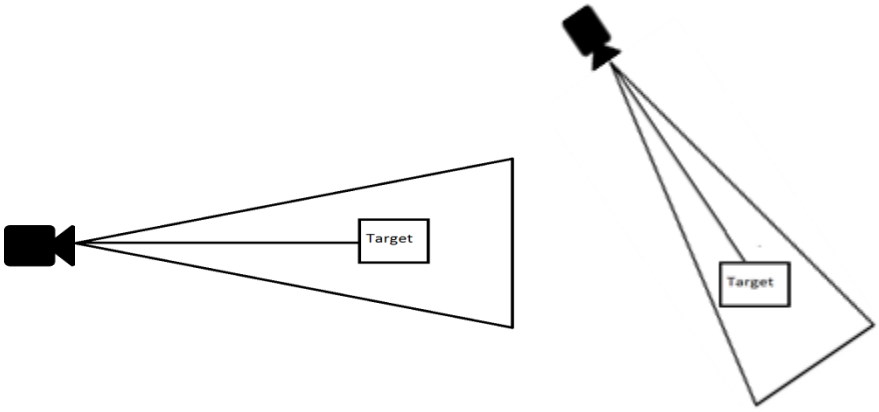


Figura 35: Rotación aplicada a target

El segundo componente de la cámara es el *rightref*, que funciona de forma similar al vector up de la cámara e indica cual es el vector de izquierda a derecha, este vector de referencia sirve para guiar los movimientos de los objetos en relación a la posición de la cámara.

En tercer lugar tenemos a la cámara en sí misma y representa al objeto de *Blender* de tipo cámara, objeto del cual obtenemos los parámetros necesarios para modificar el zoom y para detectar las interacciones de teclado y ratón. Esta cámara es una cámara de tipo ortogonal que a diferencia de una cámara perspectiva su volumen de visión tiene el aspecto de un prisma en vez de una pirámide truncada.

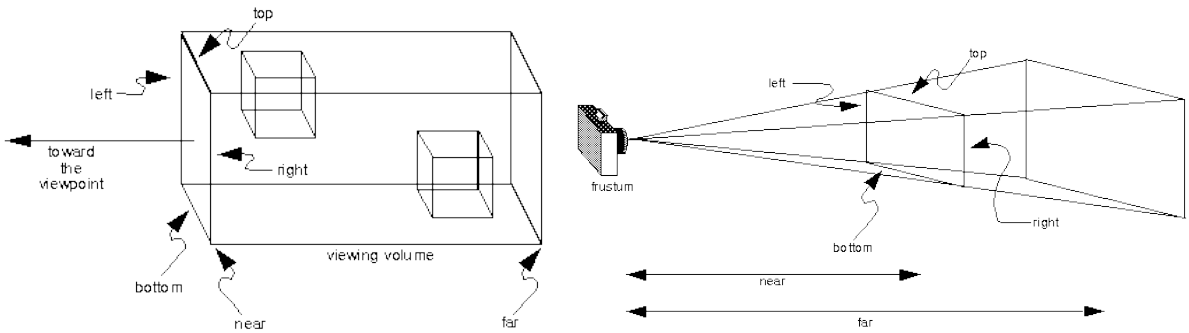


Figura 36: Proyección ortogonal vs perspectiva [4]

En la cámara ortogonal hay dos formas de representar el zoom, la primera es alejar o acercar la posición de la cámara en relación al *cameraTarget* o modificando el factor de escalado que incrementa o reduce el tamaño del área de visión de la cámara haciendo que los objetos se vean de mayor o menor tamaño.

En cuanto a la clase Cámara además de los componentes anteriormente mencionados almacena una lista de referencias que representan distintos puntos de vista predefinidos. Estos son utilizados para enfocar distintos puntos de la escena pulsando una sola tecla para facilitar la inspección de la válvula. Por este motivo la clase cámara cuenta con un atributo que indica a que punto de referencia está apuntando actualmente.

Finalmente la clase cámara cuenta con un atributo que almacena la posición original de la cámara, utilizado cuando se quiere volver a una posición y además usado como punto de referencia para el cálculo de las rotaciones que se aplican para conseguir lo que se denomina como cámaras predefinidas.

6.2.2.2 Válvula mitral

La válvula mitral está representada por la clase *VálvulaMitral*. Como ya se ha explicado en el capítulo anterior,el usuario no interacciona directamente con la válvula, sino que lo hace a través de las esferas de referencia. El objeto de dominio que representa la válvula almacena que referencia ha sido seleccionada, también almacena la lista de referencias posibles y el objeto válvula en sí mismo. Finalmente esta clase también registra el valor de la presión interna utilizada para el modo simulación.

6.2.3 Presentación

El programa está estructurado en 4 escenas interconectadas tal y como se puede ver en la figura 38. El Menú principal es el punto de entrada a la aplicación y a través de la cual el usuario puede acceder a los distintos modos de la aplicación. El usuario es capaz de observar la válvula estática en el modo inspección, editar su configuración y simularla en cualquier orden que desee.

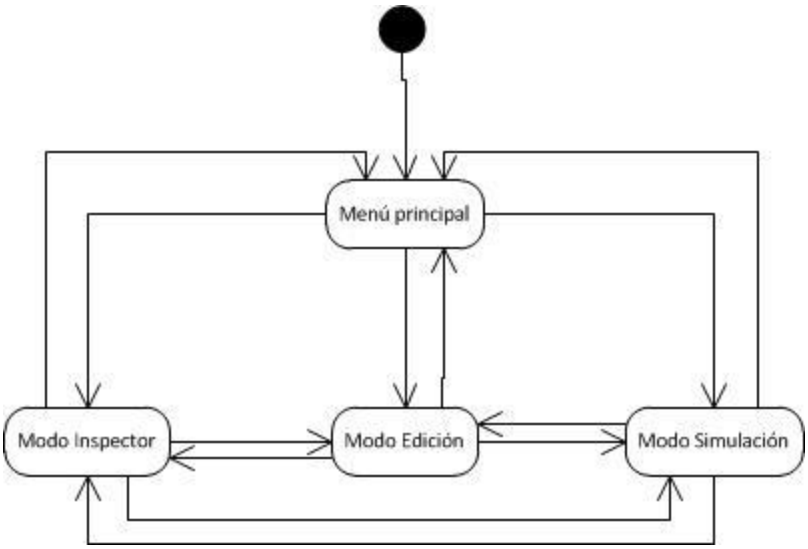


Figura 37: Storyboard

Cada escena tiene asociado un menú, en la aplicación hay dos tipos distintos de menú, el *MainMenu* y el *TopBarMenu*. *MainMenu* es un Menú formado por una parrilla de botones ubicados en la parte izquierda de la pantalla que gestiona la interconexión de las escenas y *TopBarMenu* es un panel ubicado en la parte superior de la pantalla que controla parte de las funciones de cámara, de la creación de repeticiones y el cambio de escena. No obstante ambos tipos de menú comparten algunas funciones en común y algunos componentes repetidos tales como el menú de confirmación de acción, el intercambio de escena y la finalización de la aplicación. Por este motivo y por la potencial aparición de futuros atributos o métodos comunes, ambos tipos de menú son una especialización de una clase denominada BGUIMenu.

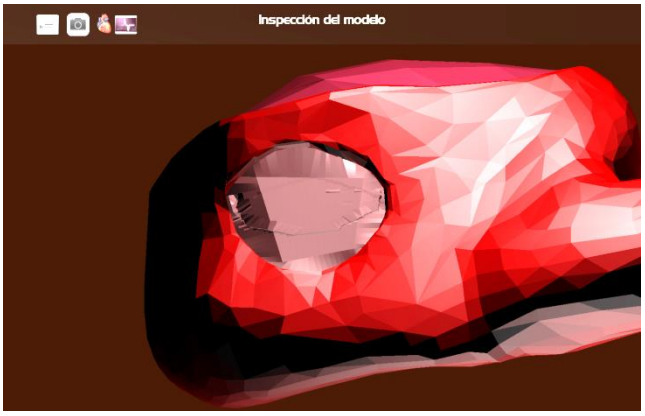


Figura 38: TopBarMenu

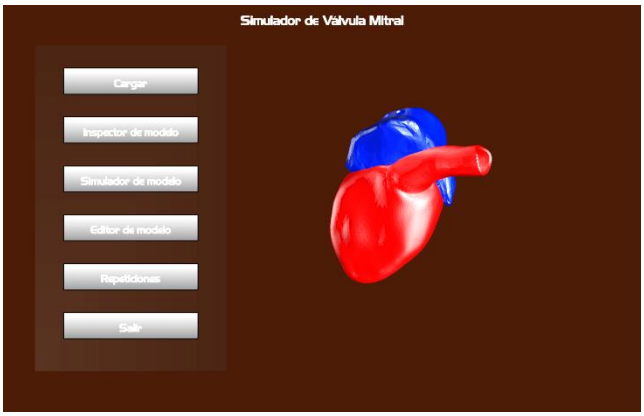


Figura 39:MainMenu

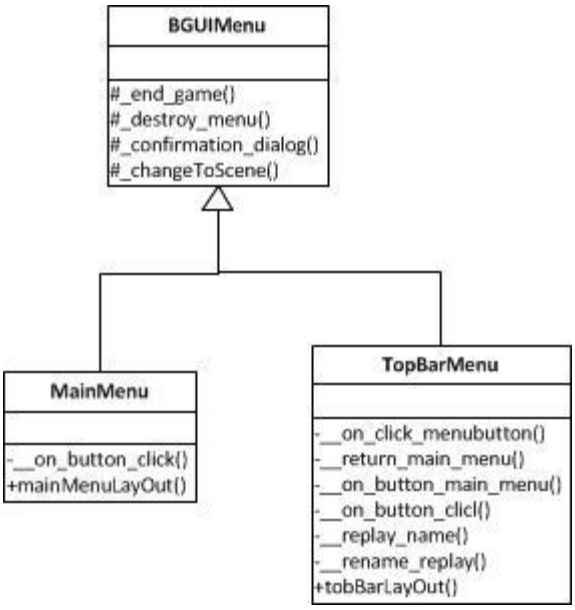


Figura 40: Clases Capa de presentación

6.2.4 Persistencia

La capa de persistencia está conformada por aquellos controladores que gestionan los ficheros que genera y lee la aplicación durante su ejecución. De esta forma tenemos un controlador para la gestión de capturas de pantalla, otro para las repeticiones y finalmente un controlador para la gestión de configuraciones de cuerdas.

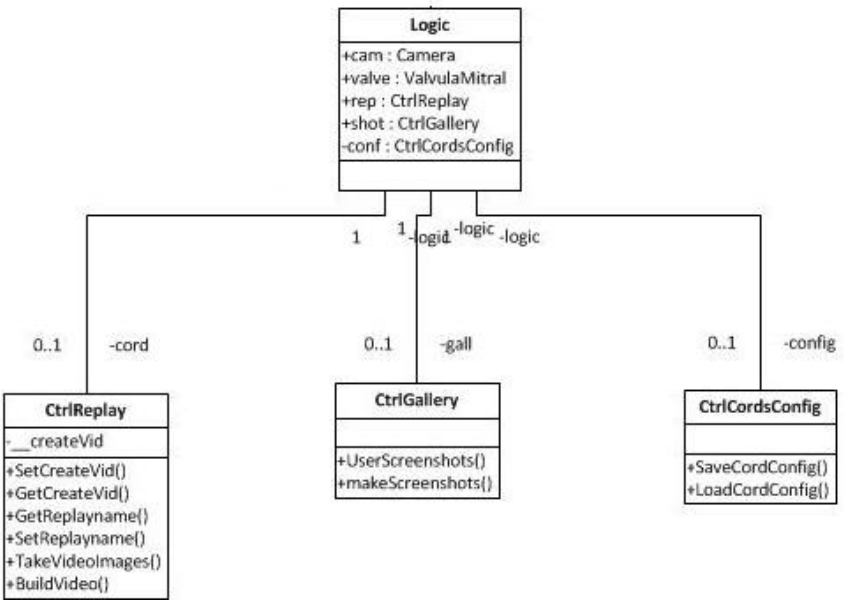


Figura 41: Capa de persitencia

Mediante el controlador CtrlReplay se puede crear y renombrar videos generados mediante el módulo ffmpeg que se ejecuta como un subproceso una vez se han realizado todas las capturas requeridas.

Existen dos métodos para la captura de imágenes, la primera es la que utiliza el usuario y por tanto está vinculada a un sensor de teclado y la segunda es la que utiliza el capturador de vídeos para generar nuevas repeticiones.

Finalmente mediante CtrlCordsConfig almacenamos y cargamos configuraciones específicas de cuerdas, esto se realiza almacenando los datos que se encuentran en la estructura de datos de Logic denominada glodalDict y posteriormente cargándola de nuevo cuando el usuario la necesita.

6.3 Implementación

El siguiente capítulo describe la implementación del diseño descrito en el capítulo anterior. Para tal objetivo se describirá en primer lugar la estructura de ficheros utilizada, el uso de librerías externas como BFUI y ffmpeg y finalmente se procederá a describir la utilización de los logic bricks y los scripts de acceso público y cómo estos se relacionan con las clases de presentación dominio y persistencia descritos anteriormente.

6.3.1 Estructura de ficheros

Para la implementación de código externo se escogió la creación de ficheros externos, editables con cualquier tipo de editor de texto que de soporte a lenguaje Python. Dado que el programa sigue la arquitectura en 3 capas, los ficheros de código están separados en packages de Python demoninados Domain, Data y Presentation respectivamente. Todos ellos agrupados dentro de un package llamado scripts.

El programa utiliza librerías externas para la implementación de una GUI y para la creación de repeticiones. Además se utilizan carpetas para depositar los ficheros de las capturas de pantalla, de las repeticiones y de las configuraciones guardadas. Para tal objetivo existen las carpetas correspondientes en la raíz de la aplicación a la misma altura que la carpeta de scripts.

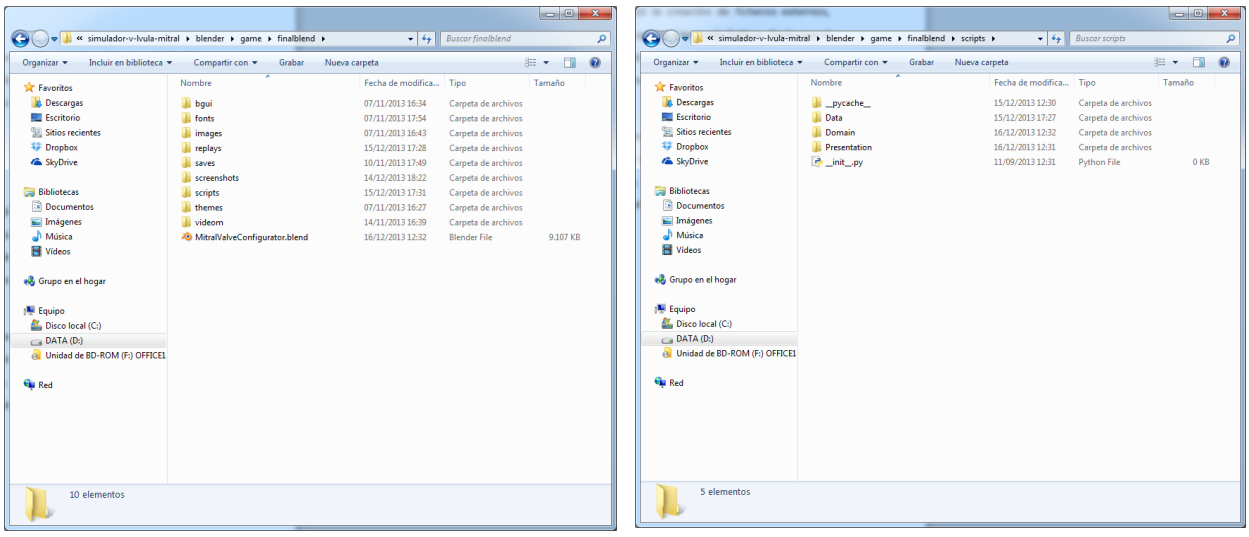


Figura 42: Estructura de ficheros

Hay que tener en cuenta que al generar ejecutables externos, este no tiene en cuenta las carpetas generadas por el usuario, con lo que al crear el ejecutable hay que copiar toda la estructura de carpetas.

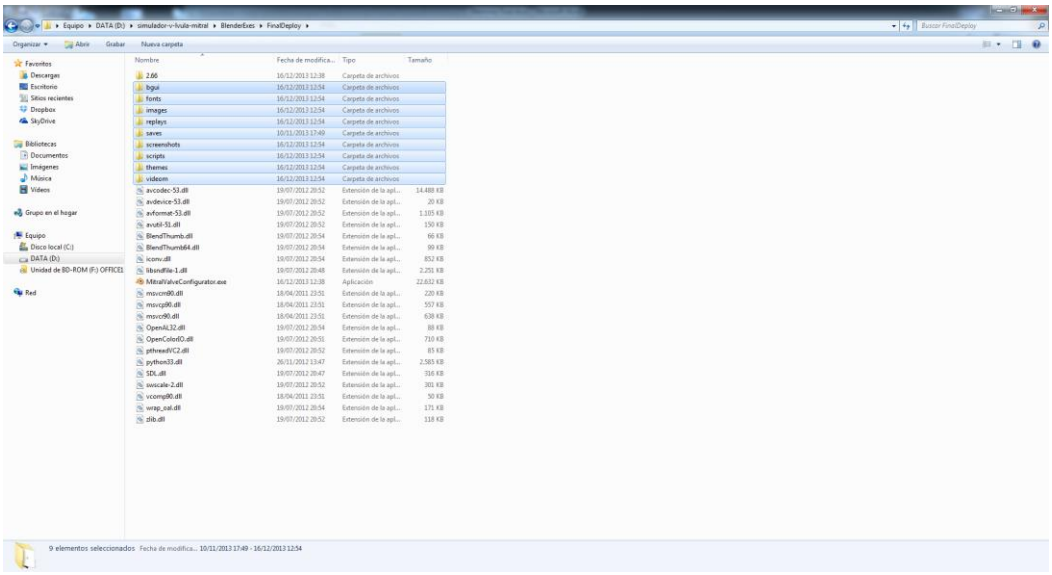


Figura 43: Ejecutable

6.3.2 BGUI

Graphic User Interface (BGUI): En cada escena existe un objeto denominado SceneMgr que gestiona la inicialización de la escena así como la interfaz gráfica de la misma. Esto quiere decir que en función de la escena que se esté ejecutando se visualizará el tipo de menú correspondiente.

Mediante BGUI es posible crear distintos tipos de componentes separados en contenedores. Cada contenedor o frame está a su vez alojado dentro de otro contenedor hasta llegar a un contenedor base que ocupa todo el viewport de la pantalla.

Con el objetivo de potenciar la mantenibilidad de la aplicación BGUI permite crear distintas configuraciones o temas gráficos para las interfaces. A la hora de crear un nuevo menú o layout basta con indicar la ruta donde se encuentra ubicada la configuración del layout.

En la aplicación actual existe una carpeta de temas llamada themes, dentro de la cual existe una carpeta para cada nuevo theme creado, en este caso el tema se llama default. En la siguiente figura se puede ver como se accede a dicho tema.

```
181
182 def topBarLayOut(cont):
183     own = cont.owner
184     mouse = g.mouse
185
186     if "sys" not in own:
187         # Create our system and show the mouse
188         own["sys"] = bgui.bge_utils.System("themes/default")
189         own["sys"].load_layout(TopBarMenu,None)
190     else:
191         own["sys"].run()
192
193
```

Figura 44: Asignación de un tema

Una vez asignado el tema al *layout* utilizado, hay que etiquetar todos los componentes creados en la interfaz con un atributo denominado *subtheme*. Dicho etiquetado asocia dicho componente un una regla de la hoja de estilos.

```
self.iconbutton3 = bgui.ImageButton(self.win, sub_theme="Editor",
                                     size=[0.035, 0.14], pos=[0.14, 0.05])
self.iconbutton3.on_click = self.__on_button_click

self.iconbutton4 = bgui.ImageButton(self.win, sub_theme="Simulator",
                                     size=[0.035, 0.14], pos=[0.175, 0.05])

self.iconbutton4.on_click = self.__on_button_click
```

Figura 45: Asignación de subtheme

Dentro de la carpeta del theme creado debe existir un fichero llamado theme.cfg. Dicho fichero contiene la lista de reglas asociados a los subthemes asignado s los componentes de la interfaz gráfica. Estas reglas pueden configurar desde el tipo de fuente de las letras, el color del fondo, etc, de una forma muy similar a una hoja CSS utilizada para el desarrollo de páginas web.

```
FillColor3=0.62, 0.0, 0.02, 1.0
FillColor4=0.62, 0.0, 0.02, 1.0
BGColor1=0.0, 0.0, 0.0, 1.0
BGColor2=0.0, 0.0, 0.0, 1.0
BGColor3=0.15, 0.15, 0.15, 1.0
BGColor4=0.15, 0.15, 0.15, 1.0
BorderSize=1
BorderColor=0.0, 0.0, 0.0, 1.0

[ImageButton:Audio]
DefaultImage=img:audio.png, 0, 0, 0.5, 1
Default2Image=img:audio.png, 0.5, 0, 0.5, 1
HoverImage=img:audio.png, 0.5, 0, 0.5, 1

[ImageButton:Menu]
DefaultImage=img:menu.png, 0, 0, 1, 1

[ImageButton:Camara]
DefaultImage=img:camara.png, 0, 0, 1, 1

[ImageButton:Simulator]
DefaultImage=img:simulate.png, 0, 0, 1, 1

[ImageButton:Editor]
DefaultImage=img:edit.png, 0, 0, 1, 1

[ImageButton:Inspector]
DefaultImage=img:inspector.png, 0, 0, 1, 1

[ImageButton:Record]
DefaultImage=img:record.png, 0, 0, 1, 1
Default2Image=img:stop.png, 0, 0, 1, 1
```

Figura 46: Reglas de theme.cfg

Finalmente, a cada componente que requiera interactividad de la GUI, por ejemplo un botón, como se puede ver en la figura 46. Se le puede asociar una función que responda cuando es activado. Esto permite asignar los métodos necesarios para configurar los casos de uso descritos anteriormente.

6.3.3 ffmpeg

Para poder implementar la captura de vídeos de repeticiones se utilizó un compresor de video externo llamado ffmpeg. Esta aplicación de consola permite manipular y crear vídeos de distintas formas. Para la implementación del proyecto se utilizó la captura de imágenes que ofrece el módulo Rasterizer de Blender. Dichas imágenes se almacenan de forma temporal en una carpeta y cuando el usuario decide detener la captura de imágenes, se lanza un subproceso utilizando el método call de Python para construir el vídeo. A dicha repetición se le asigna un nombre por defecto que posteriormente el usuario puede modificar. El ejemplo de su utilización se puede observar en la siguiente imagen.

```
def BuildVideo(self,ruta):
    call(".\\videom\\bin\\ffmpeg.exe -r 5 -i "+ruta+"\\%d.png -s 1920x1080  .\\replays\\"+self.__replayname+".avi")
    shutil.rmtree(ruta)
```

Figura 47:Construcción de video

6.3.4 scripts públicos

Los ficheros de scripts ya mencionados contienen los métodos definidos para ser accesibles desde los Logic bricks y ofrecen una capa de abstracción para evitar acoplamientos entre capas y respecto al código de BGE. Como se puede ver en la siguiente imagen, se enlazan los sensores con los controladores para crear una acción determinada.

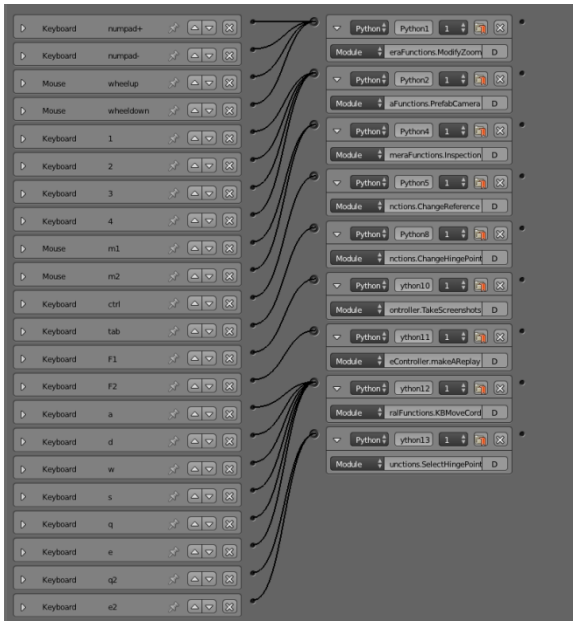


Figura 48: Logic Bricks

Estos ficheros están separados por coherencia y engloban todos los casos de uso mencionados, la implementación de sus métodos se detallará en el anexo de implementación.

CameraFunctions: Este fichero contiene los métodos necesarios para interacción entre el usuario y la cámara y define las siguientes funciones: ModifyZoom, RotateCamera, PrefabCamera, SetPrefabCamera, Inspections, ChangeReference.

ImageVideoFunctions: Este fichero contiene los métodos de captura de imagen y vídeo. Además de los métodos que permiten la interacción entre la capa de presentación y la capa de persistencia para la modificación de nombres de fichero,etc. Los métodos que define son TakeScreenshots, makeAReplay, ToggleReplay, SetReplayName.

ValvulaMitralFunctions: Este conjunto de scripts define las funciones que interactúan directamente con la VálvulaMitral, como podría ser modificar sus anclajes, seleccionar una cuerda,etc. Las funciones que define este archivo son: SelectCord, MouseMoveCord, KBMoveCord, SelectHingePoint, ChangeHingePoint, RestartWithNewConfig, CanFix, QuitarTapa y SetPresionValvular.

GlobalDictFunctions: GlobalDict es el diccionario que almacena datos accesibles entre distintas escenas. Este fichero contiene algunos métodos que permiten el guardado de información en dicha estructura de datos así como la creación de un fichero con una configuración de cuerdas determinada y la carga de una configuración almacenada con anterioridad. Los métodos definidos en este archivo son: StoreInGlobalDict, SaveGlobalDictToFile, LoadFileToGlobalDict.

BGEObjectsFunctions: Este fichero contiene métodos que no representan una interacción directa con el usuario pero si con la escena ya que su contexto es la gestión de objetos de escena. En este fichero se pueden encontrar los métodos para añadir y eliminar objetos de una escena, para obtener un objeto,etc. Los métodos definidos son los siguientes: GetOwner, GetSensor, GetObjects, AddObject, EndObject, ChangeColor.

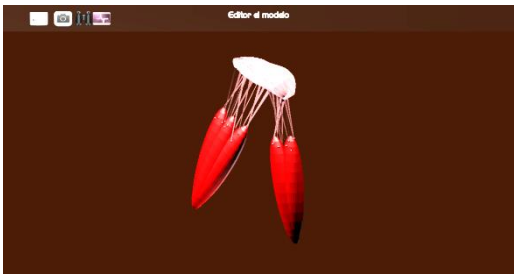
VecFunctions: Este fichero auxiliar contiene métodos para operar sobre vectores. Sus métodos son los siguientes: GetVector,GetDistance.

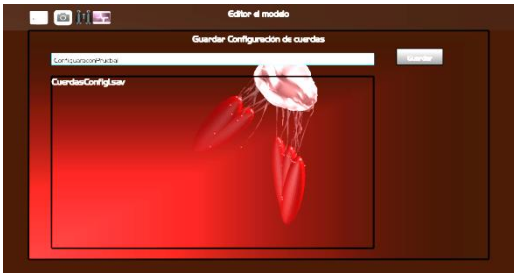
SceneController: Este fichero cumple las veces de controlador y de administrador de las escenas. Mediante sus métodos se realizan las inicializaciones de las escenas. Los métodos que contiene son los siguientes: InitProgram, StartScene, SetGravity, GetMouseSensor, isMouseMoving, GetMousePosition, ShowMouse.

6.4 Resultados

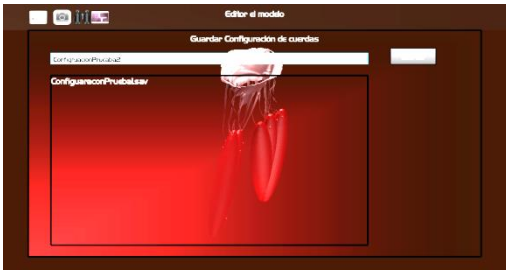
A continuación podremos ver una serie de figuras que representan los pasos de una ejecución que cubre algunos de los casos de usos ya mencionados.

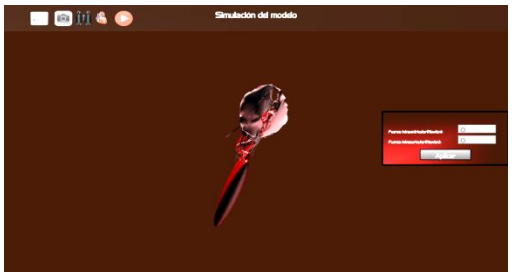
1. Los pasos a seguir serán:
2. Abrir el modo inspector.
3. Entrar en el modo editor.
4. Crear una configuración de cuerdas cualquiera y guardarla.
5. Crear una configuración de cuerdas distinta y guardarla también.
6. Entrar en el modo de simulación.
7. Pulsar reestablecer para cargar la configuración base.
8. Aplicar una fuerza auricular para forzar la apertura de la válvula en el sentido de la sangre.
9. Aplicar una fuerza ventricular igual y contraria al sentido de la sangre.
10. Cargar una de las configuraciones guardadas con anterioridad.
11. Cargar otra de las configuraciones guardadas.
12. Crear una repetición
13. Borrar todas las configuraciones creadas.



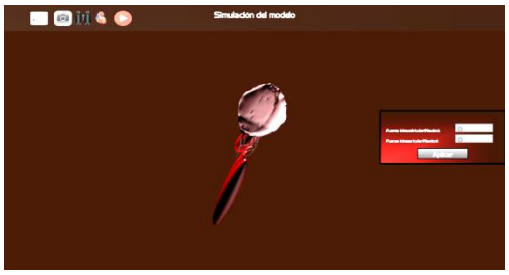



Editor el modelo


Editor el modelo
Guardar Configuración de cuerdas
ConfiguarasconPrueba1.sav


Simulación del modelo

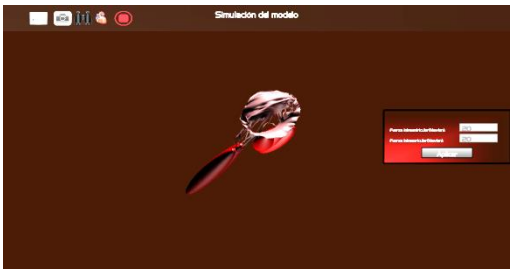

Simulación del modelo

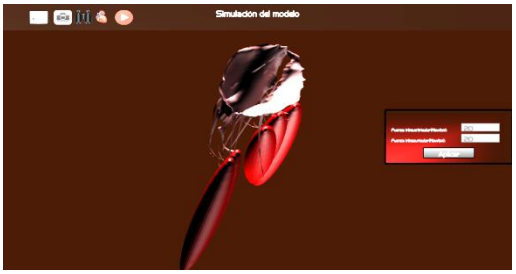

Simulación del modelo


Simulación del modelo
Cargar Configuración de cuerdas
ConfiguracionPrueba2.sav
ConfiguarasconPrueba1.sav
Partida Seleccionada: ConfiguarasconPrueba1.sav


Simulación del modelo


Simulación del modelo


Simulación del modelo


Simulación del modelo

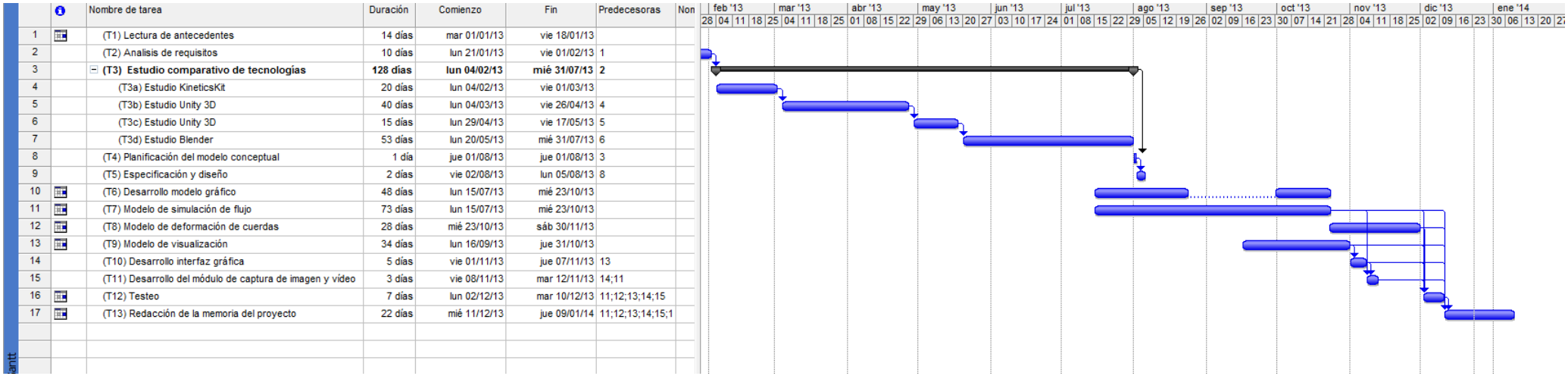
7. Planificación y análisis económico

7.1 Planificación final

El proyecto comenzó a realizarse el 1/1/13 y la fecha final del mismo es el día 22/1/14. El desarrollador del proyecto ha compaginado la duración de todo el proyecto con su actividad laboral. Además durante los primeros 4 meses del mismo el proyecto convivió con las últimas asignaturas del bloque ciclo de asignaturas optativas.

ID	Descripción	Rol	Días	H/Día	Total
E1	**Inicial**		**24**		**48**
T1	Lectura de antecedentes	R2	14	2	28
T2	Análisis de requisitos	R2	10	2	20
E2	**Investigación**		**128**		**320**
T3	Estudio comparativo de tecnologías				320
T3a	Estudio KineticsKit	R2	20	2.5	50
T3b	Estudio Panda3D	R2	40	2.5	100
T3c	Estudio Unity 3D	R2	15	2.5	37.5
T3d	Estudio Blender	R2	53	2.5	132.5
E3	**Desarrollo**		**212**		**548**
T4	Planificación del modelo conceptual	R1	1	3.5	3.5
T5	Especificación y diseño	R1	2	3.5	7
T6	Desarrollo de modelo gráfico	R3	48	2.5	120
T7	Desarrollo de modelo de simulación de flujo	R3	73	2.5	182.5
T8	Desarrollo de modelo de deformación de cuerdas	R3	28	3	84
T9	Desarrollo del modelo de visualización	R3	34	3	102
T10	Desarrollo interfaz gráfica	R3	5	3.5	17.5
T11	Desarrollo del módulo de captura de imagen y vídeo	R3	3	3.5	10.5
T12	Testeo de la aplicación	R4	7	3	21
E4	**Documentación**		**22**		**66**
T13	Redacción de la memoria del proyecto	R1	22	3	66
				Total	982

7.2 Diagrama de Gantt



	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	(T1) Lectura de antecedentes	14 días	mar 01/01/13	vie 18/01/13	
2	(T2) Analisis de requisitos	10 días	lun 21/01/13	vie 01/02/13	1
3	**(T3) Estudio comparativo de tecnologías**	**128 días**	**lun 04/02/13**	**mié 31/07/13**	**2**
4	(T3a) Estudio KineticsKit	20 días	lun 04/02/13	vie 01/03/13	
5	(T3b) Estudio Unity 3D	40 días	lun 04/03/13	vie 26/04/13	4
6	(T3c) Estudio Unity 3D	15 días	lun 29/04/13	vie 17/05/13	5
7	(T3d) Estudio Blender	53 días	lun 20/05/13	mié 31/07/13	6
8	(T4) Planificación del modelo conceptual	1 día	jue 01/08/13	jue 01/08/13	3
9	(T5) Especificación y diseño	2 días	vie 02/08/13	lun 05/08/13	8
10	(T6) Desarrollo modelo gráfico	48 días	lun 15/07/13	mié 23/10/13	
11	(T7) Modelo de simulación de flujo	73 días	lun 15/07/13	mié 23/10/13	
12	(T8) Modelo de deformación de cuerdas	28 días	mié 23/10/13	sáb 30/11/13	
13	(T9) Modelo de visualización	34 días	lun 16/09/13	jue 31/10/13	
14	(T10) Desarrollo interfaz gráfica	5 días	vie 01/11/13	jue 07/11/13	13
15	(T11) Desarrollo del módulo de captura de imagen y vídeo	3 días	vie 08/11/13	mar 12/11/13	14;11
16	(T12) Testeo	7 días	lun 02/12/13	mar 10/12/13	11;12;13;14;15
17	(T13) Redacción de la memoria del proyecto	22 días	mié 11/12/13	jue 09/01/14	11;12;13;14;15;1

7.3 Recursos humanos

El desarrollo del proyecto ha requerido la intervención de los cuatro roles tradicionales de un proyecto de software informático. El Jefe de proyecto encargado de la planificación y gestión de recursos, el analista/diseñador que ha realizado las tareas correspondientes al análisis de requisitos y antecedentes, el programador encargado del desarrollo de la aplicación y el tester encargado de la depuración de la aplicación.

Id Rol	Recurso	Salario
R1	Jefe de proyecto	50 €/h
R2	Analista / Diseñador	40 €/h
R3	Programador	30 €/h
R4	Tester	20 €/h

Realizar el análisis económico de este proyecto es complicado ya que gran parte del tiempo dedicado al mismo ha consistido en investigación y búsqueda de herramientas. También se ha tenido en cuenta que dicha búsqueda de tecnologías responde a una voluntad más académica que comercial.

Sin embargo de querer iniciar una actividad comercial basada en los conocimientos adquiridos, el coste del proyecto estará valorado sobre una base de conocimientos ya adquirida y la decisión de escoger una tecnología ya no provendría de un estudio extenso si no del análisis de requerimientos del proyecto.

En este sentido y partiendo también de la hipótesis de querer iniciar una actividad comercial basada en este trabajo, los *stakeholders* del mismo serían los siguientes:

1- Estudiantes de cardiología
2- Cirujanos en activo.
3- Pacientes que se beneficien del entrenamiento recibido por los cirujanos.
4- Empresas de desarrollo de Software especializado en medicina.
5- Hospitales universitarios.

Id Rol	Horas Totales	Salario	Subtotal
R1	10.5	50 €/h	525 €
R2	48	40 €/h	1920 €
R3	516.5	30 €/h	15495 €
R4	21	20 €/h	420 €
		Total	18360 €

7.4 Recursos materiales

Para el desarrollo de la aplicación no se requirieron nuevos materiales ya que asumiendo la antigüedad del equipo superior al periodo usual de amortización de 3 años su coste ha sido de 0 €.

8. Conclusiones y líneas futuras de trabajo

8.1 Conclusiones

El desarrollo de este proyecto ha supuesto la realización de un trabajo previo de búsqueda y de estudio de herramientas totalmente nuevas para el autor. El desarrollo de aplicaciones gráficas, y específicamente el de los videojuegos supone un reto y este proyecto ha servido entre otras cosas para descubrir que afrontar una empresa de este tipo se ha de realizar de una forma distinta a la del desarrollo de aplicaciones de otro tipo. A diferencia de las aplicaciones de escritorio, a las que el desarrollador está acostumbrado, el peso de la capa de presentación es mucho mayor que la de dominio y requiere de un cuidado especial.

El punto central del proyecto ha sido siempre el tratamiento de objetos deformables en tiempo real. Esto supone un coste computacional elevado, funcionalidad que suele quedar fuera de las grandes producciones que en su lugar utilizan deformaciones empaquetadas en animaciones. Sin embargo algunos equipos de desarrollo independientes están empezando a apostar por la deformación física en tiempo real, como es el caso de Space Engineers, distribuido por *Keen Software House*. Este hecho puede suponer un impulso para esta tecnología, sobre todo paras las librerías de físicas actuales que le dan soporte.

El autor del proyecto tiene muy en mente que aún queda mucho camino que recorrer, pero los conocimientos adquiridos durante el desarrollo de esta aplicación son impagables y que si se compara el punto en el que se encuentra el proyecto en este punto con el punto de partida, seguramente alguna de las decisiones tomadas serían distintas.

La valoración personal es en definitiva muy positiva debido al interés del autor en conocer y aprender a utilizar herramientas diseñadas para el modelado en 3D y el desarrollo de videojuegos y que sin duda servirán de base para futuros proyectos que emprenda el desarrollador algunos de los cuales ya tiene en mente.

8.2. Líneas futuras de trabajo

Este proyecto ha alcanzado el punto en el que se pueden simular distintas configuraciones de una válvula mitral, crear repeticiones, capturas de pantalla y estudiar el comportamiento de la válvula colocando las cuerdas en distintas ubicaciones. A partir de este punto se abren diversas posibilidades para el futuro de este proyecto.

- En primer lugar es posible mejorar el realismo de la aplicación mejorando la incidencia y la frecuencia de la presión recibida sobre la superficie velar de la válvula mitral.
- En segundo lugar se abre una línea de trabajo en la posibilidad de añadir jugabilidad a la aplicación, lo cual permitiría pasar de la fase simulador a la fase juego serio. Entre dichos factores se incluye:
    - **Añadir un sistema de puntuaciones.**
    - **Agregar condiciones de medida de calidad**: Esto permitiría calibrar que tan buena o mala es una determinada configuración de las cuerdas advirtiendo de la aparición de prolapsos a largo de la zona de coaptación, etc.
- En tercer lugar se contempla ampliar el abanico de casos tratados por la aplicación. Por ejemplo:
- **Incluir la funcionalidad de la instalación de anillos protésicos semirígidos correctivos** que se instalan en el surco que separa la aurícula y el ventrículo izquierdo para tratar algunas dolencias.

Finalmente pero no menos importante, existe un importante campo de trabajo en la manipulación de SoftBodies. El uso de librerías externas para este apartado proporciona ventajas y desventajas pero principalmente una clara dependencia en función de la tecnología escogida. Por tanto la idea de desarrollar un motor de deformaciones específico que ofrezca algunas opciones como la asignación de distintos valores de elasticidad para diferentes áreas del modelo o la gestión eficiente y usable de puntos de anclaje del modelo deformable, probablemente pueda suponer un proyecto en sí mismo.

9. Anexos

9.1 Anexo 1

9.1.1 Vpython y Kinetics Kit

9.2.1.1 Creación de una Mesh simple deformable

Crear una aplicación *VPython* con *KineticsKit*, equivale a especificar las propiedades de un entorno físico al cual se le añadirán objetos que *VPython* se encargará de renderizar. Con este propósito se crea un objeto de tipo *System* y se le asignan propiedades tales como *timestep*, que es el ratio de tiempo para los *frames*, *gravity*, que es la fuerza de aceleración de la gravedad y *viscosity* que es la viscosidad del escenario. También podemos indicar algunos atributos de la ventana de ejecución accediendo al atributo *display*, esto permite entre otras cosas mostrar u ocultar la ventana de ejecución, asignarle un nombre, etc.

```
1  from KineticsKit import System,Mass,DoubleHelixSpring,CylinderSpring
2  from visual import faces,frame,rate,scene,extrusion
3  from math import pi
4
5
6
7  def setScene():
8      rate = 30
9      s =  System(timestep=1./rate, gravity=9.84, viscosity=0.04)
10     s.display.title = "Filaments4"
11     s.display.visible = True
12     return s
13
14
15
16
17
18
19
```

Figura 49: Uso de KineticsKit

Una vez definida una función que crea el entorno físico, solo hace falta llamar la funcionalidad desde un *main*, como en cualquier aplicación de Python y comenzar a añadir elementos.

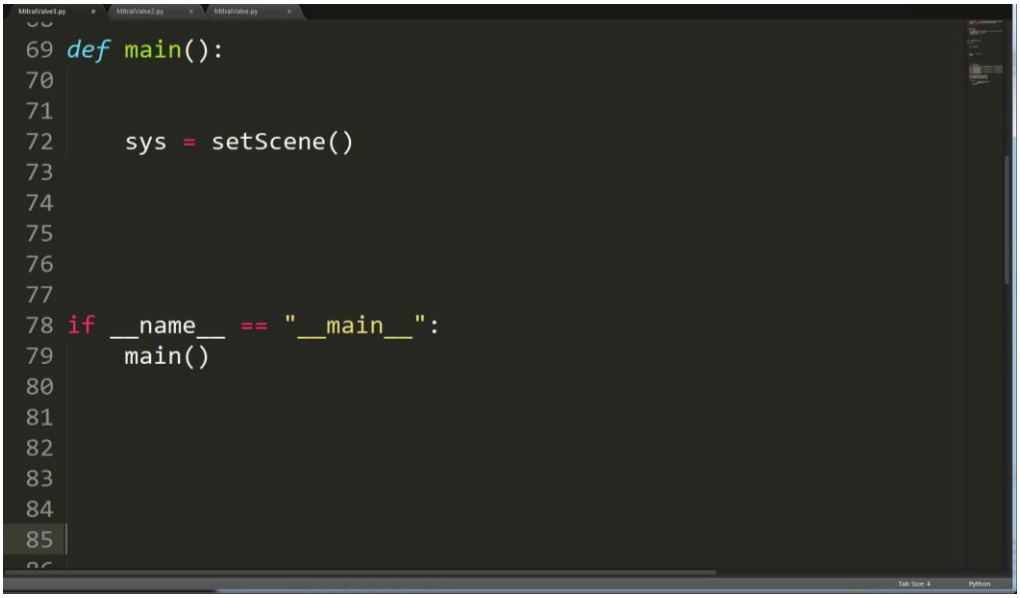


Figura 50: Main Simplet

Cuando se ha definido el objeto *System*, se puede utilizar para insertar objetos nuevos en la escena. Para esta prueba se definió un objeto simple llamado *SimpleMesh* compuesto por 4 figuras planas formando un cuadrado dividido en 4 partes. Esta *simpleMesh* incorpora una pequeña red de masas, donde cada masa estará asignada a un vértice de la *mesh*. De esta forma cuando las masas se muevan por influencia de la gravedad, se ejecutará un método que se encargará de actualizar la posición de cada vértice.

La implementación de *SimpleMesh* está compuesta por la lista de caras que define cada figura, donde cada cara está representada por un triángulo y cada cara tiene asignado un color para poder distinguirlas. Toda la información geométrica se almacena en la estructura de datos pública **tri.** Esta clase además contiene con una lista de masas llamada *masslist* donde se almacena el par masa-índice, donde el índice apunta al vértice asignado a dicha masa.

Cuando se define una nueva masa podemos asignarle 3 parámetros a la creadora **Mass** de *KineticsKit*, m,pos y *fixed*. m es el valor de la masa, pos, es la posición de la masa en la escena, y *fixed* es un atributo booleano que indica si la masa está fijada en la escena o por el contrario la fuerza de la gravedad le puede afectar.

Finalmente *UpdatePos* es el método que se encarga de la actualización de la posición de los vértices en función de la posición de la correspondiente masa y recibe como parámetro de entrada el correspondiente índice.

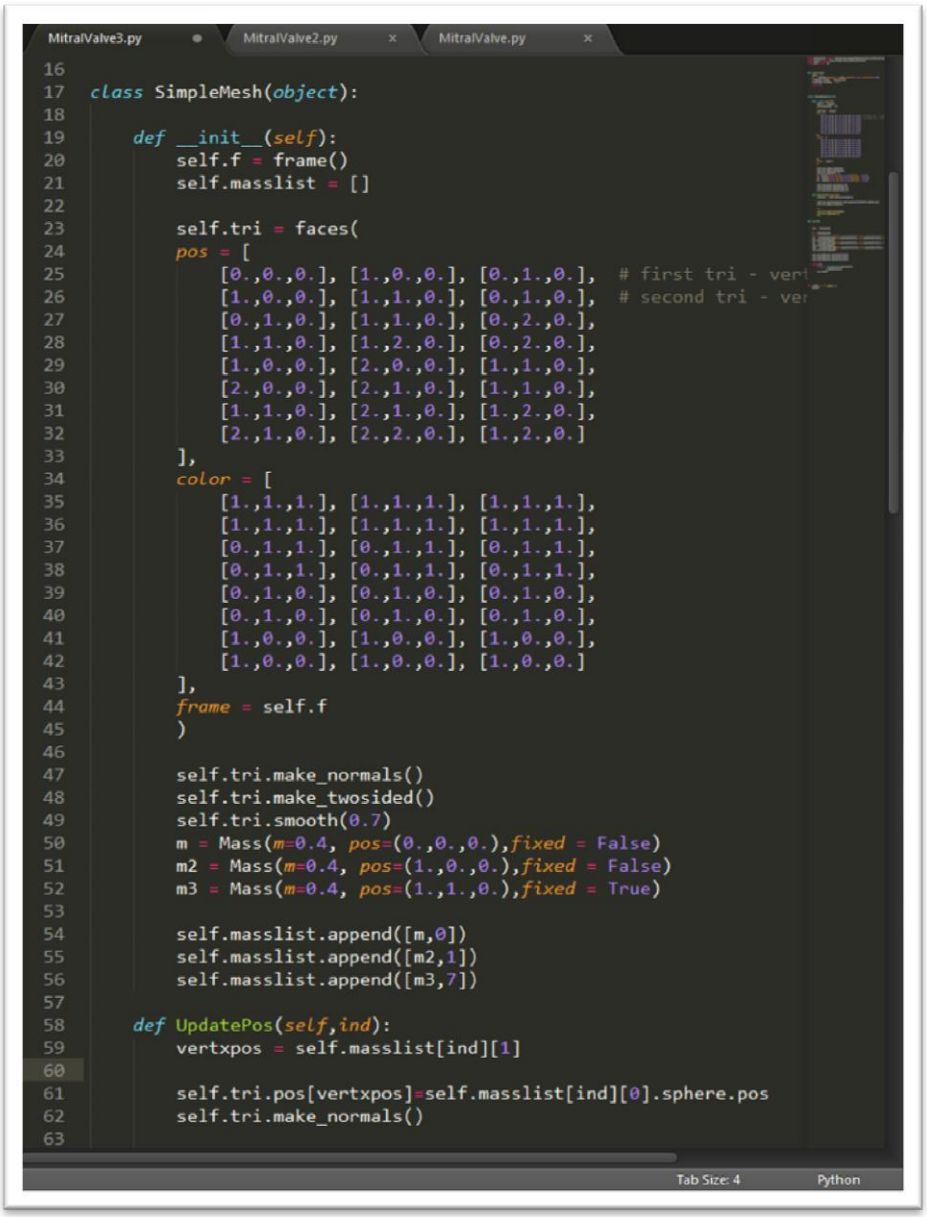


Figura 51: Implementación SimpleMesh

Cuando se crea la lista de masas también se deben interconectar entre ellas. Los conectores o *springs* son objetos de *KineticsKit* y para este ejemplo se utilizó el conector de tipo cilindro (*CylinderSpring*). Además los objetos que tienen algún comportamiento físico deben agregarse al objeto *System* para que tengan algún efecto real en la escena.

En la siguiente figura vemos como tras crear un objeto *SimpleMesh*, interconectamos las masas de su *masslist* y posteriormente insertamos tanto conectores como masas al objeto *System*.

También podemos ver como se crean los conectores de la red de masas utilizando la creadora *CylinderSpring*. Este método recibe como parámetros, m0, m1, k y *damping*. Los valores m0 y m1 son las instancias de las masas que queremos conectar, k es el coeficiente de elasticidad y *damping* es el valor de amortiguación de dicho conector.

Finalmente dentro del bucle *while*, se llama a la función que se encarga de comprobar y actualizar las posiciones de los vértices mediante el método *UpdatePos* y de ejecutar la simulación del siguiente *frame* mediante la función *step*.

```
68
69  def main():
70
71
72      sys = setScene()
73
74      r = SimpleMesh()
75
76      sp = CylinderSpring(m0=r.masslist[0][0], m1=r.masslist[2][0], k=10, damping=0.08)
77      sys.insertSpring(sp)
78      sp = CylinderSpring(m0=r.masslist[0][0], m1=r.masslist[1][0], k=10, damping=0.08)
79      sys.insertSpring(sp)
80      sp2 = CylinderSpring(m0=r.masslist[2][0], m1=r.masslist[1][0], k=10, damping=0.08)
81      sys.insertSpring(sp2)
82
83      sys.insertMass(r.masslist[0][0])
84      sys.insertMass(r.masslist[1][0])
85      sys.insertMass(r.masslist[2][0])
86
87      while(1):
88          for i in range(len(r.masslist)):
89                  r.UpdatePos(i)
90          sys.step()
91
92
93
                                                                    Tab Size: 4    Python
```

Figura 52: Agregando objetos a System

El resultado de la prueba se puede observar en las siguientes imágenes.

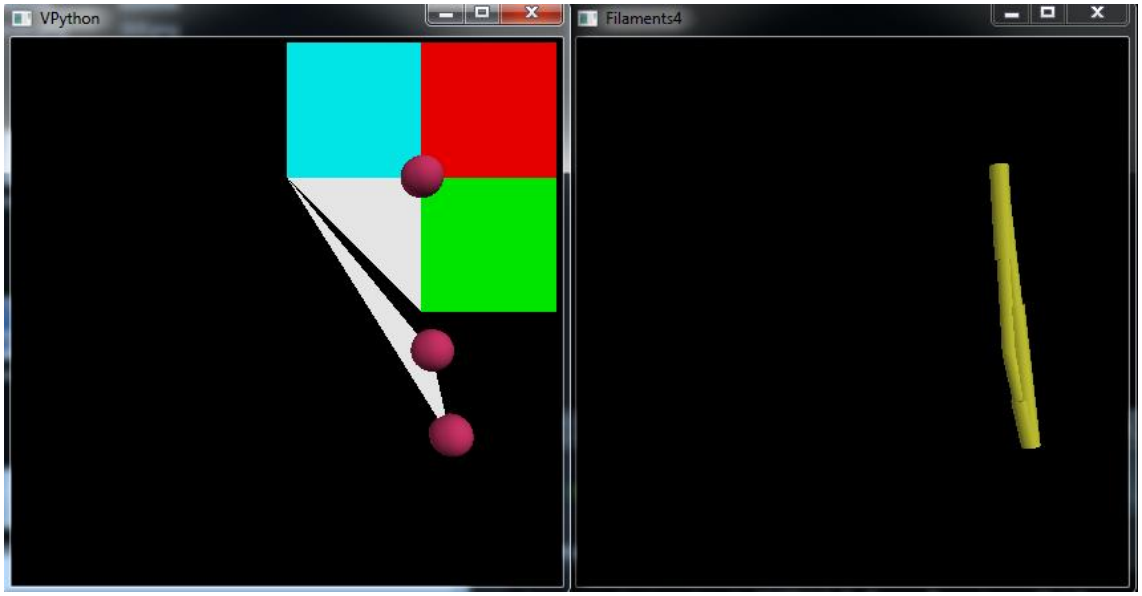


Figura 53: Deformación SimpleMesh

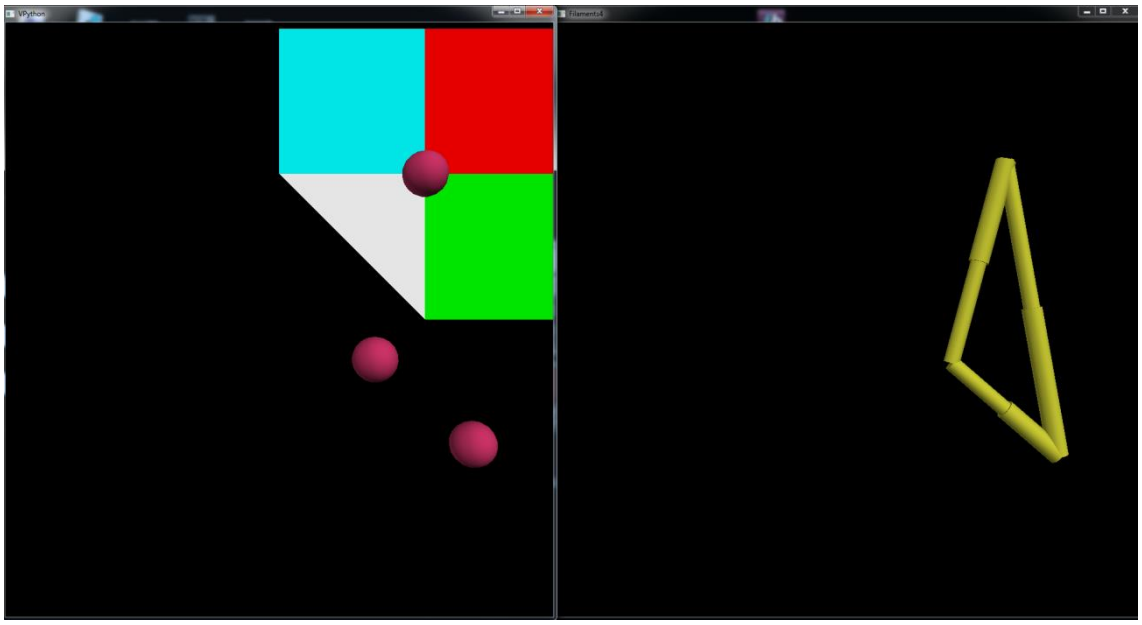


Figura 54: Deformación SimpleMesh

9.2.1.2 Creación de un filamento interactivo

En la siguiente prueba creamos un sencillo código encargado únicamente de estudiar cómo crear nuevos objetos físicos e interconectarlos entre sí.

Para la siguiente implementación, se define una función llamada CreateFilaments que a partir de una masa m, una densidad d y una longitud L crea un filamento de longitud L, con una

cantidad de masas L/d tomando como punto inicial la masa m. Podemos ver la implementación de la función en la siguiente imagen.

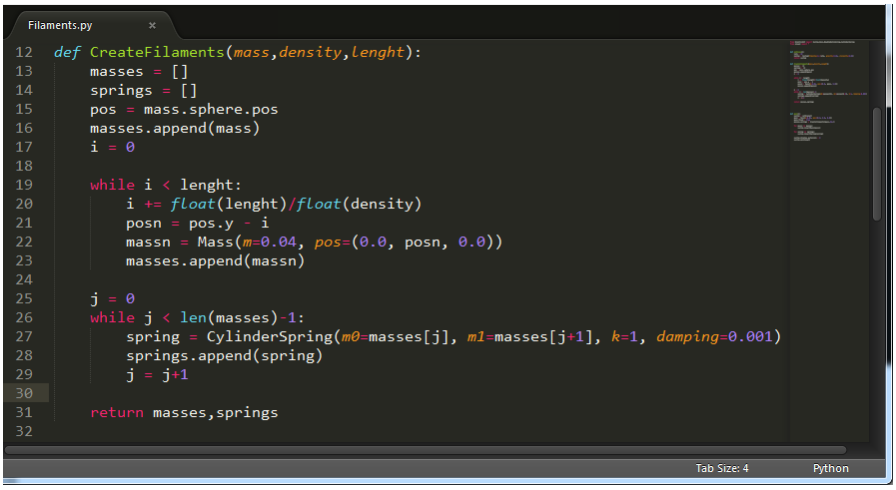


Figura 55: CreateFilaments

A continuación podemos ver un ejemplo de la utilización de la función CreateFilaments. Dado que el método retorna la lista de masas y springs, a continuación hemos de agregarlos a la variable System antes de ejecutar la simulación física. Dado que esta vez no tenemos asociada una Mesh a nuestra red de masas, la simulación física en vez de ejecutarse paso a paso como en el ejemplo anterior, se puede ejecutar mediante una única función denominada **mainloop**.

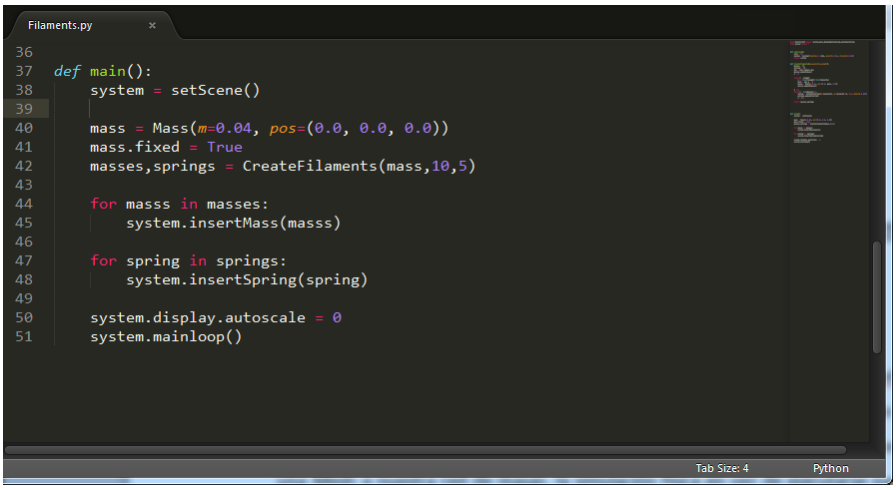


Figura 56: Creando filamentos

El resultado de la ejecución podemos verla en las siguientes imágenes, donde se han definido cuerdas de distintas densidades.

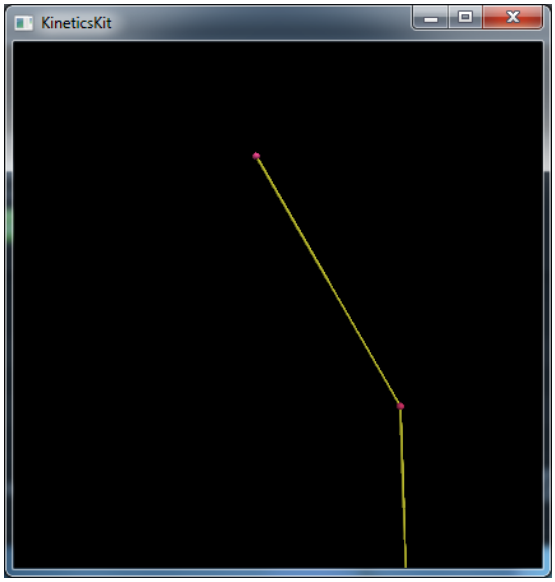


Figura 58: CreateFilamens 10 masas

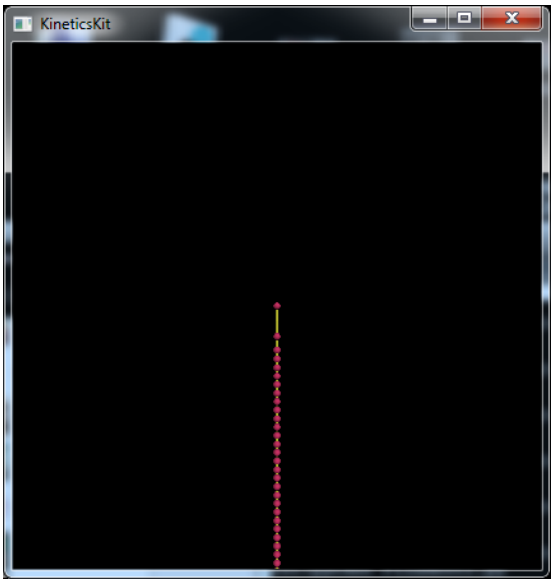


Figura 57: Create filamens 30 masas

9.1.2 Panda 3D

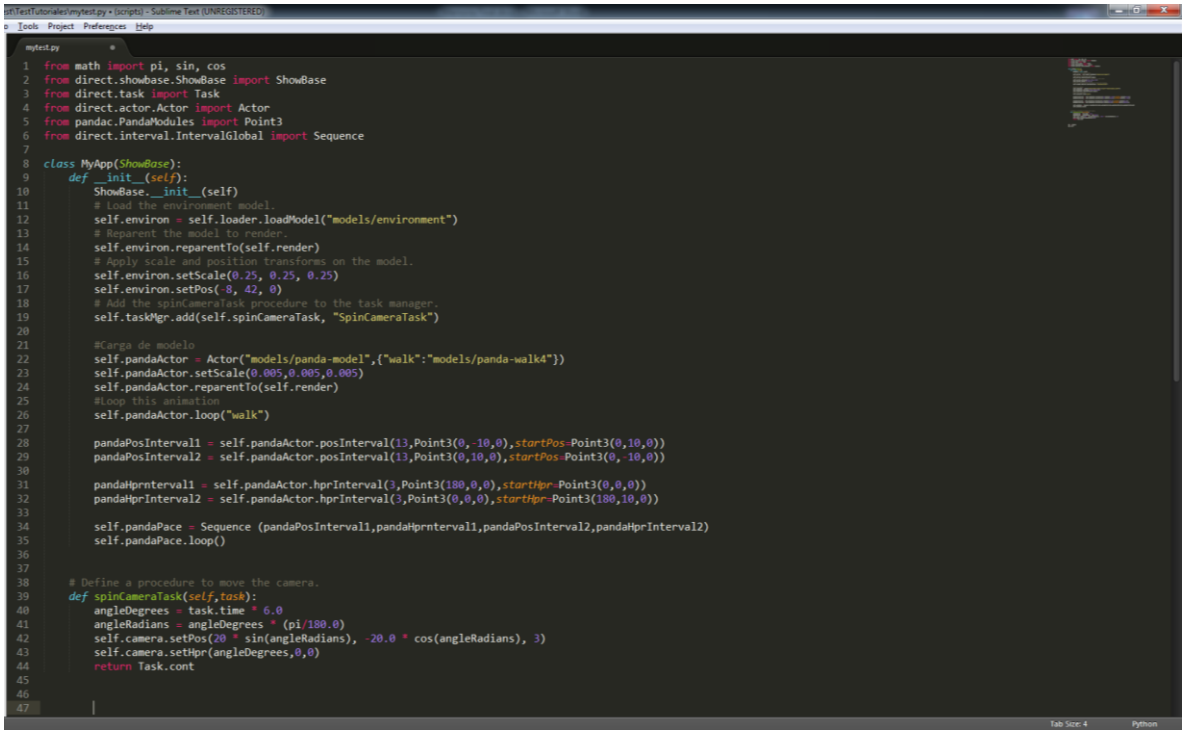
Antes de hablar de las implementaciones realizadas con esta plataforma, se describirá como implementar una aplicación simple con Panda 3D y como añadir un objeto.

Las pruebas requerían de implementar una y otra vez las mismas líneas de código, así que se implementó un sistema de clases sencillo que también se describirá, aunque no forma parte de la implementación final, facilitará el trabajo de lectura de las pruebas.

9.1.2.1 Pruebas realizadas

Este ejemplo ilustra cómo implementar una aplicación sencilla utilizando Panda 3D. En Panda 3D podemos crear una clase base que representará el universo, la escena del juego. Esta clase creada hereda de la clase ShowBase de Panda y hereda los métodos necesarios para gestionar una escena.

Para este ejemplo creamos la clase MyApp que hereda de ShowBase y le añadimos un método denominado spinCameraTask que explicaremos a continuación. Para ejecutar el ejemplo basta con instanciar la clase Myapp y ejecutar el método run.



```
from math import pi, sin, cos
from direct.showbase.ShowBase import ShowBase
from direct.task import Task
from direct.actor.Actor import Actor
from pandac.PandaModules import Point3
from direct.interval.IntervalGlobal import Sequence

class MyApp(ShowBase):
    def __init__(self):
        ShowBase.__init__(self)
        # Load the environment model.
        self.environ = self.loader.loadModel("models/environment")
        # Reparent the model to render.
        self.environ.reparentTo(self.render)
        # Apply scale and position transforms on the model.
        self.environ.setScale(0.25, 0.25, 0.25)
        self.environ.setPos(-8, 42, 0)
        # Add the spinCameraTask procedure to the task manager.
        self.taskMgr.add(self.spinCameraTask, "SpinCameraTask")

        #Carga de modelo
        self.pandaActor = Actor("models/panda-model",{"walk":"models/panda-walk4"})
        self.pandaActor.setScale(0.005,0.005,0.005)
        self.pandaActor.reparentTo(self.render)
        #Loop this animation
        self.pandaActor.loop("walk")

        pandaPosInterval1 = self.pandaActor.posInterval(13,Point3(0,-10,0),startPos=Point3(0,10,0))
        pandaPosInterval2 = self.pandaActor.posInterval(13,Point3(0,10,0),startPos=Point3(0,-10,0))

        pandaHprnterval1 = self.pandaActor.hprInterval(3,Point3(180,0,0),startHpr=Point3(0,0,0))
        pandaHprInterval2 = self.pandaActor.hprInterval(3,Point3(0,0,0),startHpr=Point3(180,10,0))

        self.pandaPace = Sequence (pandaPosInterval1,pandaHprnterval1,pandaPosInterval2,pandaHprInterval2)
        self.pandaPace.loop()


    # Define a procedure to move the camera.
    def spinCameraTask(self,task):
        angleDegrees = task.time * 6.0
        angleRadians = angleDegrees * (pi/180.0)
        self.camera.setPos(20 * sin(angleRadians), -20.0 * cos(angleRadians), 3)
        self.camera.setHpr(angleDegrees,0,0)
        return Task.cont
```

Figura 59: Implementación MyApp

Panda 3D ya incluye algunos ejemplos de modelos y *terrains* que se han utilizado en el ejemplo. En primer lugar configuramos el terreno del ejemplo asignando a la variable *environ* un modelo de terreno mediante el método *loadModel* y lo insertamos en el *SceneGraph* utilizando el método *reparentTo* con parámetro el objeto *render*.

Queremos que la escena ejecute una función cada cierto tiempo, así que definimos el método *spinCameraTask* que recibe como parámetro el objeto *task* y lo añadimos al *taskMgr* de la clase *MyApp*.

También agregamos un actor a la escena que tendrá una animación simple. Para ello creamos un Actor, lo configuramos y también lo agregamos al *SceneGraph*. Finalmente configuramos la animación de caminar del actor.

Al ejecutar la aplicación veremos la siguiente imagen.

Figura 60: Ejemplo MyApp

Con este primer ejemplo vemos que la API nos permite configurar una escena de forma sencilla heredando de una clase de Panda que nos facilita la creación de un controlador de escena al cual le podemos añadir elementos que el Framework reconoce sin mayor esfuerzo. Esta característica hemos de tenerla en cuenta antes de pasar a las implementaciones de prueba final.

La siguiente prueba representa un filamento sujeto en el espacio por un extremo y por el otro a extremo a una masa que el usuario es capaz de estirar para comprobar el manejo de SoftBodies.

```
from AnilloMitral import AnilloMitral
from SoftTest import SoftTest
from RopeTest import Game
from panda3d.core import Point3
from panda3d.bullet import BulletSoftBodyNode
import direct.directbase.DirectStart
from panda3d.bullet import BulletBoxShape
from panda3d.bullet import BulletRigidBodyNode
from panda3d.core import BitMask32


if __name__ == '__main__':

    w = World()

    #Test de filamento
    p1 = Point3(5,0,30)
    p2 = Point3(5,0,20)
    mass = 50.0
    n = 10

    go = FilamentTest(w.getInfo(),p1,p2,n,mass)
    visNode = go.GetVisNode()
    go.SetNP(w.appendGameObject("Soft",go.getNode(),"S"))
    visNP = w.appendGameObject("Sof",visNode,"-")
    visNP.setTexture(loader.loadTexture(r'../models/wood.png'))


    w.acceptBehaviour("arrow_down",go.moveAnchorDown)
    w.acceptBehaviour("arrow_left",go.fixanchor)
    w.acceptBehaviour("arrow_right",go.releaseanchor)
    #Creando ancla

    # Box
    shape = BulletBoxShape(Vec3(1,1,1))
    node = BulletRigidBodyNode('Box')
    node.setMass(10000)
    node.addShape(shape)
    boxNP = w.appendGameObject("RB",node,"R")
    boxNP.setPos(p2)
    boxNP.setCollideMask(BitMask32.allOn())
    #Anyadiendo ancla
    go.appendAnchor(boxNP)

    run()
```

Figura 61: Ejemplo de Filamento

Para este ejemplo consideramos las clases World y FilamentTest como clases desarrolladas para las pruebas que posteriormente describiremos. World es el controlador de escena y equivale a la clase MyApp del ejemplo anterior. FilamentTest es una clase que implementa una cuerda SoftBody que va desde el punto p1 al punto p2 con una densidad n y una masa mass determinada. Esta clase solo devuelve el node del objeto y hay que añadirlo a posteri al SceneGraph para obtener el NodePath correspondiente. Entre las líneas 26 y 29 creamos un nodo de visualización para agregar la información geométrica del objeto.

Entre las líneas 32 a 35 añadimos a World algunas funciones que se comportan como **tasks,** estas son las que permiten la interacción con el usuario y son las que darán al usuario la funcionalidad de estirar la cuerda, atarla al espacio y volverla a soltar.

Finalmente se ata al otro extremo de la cuerda una masa con peso y se añade a la cuerda utilizando la función appendAnchor.

El resultado de la prueba se puede observar en las siguientes imágenes.

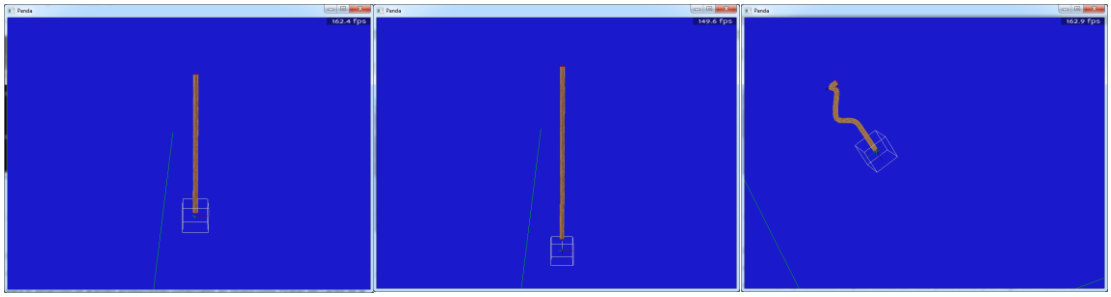


Figura 62: Estirar y liberar cuerda

Antes de describir las dos pruebas denominadas Implementación A e Implementación B.

Hablaremos de la estructura de clases diseñada para ambas pruebas. Esta capa lógica.

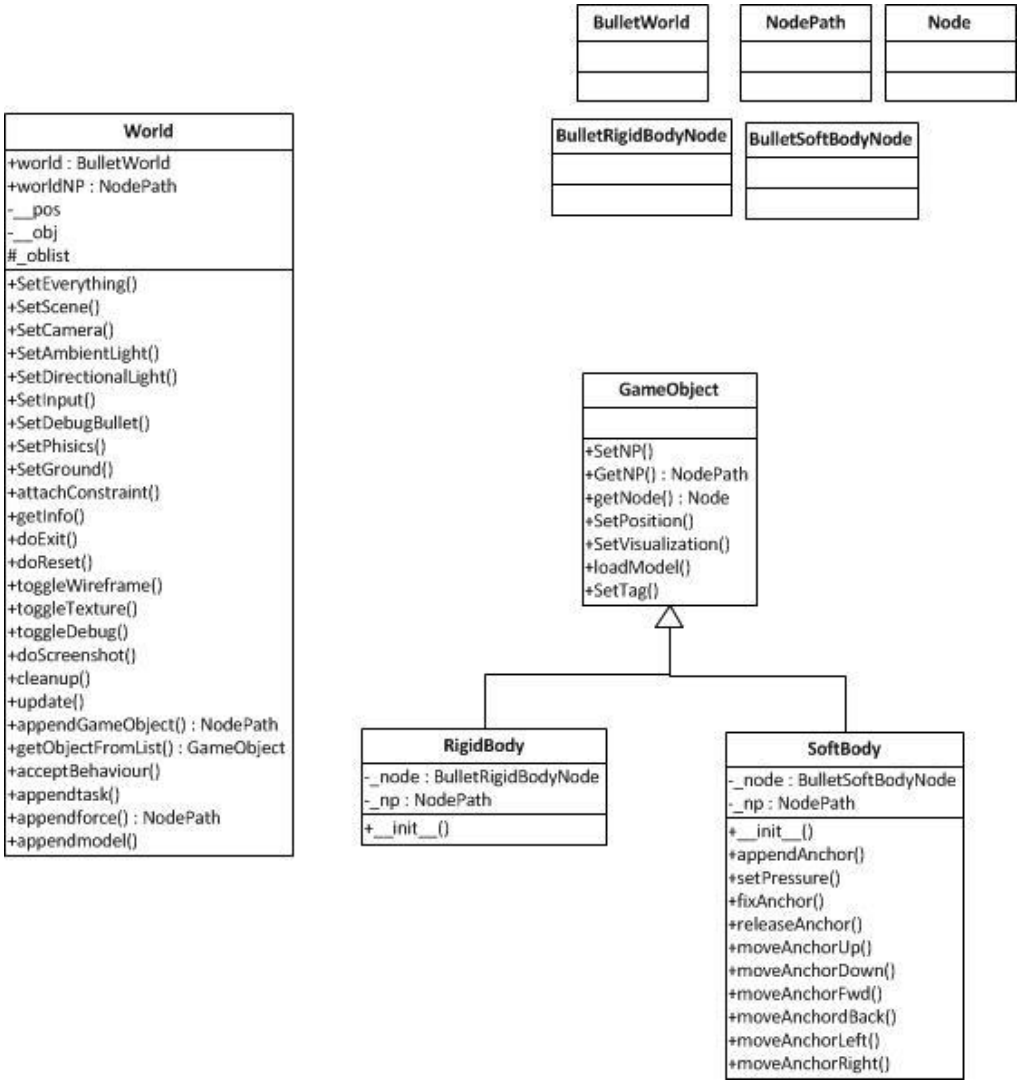


Figura 63: Esquema de Dominio de pruebas

Sin entrar demasiado en tareas de especificación, esta es la descripción de las clases que aparecen en la imagen anterior.

World: Representa el controlador de escena y la clase que se encarga de administrar el objeto ScenGraph. Esta clase posee los métodos necesarios para añadir nuevos objetos a la estructura de datos de escena.

World::SetEverything: Este método crea un nuevo nodo en la SceneGraph que se corresponde a la nueva escena creada, configura la cámara,la iluminación. También se encarga de configurar las teclas que podrá reconocer la configuración, de activar el modo Debug en caso que sea necesario y activa el motor de físicas.

World::SetScene: Configura el escenario, permite cambiar el Background y el ratio de frames por segundo.

World::SetCamera: Configura la posición y el punto de referencia de la cámara.

World:: SetAmbientLight: Configura la luz ambiental de la escena dado un color.

World:: SetDirectionalLight: Permite añadir una luz direccional con un vector y un color.

World:: SetInput: Configura que métodos deben reaccionar al pulsar una tecla determinada. En este caso el método está configurado para activar los distintos métodos del modo Debug.

World:: SetDebugBullet: Anida al NodePath del World(worldNP) un nodo de Debug. El nodo de bug active el DebugMode que permite visualizer el eje de coordenadas, capturar los frames por Segundo (fps), cambiar el modo de visualización de los modelos, y capturar screenshots.

World:: SetPhisics: Configura las físicas de la escena, es decir, la aceleración y dirección de la gravedad, la densidad del aire,etc.

World:: SetGround: Crea un plano formado por los vectores (0,0,1) y (1,0,0) que simula ser el "suelo" de la escena".

World:: getInfo: Devuelve el atributo info de la clase DirectObject.

World:: doExit: Cierra la aplicación.

World:: doReset: Reinicia la aplicación.

World:: toggleWireframe: Cambia el modo de visualización para que sea visible/invisible la malla del modelo.

World:: toggleTexture: Cambia el modo de visualización para que sea visible/invisible la textura del modelo.

World:: toggleDebug: Activa/Desactiva el modo Debug.

World:: doScreenshot: Realiza una captura de pantalla.

World:: cleanup: Elimina todos los nodos que estuvieran anidados en la escena para eliminarlos.

World:: update: Este método, es una task que se ejecuta en cada frame y tiene por objetivo actualizar las físicas.

World:: appendGameObject: Añade un nuevo objeto a la escena anidándolo a la escena actual. Este método distingue entre RigidBodies y SoftBodies.

World:: getObjectFromList: Devuelve un objeto determinado de los objetos que tiene anidados la escena.

World:: acceptBehaviour: Este método sirve para asociar eventos de teclado con métodos nuevos.

World:: appendtask: Este método permite agregar una nueva task a la escena.

World:: appendforce: Mediante este método se puede crear un nodo de fuerza a la escena.

World:: appendmodel: Mediante esta función se puede agregar un nuevo nodo que contenga información geométrica a la escena.

GameObject Esta clase permite almacenar información de objetos de escena nuevos , distinguimos entre dos tipos de GameObjects, RigidBodies y SoftBodies.

GameObject::SetNP: Setter que almacena el NodePath en el atributo local _np.

GameObject::GetNP: Consultora que obtiene el NodePath almacenado.

GameObject::getNode: Consultora que obtiene el Node del objeto al que representa el GameObject.

GameObject::SetPosition: Modifica la posición del objeto en la escena.

GameObject::loadModel: Carga de un modelo a partir de un fichero externo.

GameObject::SetTag: Se añade una etiqueta o Tag a un NodePath.

RigidBody: Esta clase permite crear objetos RigidBody utilizando la clase BulletRigidBodyNode.

SoftBody: Esta clase permite crear objetos RigidBody utilizando la clase BulletSoftBodyNode. Esta implementación de SoftBodies requiere los ficheros elem, facem y nodem que explicaremos más adelante.

SoftBody::appendAnchor: Permite agregar un ancla a un SoftBody. Un ancla es un punto de fijación, un punto del SoftBody que no queda afectado por la gravedad u otras aceleraciones en función de si está fijada o liberada.

SoftBody:: moveAnchorDown: Método que permite mover un ancla en dirección (0,-1,0).

SoftBody:: moveAnchorUp: Método que permite mover un ancla en dirección (0,+1,0).

SoftBody:: moveAnchorBack: Método que permite mover un ancla en dirección (0,0,-1).

SoftBody:: moveAnchorFwd: Método que permite mover un ancla en dirección (0,0,+1).

SoftBody:: moveAnchorLeft: Método que permite mover un ancla en dirección (-1,0,0).

SoftBody:: moveAnchorRight: Método que permite mover un ancla en dirección (+1,0,0).

SoftBody:: fixanchor: Método que fija un ancla y evita que le afecte la aceleración de la gravedad.

SoftBody:: releaseanchor: Método que libera un ancla y permite que le afecte la aceleración de la gravedad.

9.2.2.2 Implementación A

Esta primera implementación tuvo por objetivo estudiar la librería Bullet en el campo del manejo de los SoftBodies. El objetivo es poder estirar una esfera SoftBody y observar su deformación con diferentes configuraciones de elasticidad. Para la prueba se utilizó el modelo de una esfera. Inicialmente los modelos están poligonados con triángulos pero un SoftBody tipo BulletSoftBodyNode requiere que estén poligonados utilizando tetraedros.

Para tal objetivo antes de iniciar la primera implementación se utilizó la aplicación tetgen que convierte un archivo en formato .poly y da como resultado tres archivos con las extensiones .ele, .face y .node que contiene la información geométrica de un modelo utilizando tetraedros en vez de triángulos.

Además antes de hacer la transformación a tetraedro hay que hacer una conversión previa ya que el modelo de la esfera que se utilizó está almacenado en un fichero con formato egg o egg.pz si está comprimido, por este motivo se utilizó un script de conversión llamado egg2polyp.

Lo primero que nos encontramos es que no hay una forma de manipular directamente el SoftBody. Panda 3D no ofrece un modo de edición que permita visualizar la escena y ubicar los modelos de forma visual. Además vemos que la versión de Bullet de Panda no ofrece una forma de interactuar directamente con el objeto SoftBody. Tras consultar con miembros de la comunidad de desarrolladores de Panda nos comentan que esto no es posible ya que la geometría del modelo se reescribe al final de cada frame al calcularse las deformaciones, por tanto se evita que pueda modificarse para prevenir errores críticos en la aplicación.

Lo segundo que observamos es que a pesar que Panda 3D facilita la implementación de handlers de interrupciones de teclado, no es así con las interacciones de ratón, al menos no sin utilizar código de terceras personas. Debido a esto y para mantener el alcance de la prueba dentro de unos límites, se decidió utilizar un objeto tipo RigidBody anclado al SoftBody para utilizarlo como elemento de interacción.

Así pues el código de la implementación es el siguiente:

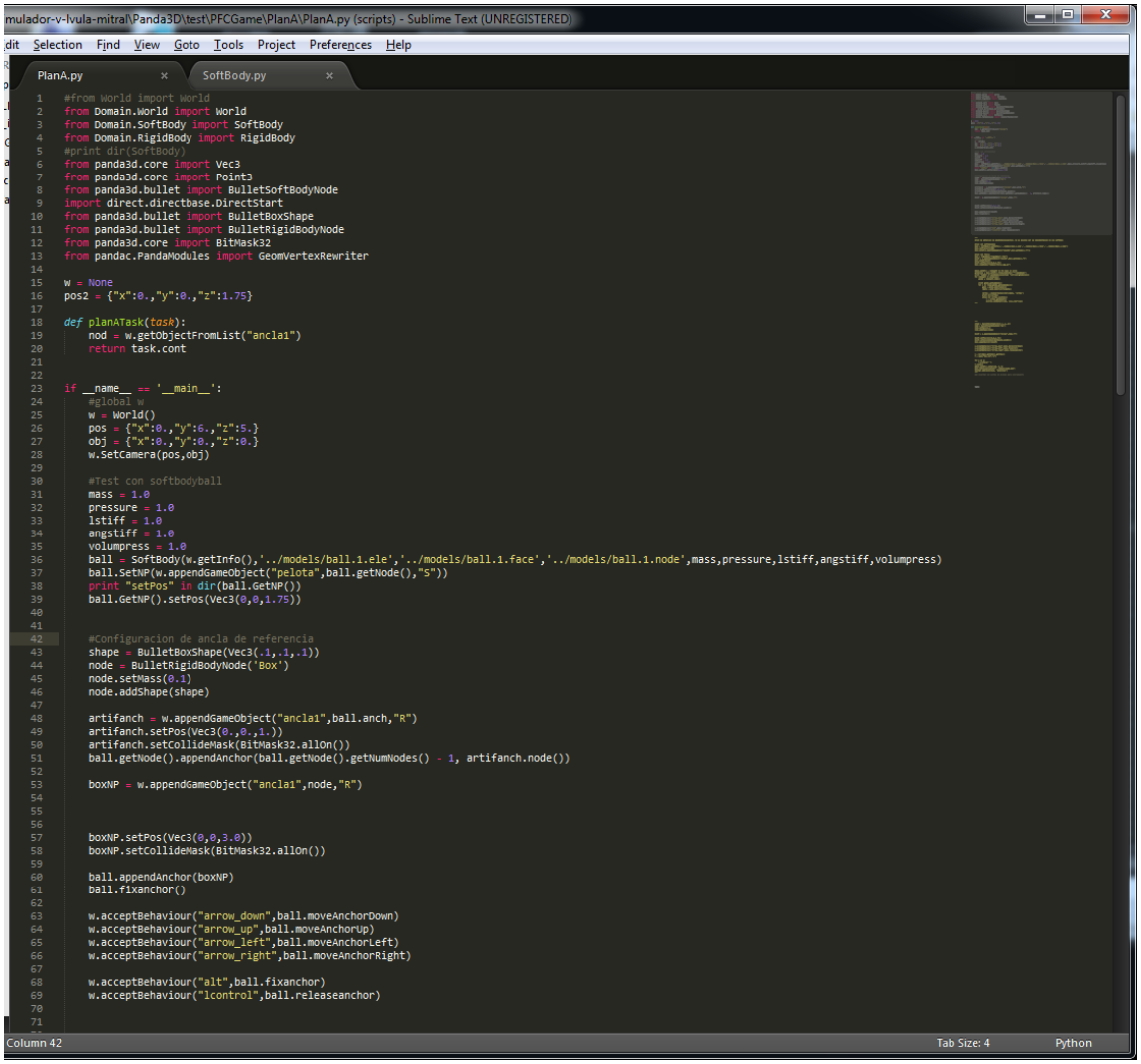


Figura 64: Implementación Plan A

Como se puede ver en el código en primer lugar se crea una instancia de World y se configura la cámara. Posteriormente se carga el modelo de la esfera previamente transformado y se le asignan los atributos físicos que se irán modificando para las distintas pruebas. Los atributos son la masa, la presión interna, la eslasticidad lineal, la elasticidad angular y el ratio de preservación de volumen.

A continuación se crea el ancla de referencia que servirá de elemento de interacción del usuario y finalmente se le indica a la instancia de World las funcionalidades necesarias para que el ancla se mueva utilizando las teclas de la cruz de flechas del teclado.

Estas son las pruebas realizadas con diferentes configuraciones físicas.

Masa = 1.0 kg, **presión** = 1.0, **elasticidad lineal** = 1.0, **elasticidad angular** = 1.0, **volumepress** = 1.0

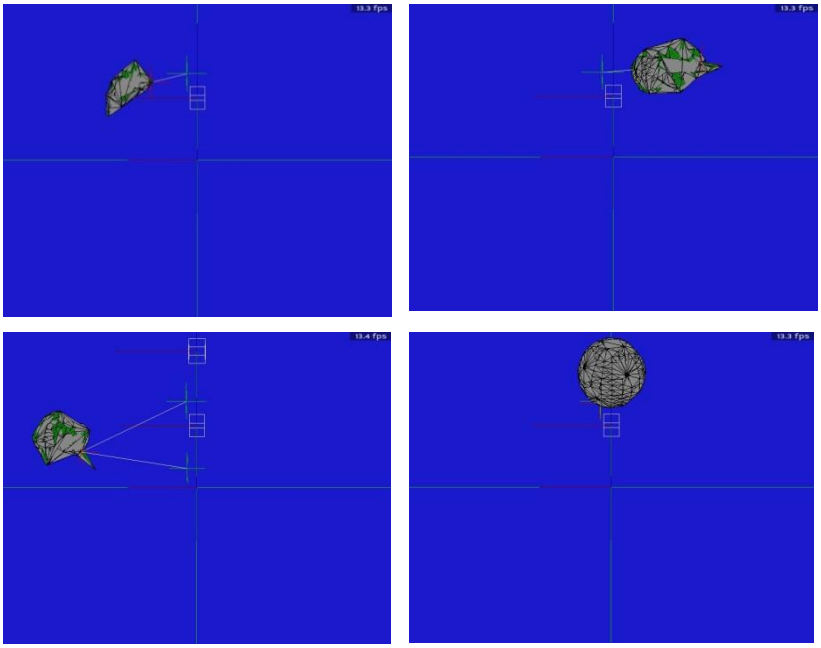


Figura 65: Prueba1

Masa = 1.0 kg, **presión** = 1.0, **elasticidad lineal** = 0.5, **elasticidad angular** = 1.0, **volumepress** = 1.0

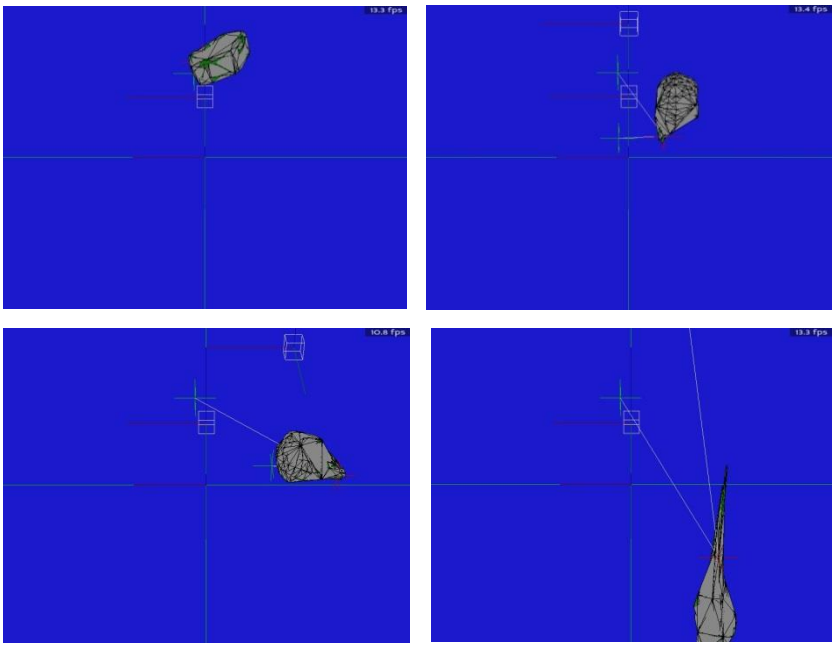


Figura 66: Prueba2

Masa = 1.0 kg, **presión** = 1.0, **elasticidad lineal** = 0.1, **elasticidad angular** = 1.0, **volumepress** = 1.0

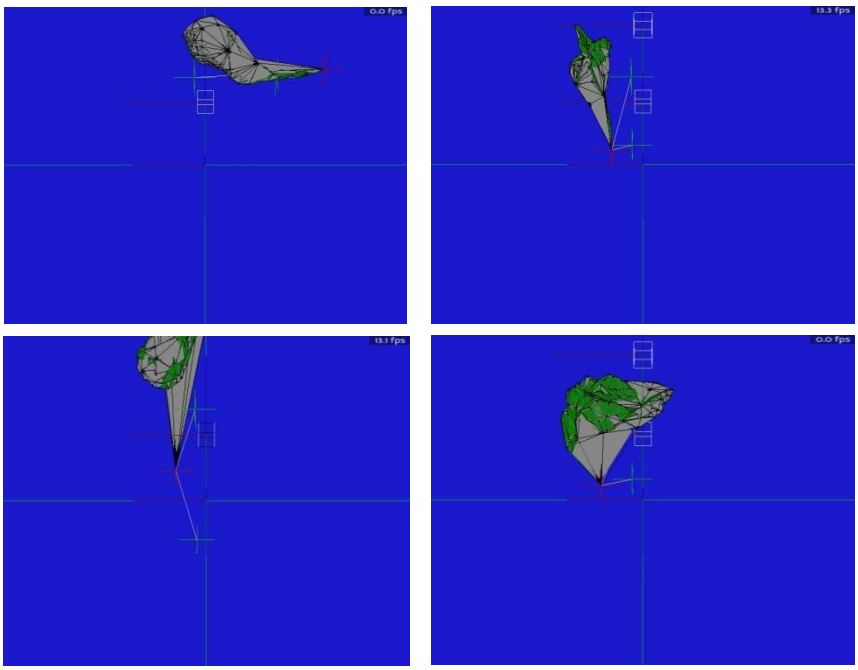


Figura 67: Prueba3

Masa = 1.0 kg, **presión** = 1.0, **elasticidad lineal** = 1.0, **elasticidad angular** = 1.0, **volumepress** = 0.5

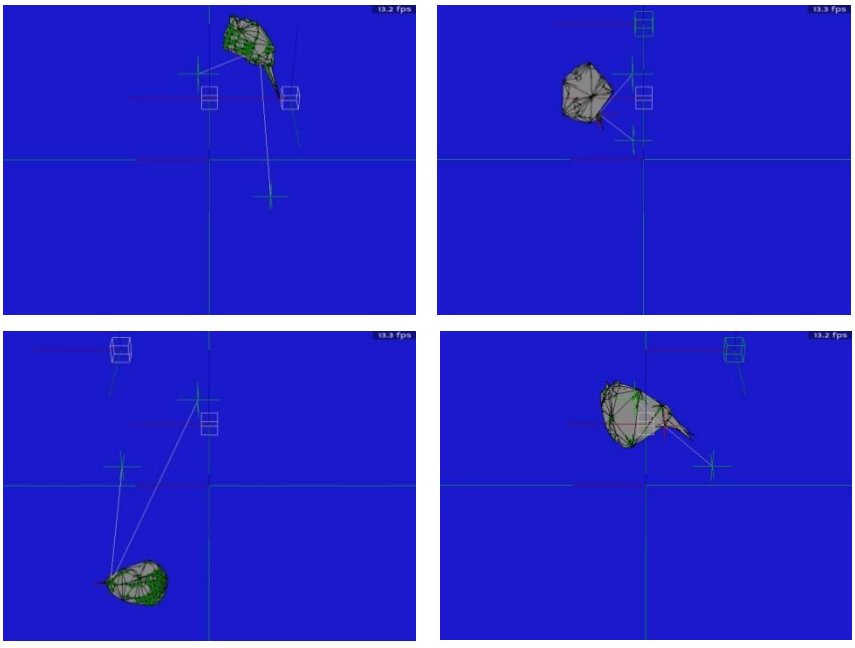


Figura 68: Prueba4

Masa = 1.0 kg, **presión** = 1.0**, elasticidad lineal** = 1.0, **elasticidad angular** = 1.0, **volumepress** = 0.1

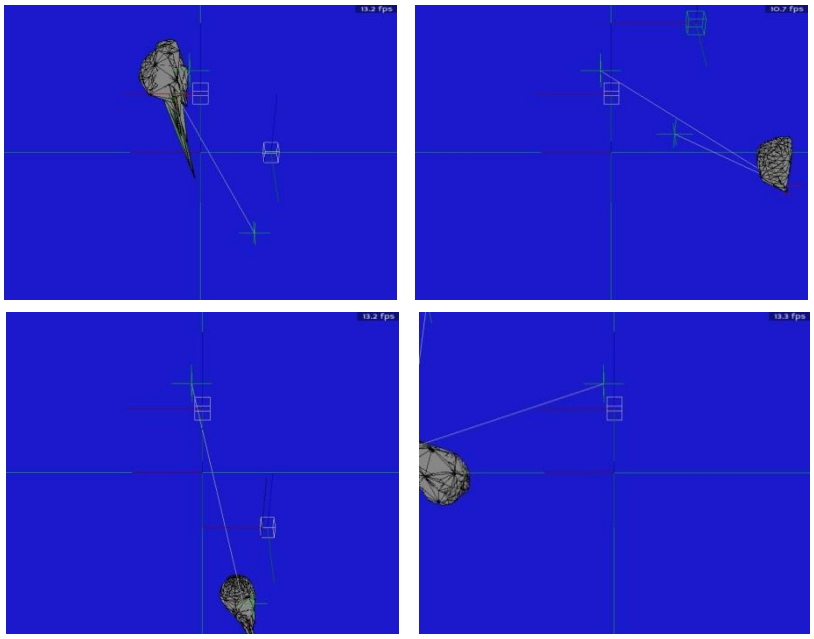


Figura 69: Prueba5

Masa = 1.0 kg, **presión** = 1.0, **elasticidad lineal** = 1.0, **elasticidad angular** = 0.5, **volumepress** = 1.0

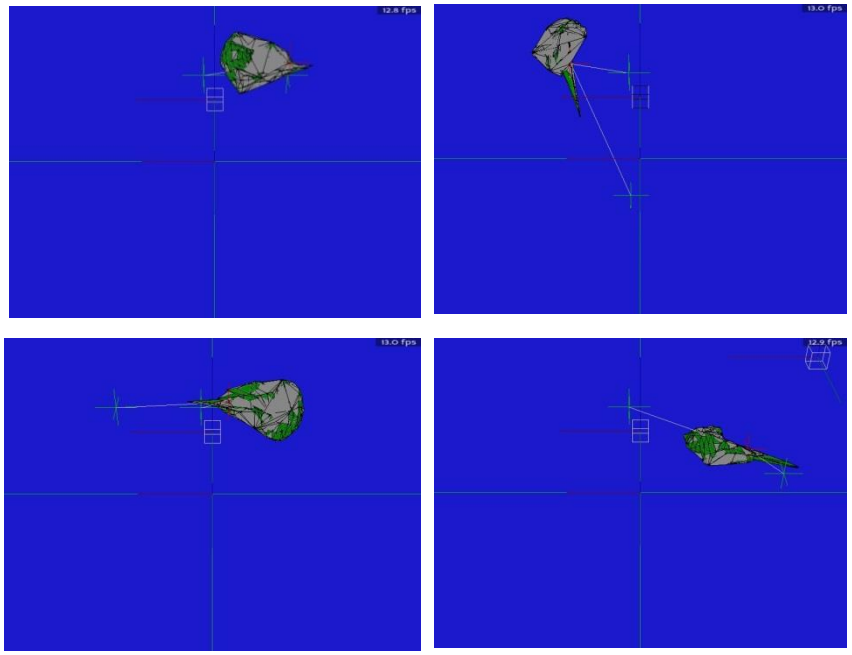


Figura 70: Prueba6

Masa = 1.0 kg, **presión** = 1.0, **elasticidad lineal** = 1.0, **elasticidad angular** = 0.1, **volumepress** = 1.0

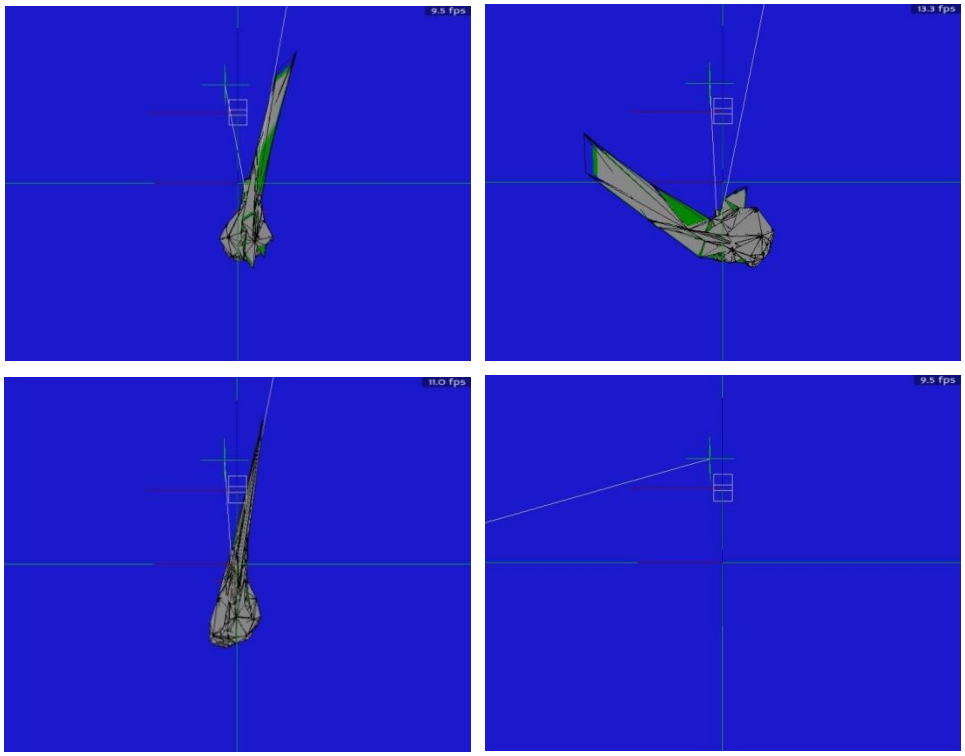


Figura 71: Prueba7

Masa = 1.0 kg, **presión** = 5.0, **elasticidad lineal** = 1.0, **elasticidad angular** = 1.0, **volumepress** = 1.0

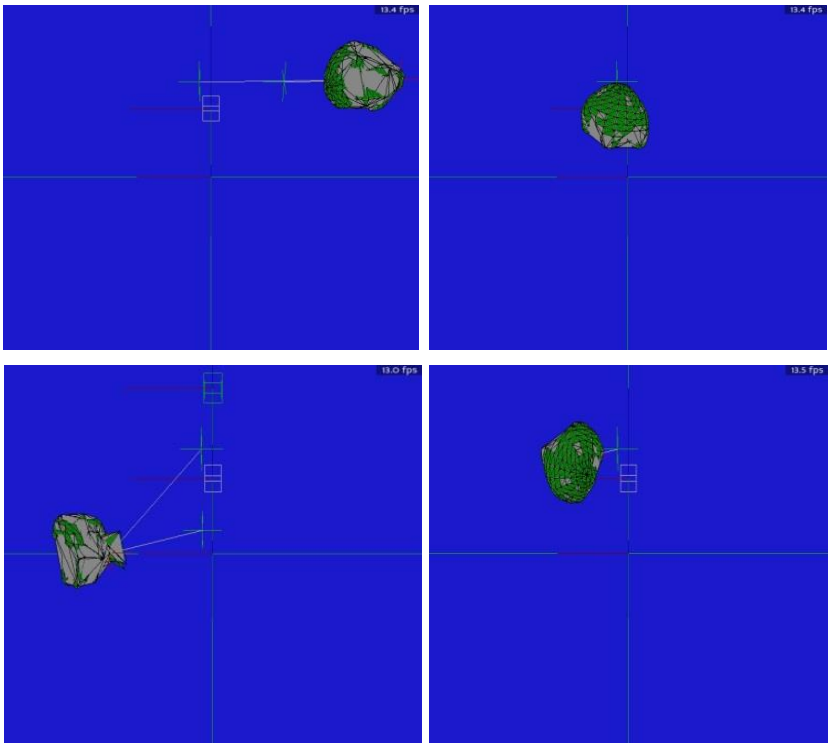


Figura 72: Prueba8

Masa = 1.0 kg, **presión** = 10.0, **elasticidad lineal** = 1.0, **elasticidad angular** = 1.0, **volumepress** = 1.0

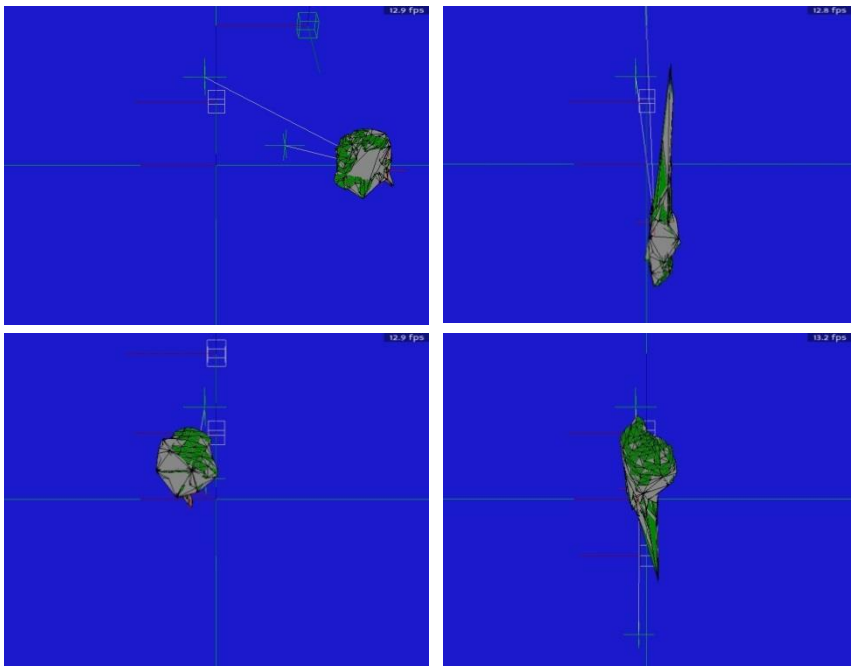


Figura 73: Prueba9

Podemos observar el comportamiento de la esfera a lo largo de las 9 pruebas, pero antes de sacar conclusiones observamos algo común a todas ellas. Como se puede observar en esta imagen recortada el Debuger ha capturado que con una simple esfera, los fps caen a 12.9. Esto es alarmante teniendo en cuenta que el modelo de válvula mitral final es mucho más complejo y las interacciones también lo serán.

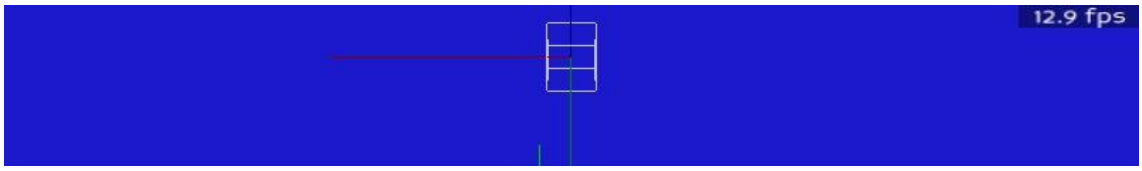


Figura 74: Caída de fps

Prueba 1: Considerando los primeros parámetros podemos observar un modelo estable que tiende a mantener la forma original sin ser demasiado rígido, la sensación es la de estar moviendo una pelota de playa.

Prueba 2: Podemos observar como al reducir el ratio de elasticidad lineal a la mitad el objeto pierde consistencia y al desplazarlo mínimamente tiende a colapsarse sobre sí mismo.

Prueba 3: En este caso cuando la elasticidad lineal es 0.1 el modelo se vuelve tan inestable que incluso la aplicación llega a producir crashing al expandirse exageradamente el modelo a causa de las fuerzas recibidas.

Prueba 4: Al modificar la preservación de volumen se produce un efecto similar que al reducir la elasticidad a 0.1 pero sin perder estabilidad. Vemos pues como cuando el modelo al recibir un impulso determinado y luego quedar en reposo no vuelve a su forma original.

Prueba 5: Al reducir aún más la preservación de volumen se observa un efecto más acusado de lo anteriormente mencionado.

Prueba 6: Cuando lo que reducimos es la elasticidad angular observamos que el modelo es mucho más elástico al hacer movimientos circulares, esto se debe a que la elasticidad angular y lineal son componentes de un mismo tipo de elasticidad, donde la deformación del modelo está influenciada por la dirección en la que se aplica el impulso.

Prueba 7: En esta prueba podemos observar de forma más acusada las deformaciones producidas en el modelo al realizar movimientos circulares.

Prueba 8: Variando la presión interna del modelo observamos que al mover el modelo este se deforma correctamente, pero la presión interna del mismo amortigua estas deformaciones y las compensa intentando mantener la forma original.

Prueba 9: Finalmente observamos las deformaciones producidas a mayores niveles de presión con los valores actuales de elasticidad el modelo se comporta de forma inestable pero en estado de reposo tiende a mantener la forma original.

9.2.2.3 Implementación B

Esta segunda prueba se diseñó para crear un modelo de comportamiento de la válvula, independientemente de la utilización de un SoftBody, este fue el primer modelo de apertura y cierre del proyecto.

Para esta prueba realizamos diversas aproximaciones. En la primera de estas aproximaciones, se implementó un mecanismo simple de apertura y cierre con dos cajas físicas que simulan ser una compuerta. Para poder realizar dicha prueba se implementó un script que simulaba el incremento de presión intraauricular traducida en una fuerza aplicada en dirección vertical de arriba hacai abajo, fuerza que cada cierto periodo de tiempo se invertía para imitar el cambio de la ventaja de presión intraventricular, provocando de esta forma que las puertas se abrieran en el sentido contrario.

El resultado de esta primera aproximación la podemos ver en las siguientes imágenes.

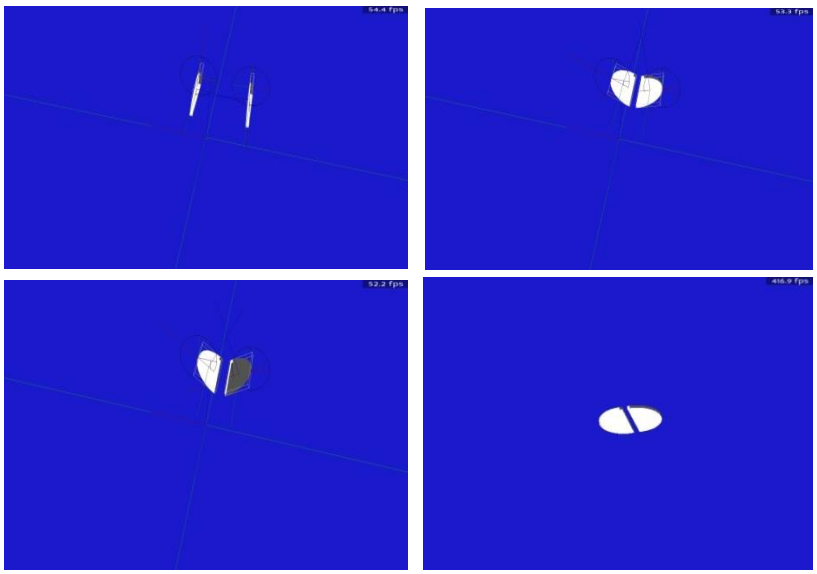


Figura 75: Pruebas Puertas RB

Dado que no había ningún componente que impidiera que la puerta se diera la vuelta como cabía esperar estas giraban como una ruleta. Por este motivo la segunda aproximación incorporó cuatro cuerdas, dos de ellas representan ser cuerdas tendinosas de la válvula y las otras dos representan la tensión provocada dentro del ventrículo cuando el flujo sanguíneo pasa desde la aurícula hasta el ventrículo.

El resultado de dichas pruebas se puede apreciar en las siguientes imágenes:

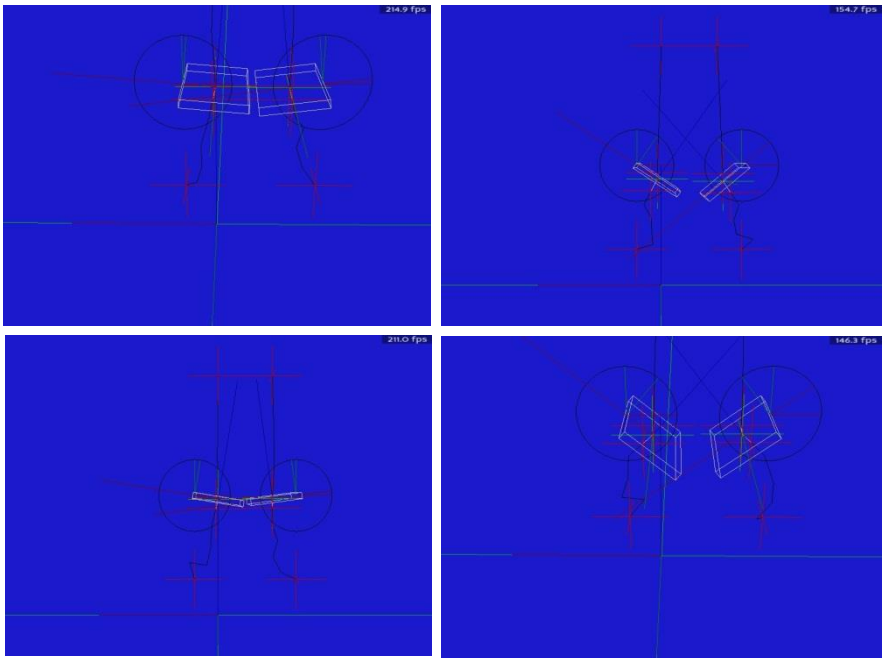


Figura 76: Prueba de puertas

Tal y como se puede ver en las imágenes este sencillo ejemplo presenta el comportamiento básico buscado utilizando elementos rígidos para las puertas y SoftBodies para las cuerdas. En este punto del proyecto la cuestión de implementar un mecanismo de apertura y cierre con una tecnología determinada se ha conseguido. Sin embargo aparecen otras cuestiones relacionadas con la decisión de tratar válvulas y cuerdas como un solo elemento o implementarlos como objetos separados aunque dadas las funcionalidades de Panda 3D lo más lógico era implementar ambos objetos por separado. Por este motivo se construyó la siguiente aproximación, el cual vuelve a introducir el elemento de SoftBody en la misma funcionalidad.

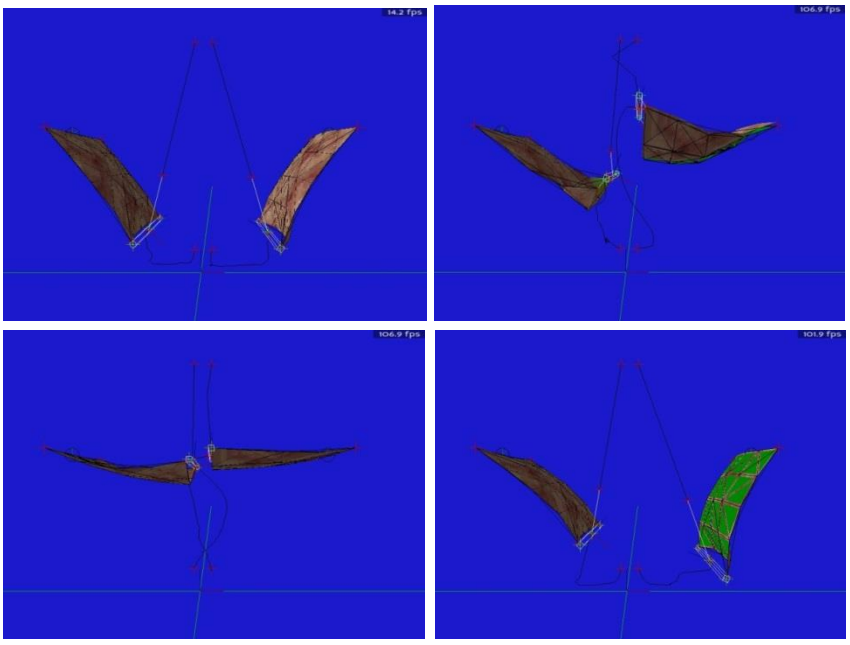


Figura 77: Prueba Velos Soft

La siguiente aproximación vuelve a introducir el elemento SoftBody en la ecuación partiendo de la premisa de repetir el experimento utilizando cuerdas y velos Softodies como elementos separados. Este es el resultado.

Esta implementación se comporta de una forma muy similar a la prueba con puertas RigidBody, sin embargo el desarrollo de esta prueba dejó al descubierto algunas limitaciones tecnológicas.

En primer lugar averiguamos que no es posible interconectar dos objetos SoftBody directamente. Para la prueba se utilizó un RigidBody con forma de prisma en el área de cada velo donde estaría ubicada la zona de coaptación. La librería de Bullet permite ajustar la "dureza" de las anclas entre objetos pero aun ajustando la dureza al máximo, durante la prueba se puede apreciar claramente que aparece una separación física no deseada entre las cuerdas y los velos. Esto quiere decir que en caso de implementar la válvula utilizando Panda 3D, esta no se podría crear manteniendo los elementos separados y luego anidándolos dentro de un solo objeto como hubiera parecido lógico, si no que habría que crearlo como un solo elemento, impidiendo la configuración física de cada elemento de forma individual.

La última implementación consistió en reemplazar los velos rectangulares por unos modelos que ya empezaban a aproximarse a un modelo visualmente realista, como el que se puede apreciar en la figura 50. El principal inconveniente encontrado, es que la API no proporciona ninguna funcionalidad que permita detectar vértices determinados de un modelo. Esto quiere decir que no es practicable detectar un punto determinado del modelo para poder crear las anclas de las cuerdas tendinosas o fijar los velos al anillo mitral. Lo más parecido a lo requerido es una funcionalidad que devuelve el punto en el espacio más cercano donde haya geometría del modelo, información que se falsea nada más ejecutar dicha función ya que esto cambia constantemente. Además, tal y como se pudo observar en la Implementación A la caída de los frames por segundo es considerable y llega a provocar el crash de la aplicación. Se repitió la prueba aplicando simplificadores al modelo sin conseguir resultados satisfactorios.

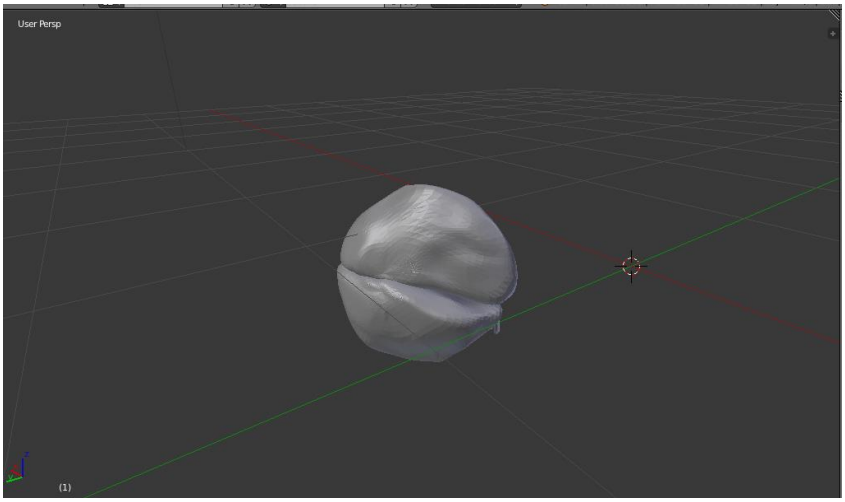
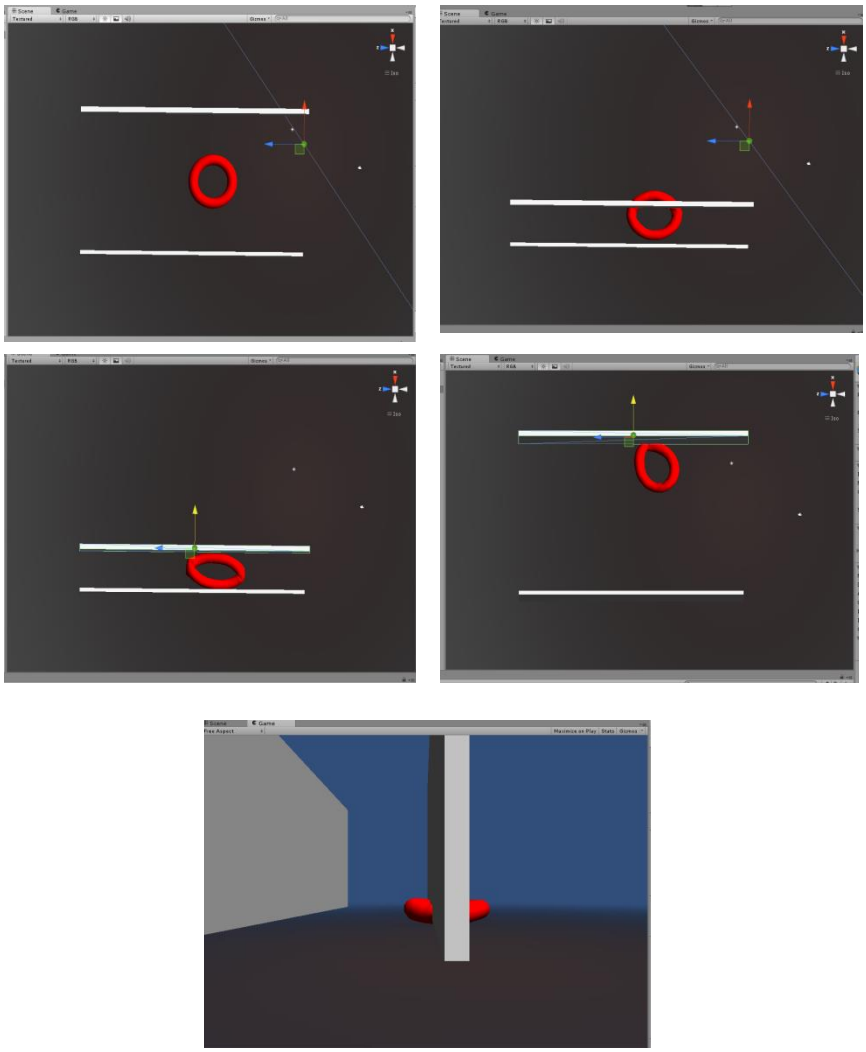


Figura 78: Velos realistas

9.1.3 Unity 3D

9.1.3.1 Prueba SoftBody del anillo Mitral

La primera prueba consiste en la creación de un objeto sencillo al que se le aplica las propiedades físicas de un SoftBody. El objeto escogido es el modelo de un Thorus, ya que geométricamente es lo más parecido visualmente a lo que será una válvula mitral.



Figua 79: SBs en Unity 3D

Como se puede observar en la imagen. El Thorus utilizado tiene una propiedad de SoftBody y vemos como dos objetos RigidBody lo presionan para comprobar las deformaciones.

Observamos un comportamiento inesperado al apreciar clipping entre el Thorus y la barra superior y como el objeto queda enganchado como si el SoftBody fuera un objeto pegajoso.

9.1.3.2 Prueba SoftBody de filamentos

Tras modificar diversas configuraciones del objeto y comprobar que el resultado se repetía una y otra vez se decidió realizar una última prueba. Esta prueba consistió en probar el estiramiento de cuerdas tendinosas. Una vez más se aprovechó el Thorus para crear un modelo simplificado con diversas cuerdas. A dichas cuerdas se le añadieron puntos de referencia que sirven para el estiramiento de las mismas.

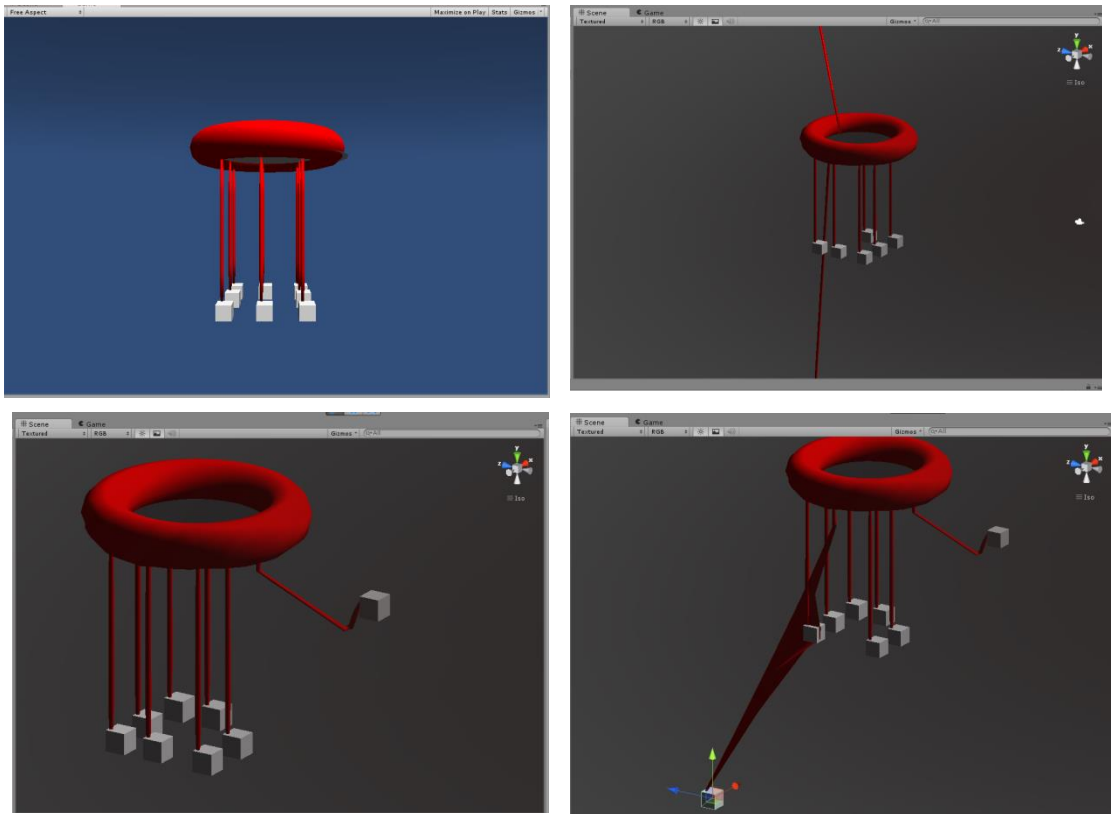


Figura 80: Estiramiento de cuerdas

El resultado es el siguiente:En la figura 80 se puede observar el estiramiento de las cuerdas y el extraño efecto visual que vuelve a repetirse cuando un RigidBody se interpone en el camino, produciéndose el efecto pegajoso antes mencionado, aunque el estiramiento por sí mismo no presenta un comportamiento extraño.

9.1.4 Blender Game Engine

9.1.4.1 Pruebas

Este apartado describe las primeras pruebas realizadas con Blender. No incluyen código, sino que consistieron en investigar las distintas opciones disponibles con BGE. Estos son algunos de los ejemplos conseguidos.

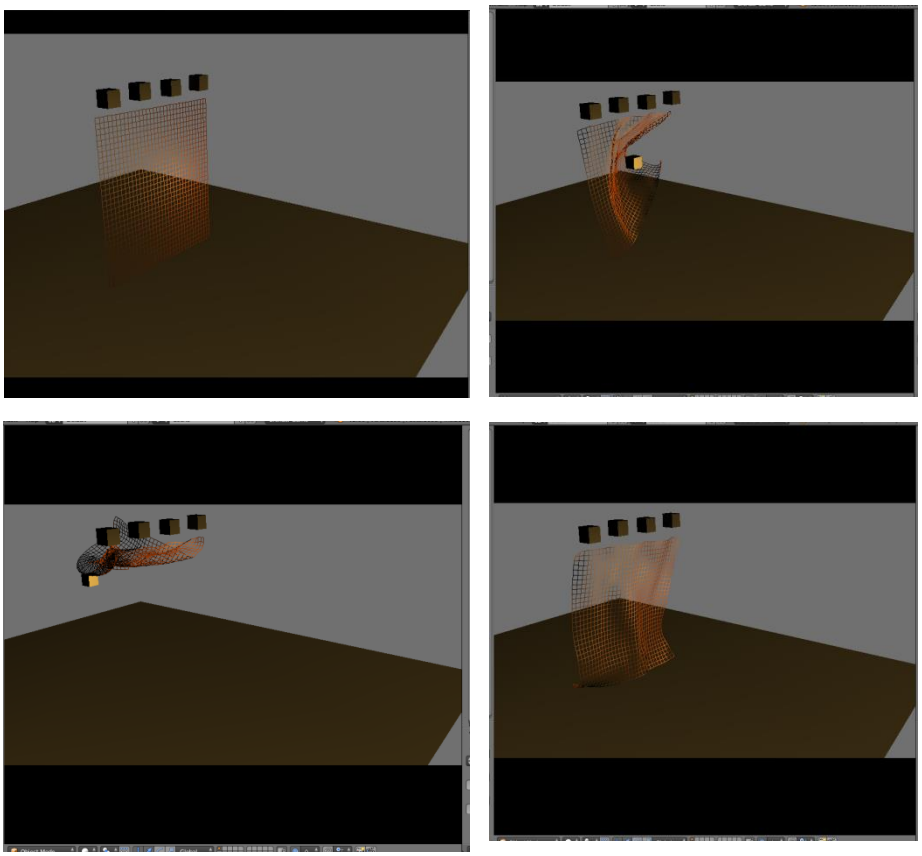


Figura 81: Cortina SoftBody

La figura 81 muestra una cortina SoftBody anclada a cuatro objetos suspendidos en el aire mientras que un quinto objeto se desplaza horizontalmente y provoca una deformación en la misma. Podemos ver como el efecto visual es muy realista y observamos como la cortina vuelve a su estado de reposo original una vez ha sido atravesada por el objeto en movimiento.

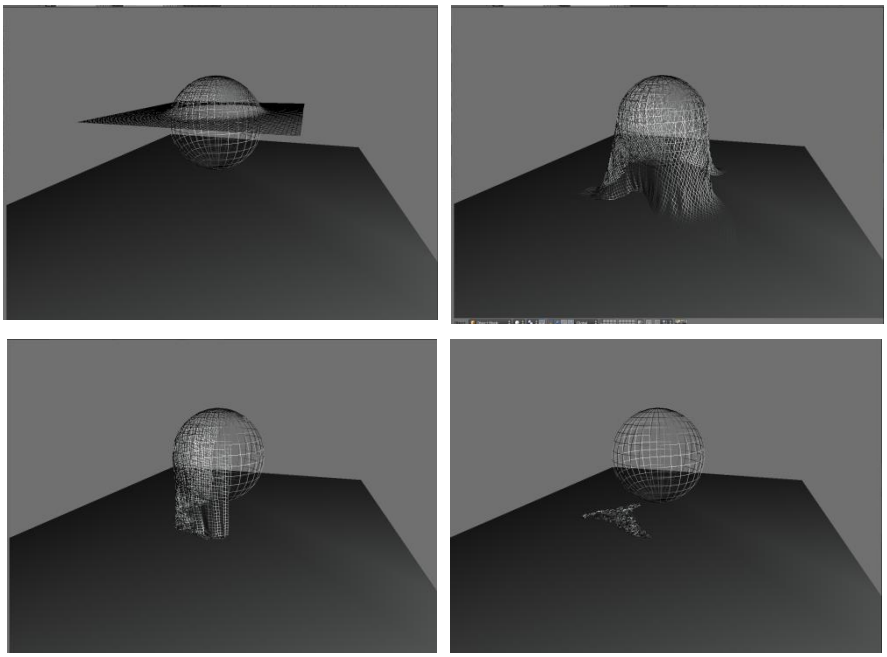


Figura 82: Tela vs esfera

En la figura 82 vemos otra prueba en la que observamos una tela libre caer por efecto de la gravedad sobre una esfera estática y vemos como la tela se deforma para adaptarse a la forma de la esfera para posteriormente caer a un lado.

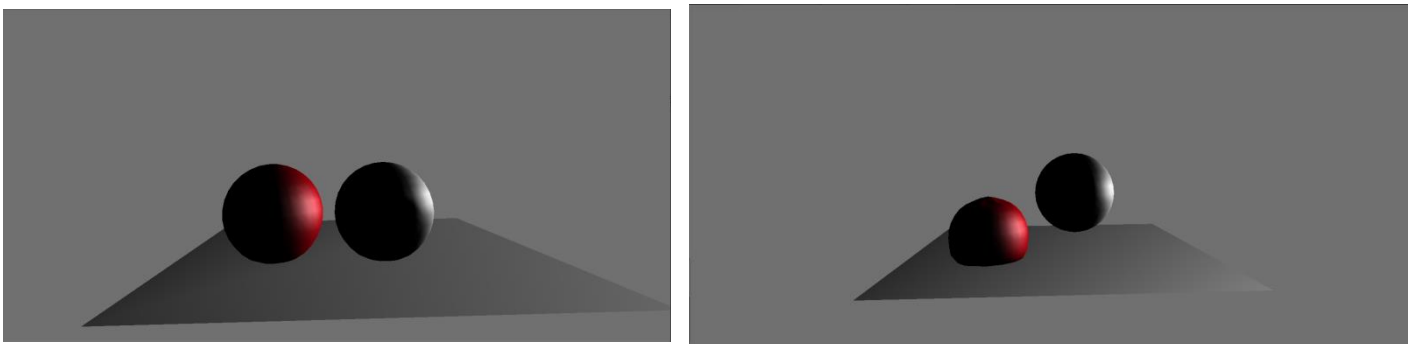


Figura 83: Comparación de SoftBodies

La prueba que se ve en la figura 83 compara dos esferas *SoftBodies* impactando contra un plano. La diferencia entre ambas radica en la preservación de volumen. La esfera blanca está configurada para mantener su forma mientras que la esfera roja está configurada para no mantenerla. No se observan efectos visuales extraños como con *Unity* 3D.

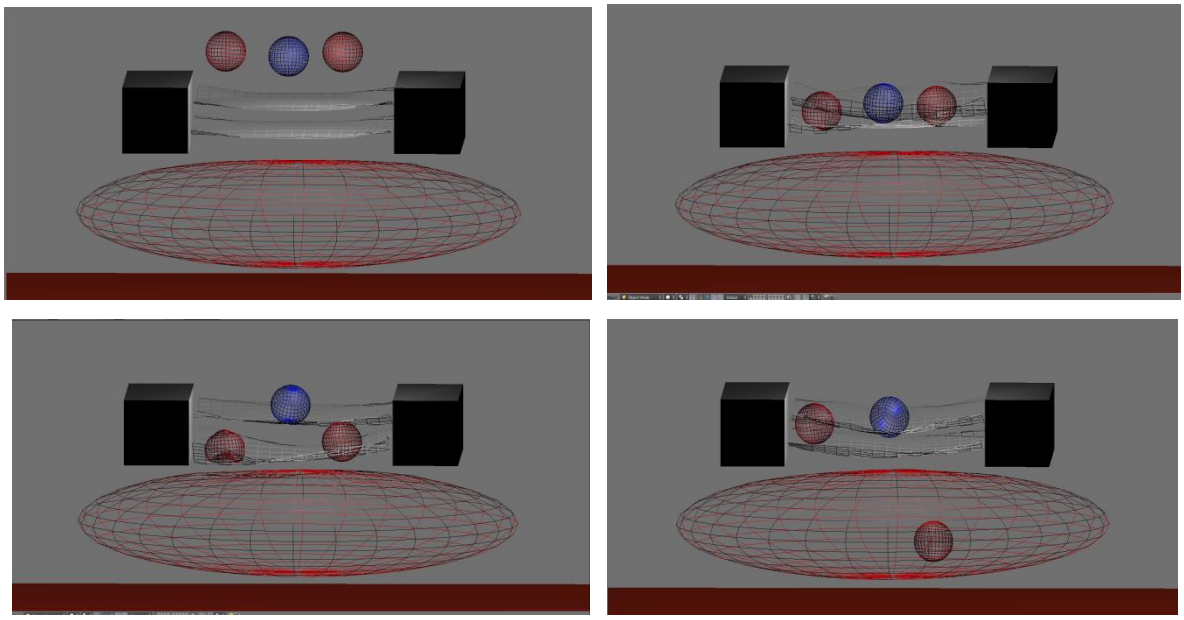


Figura 84: Interacción entre SoftBodies

La figura 84 muestra una prueba de colisión entre SoftBodies configurados de formas distintas. En este escenario hay 2 planos anclados a modo de red y una esfera roja a modo de colchón. Mientras tenemos 3 esferas suspendidas que al iniciar la simulación se dejan caer. La esfera azul es un RigidBody. La esfera roja de la izquierda es un SoftBody que detecta la colisión entre SoftBodies y la esfera de la derecha no detecta colisiones con otros SoftBodies.

Como se puede observar, la esfera azul queda suspendida en la primera tela, mientras que la esfera de izquierda a pesar de tener configurada la colisión con otros SoftBodies atraviesa la primera y queda atrapada en la segunda. Esto se debe a que la primera tela tiene desactivada la colisión con otros SoftBodies, mientras que la segunda tiene esta opción activada. Esta opción puede ser útil para ahorrar al motor de físicas cálculos innecesarios.

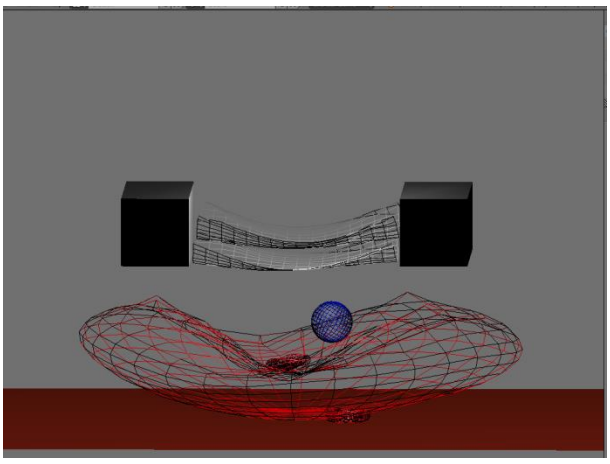


Figura 85: Nivel bajo de substeps

Otra opción a tener en cuenta, son los substeps, un parámetro de configuración que afecta al motor de físicas y que altera la calidad del cálculo de las deformaciones para obtener un mayor rendimiento o un mayor realismo. Además un número bajo de substeps puede provocar errores en los cálculos de

deformaciones y colisiones provocando que un objeto atraviese a otro cuando no debería, por ejemplo la figura 85 es una simulación con el número mínimo posible de *substeps*. Podemos ver como todos los objetos atraviesan las telas y caen directamente sobre el colchón.

10.2 Anexo 2

Capa de presentación:

BGUIMenu

BGUIMenu::_end_game(widget)

Pre: -

Post: Finaliza la ejecución de la aplicación.

BGUIMenu::_destroy_menu(widget)

Pre: widget es un componente de GUI visible.

Post: widget es un componente de GUI destruido.

BGUIMenu::_confirmation_dialog(widget)

Pre: widget es un componente de GUI visible

Post: Se crea un componente de GUI que pregunta al usuario si quiere confirmar una acción determinada.

BGUIMenu::_changeToScene(scenename)

Pre: -

Post: La aplicación cambia desde la escena actual hasta la escena con nombre scenename

TopBarMenu

TopBarMenu:: __on_click_menubutton(widget)

Pre: -

Post: Se muestra en pantalla un menú desplegable y muestra las opciones de Guardar,Salir del programa y Mostrar Controles.

TopBarMenu:: __return_main_menu(widget)

Pre: La escena actual no es el menú principal.

Post: Se sustituye la escena actual por la del menú principal.

TopBarMenu:: __on_button_click(widget)

Pre: widget es un componente de GUI tipo botón y su atributo de nombre indica una escena nueva.

Post: Se sustituye la escena actual por la escena indicada por el nombre del botón widget.

TopBarMenu::__replay_name()

Pre: Se está creando una repetición.

Post: Se solicita al usuario un nombre para la repetición creada. Se detiene el proceso de captura de imágenes para la repetición.

TopBarMenu:: __rename_replay(widget)

Pre: En la carpeta de replays existe una replay nueva cuyo nombre está almacenado como atributo en la instancia de CtrlReplay asociada a Logic.

Post: Se sustituye el nombre que tenía la repetición por la indicada a través del campo de texto de widget.

TopBarMenu:: topBarLayOut(cont)

Pre: Debe existir la carpeta themes/default junto a los ficheros menu.png, record.png, simulate.png, stop.png, inspector.png, edit.png, camara.png, theme.cfg y /fonts/myfont.otf.

Post: Se muestra por pantalla el Menu denominado TopBar con un tema gráfico almacenado en la carpeta themes/default.

TopBarMenu::__changepressure(widget)

Pre: -

Post: Se capturan los valores de las presiones a modificar.

TopBarMenu::__pressurepanel (widget)

Pre: -

Post: Se muestra al usuario el panel de modificación de presiones en el modo simulación.

TopBarMenu:: __showkeyconfig (widget)

Pre: -

Post: Se muestra al usuario el panel informativo de teclas de control.

TopBarMenu:: __reestablish (widget)

Pre: -

Post: La válvula mitral vuelve a su estado de configuración inicial.

TopBarMenu:: __select_sav (widget)

Pre: -

Post: Se captura la selección de la lista de partidas guardadas.

TopBarMenu:: __load_saving (widget)

Pre: -

Post: Se carga la configuración guardada.

TopBarMenu:: __load_menu (widget)

Pre: -

Post: Se muestra al usuario el panel de partidas guardadas.

TopBarMenu:: __save_config (widget)

Pre: -

Post: Se guarda la configuración actual en un fichero de nombre indicado en el campo text del widget.

TopBarMenu:: __save_menu (widget)

Pre: -

Post: Se muestra el panel de guardado de configuraciones.

MainMenu

MainMenu:: __on_button_click(widget)

Pre: widget es uno de los botones del menú principal

Post: Se ejecuta una opción determinada en función del atributo name de la escena. Sirve para cambiar de escena

MainMenu:: mainMenuLayOut(cont)

Pre: Debe existir la carpeta themes/default junto a los ficheros menu.png, record.png, simulate.png, stop.png, inspector.png, edit.png, camara.png, theme.cfg y /fonts/myfont.otf.

Post: Se muestra por pantalla el Menu denominado MainMenu con un tema gráfico almacenado en la carpeta themes/default.

Scripts

CameraFunctions

ModifyZoom()

Pre: Se ha pulsado el botón + o – o bien se ha movido la rueda del ratón arriba o abajo.

Post: Se modifica el valor de zoom del atributo cam de la clase Logic.

RotateCamera()

Pre: Se detecta la pulsación del botón intermedio del ratón y al mismo tiempo movimiento de ratón.

Post: Se aplica una rotación determinada al atributo cam de la clase Logic en función del movimiento y posición del ratón en pantalla.

PrefabCamera()

Pre: Se ha pulsado una de las teclas siguientes 1, 2, 3,4 del teclado.

Post: Se cambia el punto de referencia de la cámara a alguna de las referencias asociadas a las teclas 1, 2,3 o 4.

SetPrefabCamera(camposnum:string)

Pre: camposnum es un string cuyos valores deben ser campos1, campos2, campos3 o campos4.

Post: Se inicializa en el atributo globalDict de Logic correspondiente al valor de camposnum como True y el resto de alternativas se inicializan a False.

Inspections()

Pre: Se detecta la pulsación del botón izquierdo y a la vez movimiento del ratón.

Post: La cámara se desplaza a lo largo del vector que existe entre el punto de referencia actual y el centro de los velos de la válvula para inspeccionar las cuedas.

ChangeReference()

Pre: Se detecta la pulsación de la tecla ctrl.

Post: Se itera entre los distintos puntos de referencia de la válvula.

ImageVideoFunctions

TakeScreenshots()

Pre: Se detecta la pulsación de la tecla F1. Debe existir la carpeta denominada screenshots.

Post: Se crea una nueva screenshot en la carpeta screenshots.

makeAReplay ()

Pre: Debe existir la carpeta de replays.

Post: Se comienzan a capturar imágenes en la carpeta replays/tmp.

ValvulaMitralFunctions

SelectCord()

Pre: Se ha pulsado cualquier botón del ratón. En caso de haber cuerda seleccionada, sus atributos canPick y canFix deben estar inicializados a True.

Post: Si no hay cuerda seleccionada, el valor de currentpivot se inicializará al valor del número de cuerda asociada a la cuerda que se quiere seleccionar. En caso contrario currentpivot pasará a valer -1 para indicar que no hay cuerda seleccionada.

KBMoveCord()

Pre: Debe haber una cuerda seleccionada. Se detecta la pulsación de las teclas w,a,s,d,q,e.

Post: La cuerda seleccionada ha cambiado de posición.

SelectHingePoint()

Pre: Debe haber una cuerda seleccionada. Debe haberse activado el modo de selección de punto de anclaje.

Post: Se ha cambiado el punto de por donde se estira una cuerda.

ChangeHingePoint()

Pre: Debe haber una cuerda seleeccionada. Se detecta la pulsación de la tecla tabulador.

Post: Se cambia el modo de selección a modo de selección de punto de anclaje.

RestartWithNewConfig()

Pre: El modo de selección de punto de anclaje está activado. Se detecta la pulsación de la tecla barra espaciadora.

Post: Se reinicia la simulación modificando el anclaje entre la cuerda y la válvula mitral.

CanFix()

Pre: Debe haber una cuerda seleccionada.

Post: Se modifica el valor de canPick de la cuerda seleccionada en función de si está sobre la superfice de un músculo papilar o no.

QuitarTapa()

Pre: La escena actual es la escena de Inspección.

Post: Hace visible ó invisible el tabique interventricular.

SetPresionValvular()

Pre: La escena actual es la escena de Simulación.

Post: Se aplican fuerzas alternas sobre la válvula mitral que simulan ser la presión ventricular y auricular.

ResetHingePos ()

Pre: Hay una cuerda seleccionada

Pos: Su posición vuelve a ser la posición inicial del programa.

ChangePressures ()

Pre: -

Post: Se modifica el valor de las presiones auriculares y ventriculares.

GlobalDictFunctions

StoreInGlobalDict(fieldname:string,value):

Pre: -

Post: Se almacena en la estructura globalDict el atributo value, con el índice de string fieldname.

StoreConfigToFile (filename:string,rename:bool)

Pre: -

Post: Se almacena en un fichero de nombre filename el campo fieldname de la globalDict.

DeleteCordConfig(filename:string)

Pre: El fichero de nombre filename existe.

Post: Se llama a la función de CtrlCordConfig que elimina un fichero .sav determinado.

LoadFileToGlobalDict(filename)

Pre: El fichero con nombre filename existe.

Post: Se carga en la globalDict los atributos almacenados en el fichero.

BGEObjectsFunctions

GetSensor(nomsensor:string)

Pre: El sensor con nombre nomsensor debe existir.

Post: Se retorna

GetOwner():Object

Pre:

Post: Devuelve la instancia del objeto de blender que tiene asociado el controlador actual, en referencia al script utilizado.

GetObjects(objname): [0..*]:Object

Pre: objname es un string.

Post: Devuelve la lista de objetos que incluye en su nombre el substring objname.

AddObject(objtoadd:string,objdest:BGEObject)

Pre: El objeto objdest es un objeto que se encuentra en una capa active de la escena actual del programa. Objtoadd es el string del nombre de un objeto ubicado en una capa inactiva de Blender.

Post: El objeto con nombre objtoadd se añade a la escena activa en la ubicación de objdest.

EndObject(object:BGEObject)

Pre: El objeto object existe y se encuentra en una escena activa de la aplicación.

Post: El objeto object ya no existe.

ChangeColor(obj:BGEObject,color:[float,float,float,float])

Pre: El objeto obj es un objeto de Blender que está en una capa activa y existe. Color es una array de 4 componentes que tiene el formato RGBAlfa.

Post: El obj cambia de color al color indiciado.

VecFunctions

GetVector(destiny:Vector,origen:Vector,normalized:bool):Vector

Pre: Los parámetros destiny y origen son vectores de 3 posiciones creados con el módulo Vector.

Post: Devuelve el vector entre destiny y origen. Si normalized es indicado como True, entonces el vector estará normalizado.

GetDistance(ob1:BGEObject,ob2: BGEObject):int

Pre: Ob1 y ob2 son objetos existentes en una capa activa de la aplicación.

Post: Devuelve la distancia entre dos objetos de escena.

SceneController

InitProgram()

Pre: -

Post: Se inicializa el programa configurando los atributos rep,shots y conf de la clase Logic.

StartScene()

Pre: -

Post: Se crea la instancia cam y valve de la clase Logic, condicionados según la escena que se esté inicializando.

SetGravity(value:float)

Pre: value es un float.

Post: Se modifica el valor de la aceleración de gravedad dela escena actual.

GetMouseSensor(sensorname:string): events

Pre: sensorname es un string que representa una tecla del ratón.

Post: Se devuelve un sensor, asociado al string indiciado.

isMouseMoving():bool,bool

Pre: -

Post: Devuelve dos booleanos para indicar si el ratón se está moviendo en el eje de las X o en el de las Y.

GetMousePosition(): int,int

Pre: -

Post: Devuelve la posición actual del ratón.

ShowMouse()

Pre: -

Post: Permite que el puntero del ratón sea visible mientras la aplicación esté activa.

ReStartScene()

Pre:

Post: La escena actual vuelve a reiniciarse.

ReestablishValve():

Pre:-

Post: Se invalida el atributo DataStored para que se ignoren los valores almacenados en globalDict y se configure la válvula con las posiciones originales.

Camera

Camera::__SetZoom(zoom:float)

Pre: zoom es un float

.

Post: Se modifica el valor de zoom de la cámara.

Camera:: GetMainPosition(): int,int,int

Pre: -

Post: Devuelve la posición actual de la cámara.

Camera::GetBasePosition():int,int,int

Pre: -

Post: Devuelve la posición por defecto del Target de la cámara.

Camera:: GetActualRef():int

Pre: -

Post: Devuelve el valor que representa el punto de referencia al que está asociada la cámara.

Camera::SetActualRef(val:int):

Pre: val es un int.

Post: Se actualiza el valor del punto de referencia asociado a la cámara.

Camera::GetRRefPosition():int,int,int

Pre: -

Post: Devuelve la posición de la referencia derecha.

Camera:: GetZoom():int

Pre: -

Post: Se retorna el valor del zoom de la cámara.

Camera:: __GetReferenceByIndex (reference:int)::BGEObject

Pre: I es un int.

Post: Se retorna una de las referencias listadas.

Camera:: GetTargetPosition():int,int,int

Pre: -

Post: Se retorna la posición del target de la cámara.

Camera:: SetTargetOrientation (ori: [][]int)

Pre: -

Post:

Camera:: GetTargetOrientation():[][]int

Pre:-

Post: Retorna la matriz de transformación que representa la orientación del target de la cámara.

Camera:: Zoom(valor:float)

Pre: valor es un int.

Post: El zoom de la cámara se ha incrementado.

Camera:: RotateCamera(mposx:float,mposy:float,sensm:float,mx:float,my:float)

Pre: mposx,mposy,mx,my son floats,sensm es un sensor de Blender.

Post: La orientación del target de la cámara ha sido modificado y como resultado ha cambiado la orientación de la cámara.

Camera::ApplyRotation(rotation:[float,float,float],,axis: :[float,float,float])

Pre: rotation es un vector de 3 posiciones, axis es un vector de 3 componentes.

Post: Se aplica la rotación rotation sobre el eje axis al objeto target asociado a la cámara.

Camera::ApplyPrefabCam()

Pre: Alguno de los valores de cámara predefinida están inicializados

Post: Se aplica la cámara predefinida correspondiente al valor indicado en la globalDict de la clase Logic.

Camera:: Inspection(sens: BGEOBject,sens2: BGEOBject,mx:float,my:float)

Pre: mx y my son floats. Sens y sens2 son sensores asociados al periférico ratón.

Post: Desplaza el target asociado a la cámara en función del movimiento del ratón.

Camera::ChangeReference(ctrl:BGEOBject)

Pre: ctrl es un sensor asociado a la tecla ctrl del teclado.

Post: El target de la cámara ha cambiado de posición a a otra asociada a un punto de referencia de la lista.

Camera::RelocateCameraCenter(position:[float,float,float],reference:BGEObject)

Pre: position es un vector de 3 componentes, reference es un int.

Post: El objetivo de la cambia de posición.

ValvulaMitral

ValvulaMitral:: GetCuerdaPath():[0..*]BGEObject

Pre: -

Post: Devuelve la lista de CuerdaPaths que indican la base de cada cuerda.

ValvulaMitral:: GetPreReflist():[0..*]Cord

Pre: -

Post: Devuelve la lista de cords asociados a la válvula mitral.

ValvulaMitral:: GetCurrentPivot():int

Pre:-

Post: Devuelve el valor que identifica a la cuerda seleccionada actualmente.

ValvulaMitral:: GetPosition():int,int,int

Pre: -

Post: Devuelve la posición de la válvula mitral.

ValvulaMitral:: GetSpawner():

Pre:-

Post: Retorna el objeto spawner asociado a la válvula mitral.

ValvulaMitral:: GetUnSelectedCordColor():[float,float,float,float]

Pre:-

Post: Retorna el color de cuerda no seleccionada

ValvulaMitral:: GetSelectedCordColor():[float,float,float,float]

Pre:-

Post: Retorna el color de cuerda seleccionada

ValvulaMitral:: GetTemporalCordColor():[float,float,float,float]

Pre:-

Post: Retorna el color de cuerda temporal seleccionada.

ValvulaMitral:: SetVentricPressure (val:float)

Pre:-

Post: Actualiza el valor del atributo __vpressure

ValvulaMitral:: GetVentricPressure ():float

Pre: -

Post: Retorna el valor del atributo __vpressure

ValvulaMitral:: SetAuricPressure (val:float)

Pre: -

Post: Se actualize el valor del atributo __appressure

ValvulaMitral:: GetAuricPressure ():float

Pre: -

Post: Retorna el atributo __appressure

ValvulaMitral:: GetCanPick ():bool

Pre:-

Post: Retorna el valor del atribuo __canPick

ValvulaMitral:: SetCanPick (value:bool)

Pre: value es un atribuo opcional

Post: Se modifica el valor del atributo __canPick

ValvulaMitral:: GetCanFix ():bool

Pre:-

Post: Retorna el atributo __canFix

ValvulaMitral:: ResetHingePos (obj:BGEObject)

Pre:-

Post: La posición de la cuerda obj vuelve a ser la que tenía al arrancar el programa.

ValvulaMitral:: GetTmpCurrentpivot():int

Pre:-

Post: Devuelve el valor del temporal current pivot.

ValvulaMitral:: SetCurrentpivot(val:int)

Pre: val es un int.

Post: Se actualiza el valor del atributo currentpoint.

ValvulaMitral::GetCurrentPivot():int

Pre:-

Post: Retorna el valor de currentpoint, que indica cuando una cuerda ha sido seleccionada,-1 en caso contrario.

ValvulaMitral:: GetCurrentCord():BGEObject

Pre: -

Post: Retorna la cuerda seleccionada. Retorna -1 en caso de que no la haya.

ValvulaMitral:: GetCord(numcord): BGEObject

Pre: numcord es un int.

Post: Retorna una cuerda cualquiera dado su identificador.

ValvulaMitral:: SetPressure():

Pre: -

Post: Cambia el valor booleano del atributo pressure negándolo.

ValvulaMitral::GetPressure():bool

Pre: -

Post: Retorna el valor booleano del atributo pressure.

ValvulaMitral::ToggleBlood()

Pre:

Post: Cambia el valor booleano del atributo blood negándolo.

ValvulaMitral::GetBloodFlag():bool

Pre:-

Post: Devuelve el valor del atributo blood

ValvulaMitral::EditCordData(obj,typev,value)

Pre: obj es una Cuerda de la válvula mitral, typev es un string que indica que el campo que se va a actualizar, value es un atributo opcional externo, usado cuando esta información no la contiene el objeto cord.

Post: Se ha actualizado la información de la cuerda en el atributo globalDict de la clase Logic.

ValvulaMitral::SetPresionValvular(forcelist:[*]:Vector,local:bool)

Pre: El programa está ejecutando el modo de Simulación.

Post: Se aplican todas las fuerzas almacenadas en la lista de fuerzas a la válvula mitral.

ValvulaMitral::GetNextCordStep(oblist:[*]:BGEObject,ob:BGEObject,value:int)

Pre: El programa está ejecutando el modo de edicón

Post:

ValvulaMitral:: MoveHinge(obj:BGEObject,direction:Vector)

Pre: obj es un Cord de la válvula mitral válido, direction es un vector de dirección

Post: El obj se desplaza una unidad de Blender en la dirección indicada.

ValvulaMitral::__GetValve():BGEObject

Pre: -

Post: Retorna el objeto de Blender que representa la válvula mitral.

ValvulaMitral::__GetEditRef():[0..*]BGEObject

Pre: -

Post: Obtiene la lista de referencias, utilizadas para la edición del punto de anclaje de una cuerda.

ValvulaMitral::__StartInspection()

Pre: -

Post: Configuración de válvula para inspección. Eliminación de elementos auxiliares innecesarios.

ValvulaMitral::__StartEdition()

Pre: -

Post: Configuración de válvula para edición. Se crean los vínculos entre las cords y la válvula mitral. Se deshabilita la presión valvular se coloca la cámara en una posición predefinida para la edición de las cuerdas.

ValvulaMitral::__StartSimulation()

Pre: -

Post: Configuración de la válvula para modo de simulación. Se crean los vínculos entre los cords y la válvula. Se eliminan atributos innecesarios.

ValvulaMitral::__DeletePaths()

Pre: La escena no se encuentra en el modo de Edición.

Post: Eliminación de los puntos base de las paths de las cuerdas.No son necesarios en modo simulación e inspección.

ValvulaMitral::__LinkPaths()

Pre: No se está ejecutando la escena de inspección.

Post: Crea los vínculos físicos entre los puntos de referencia temporales y la válvula mitral.

ValvulaMitral::__LinkHinges()

Pre: No se está ejecutando la escena de inspección.

Post: Crea los vínculos físicos entre las cords y la válvula mitral.

ValvulaMitral::__SetRefVisibles()

Pre: Se está ejecutando la escena de modo de Edición.

Post: Permite que las cords sean visibles al usuario.

ValvulaMitral::__SetJoint(pivot:BGEObject,valve:BGEObject):Constraint

Pre: pivot es un objeto de Blender de tipo Rigid Body, valve es un objeto de blender de tipo RigidBody o SoftBody.

Post: Se crea un vínculo físico entre ambos objetos, lo cual hace que la fuerza que afecte al primer objeto afectará también al otro.Este método retorna el constraint físico creado.

ValvulaMitral::__SetCordPath(cord:BGEObject)

Pre: El programa está ejecutando el modo de edición. Cord es un objeto de Blender asociado a la válvula mitral.

Post: Se crea la lista de puntos de referencia temporales entre el cord y la base de la cuerda.

ValvulaMitral::__GetPath(numcord):BGEObject

Pre: El programa está ejecutando la escena de Edición. Numcord es un int

Post: Devuelve el punto base de la cuerda actual, que limita la longitud de una cuerda.

ValvulaMitral::__RemoveCordPath

Pre: -

Post: Eliminación de los puntos de referencia temporales.

CtrlReplay

CtrlReplay:: SetCreateVid()

Pre: -

Post: Modifica el valor del atributo createVid negándolo.

CtrlReplay::GetCreateVid():bool

Pre:-

Post: Retorna el valor del atributo createVid.

CtrlReplay::SetReplayname()

Pre: -

Post: Asigna un nombre por defecto a una repetición recién creada. El formato por defecto es "SimulacionValvula-Dia-Hora-Minuto-Segundo".avi

CtrlReplay::GetReplayname():string

Pre:-

Post: Retorna el atributo __replayname

CtrlReplay::TakeVideoImages(sens:BGEObject)

Pre: sens es un sensor de tiempo que se activa en cada frame de ejecución. El programa está ejecutando el modo Simulación. La carpeta replays, existe.

Post: En función de la activación del sensor, se capturan inicia el modo de captura de imágenes. Si no existe la carpeta tmp se crea.

CtrlReplay::BuildVideo(ruta:string)

Pre: Existe la carpeta replays.

Post: Se crea un video con el nombre por defecto con las imágenes generadas.

CtrlGallery

CtrlGallery::UserScreenshots(sens:BGEObject):

Pre: sens es un sensor asociado a una tecla. La carpeta screenshots existe.

Post: Se crea una nueva captura de pantalla.

CtrlGallery::makeScreenshots(ruta:string):

Pre: La carpeta screenshots existe.

Post: Se crea una screenshot nueva.

CtrlCordsConfig

CtrlCordsConfig::SaveCordConfig(prereflist:[*]string,filename:string,rename:bool)

Pre: Existe la carpeta saves. La lista prereflist no está vacia.

Post: Se almacenan los datos en un fichero.sav con nombre filename. Si rename vale True se reescribe el fichero en caso de que ya exista.

CtrlCordsConfig::DeleteCordConfig(filename:string)

Pre: El fichero de nombre filename existe

Post: El fichero de nombre filename ya no existe.

CtrlCordsConfig::LoadCordConfig()

Pre:

Post: Retorna la lista de partidas guardadas.

CtrlCordsConfig::LoadCordConfig(filename:string)

Pre: -

Post: Se carga la configuración del fichero filename en la glodalDict.

9.3 Anexo 3

Acrónimos

APM = Músculo papilar anterior

PPM = Músculo papilar posterior

EFE = Enfermedad fibroelástica

EB = Enfermedad de Barlow

RB = Rigid Bodies

SB = SoftBodies

LB = Logic Bricks

NURB = Non-Uniform Rational B-spline

BGE = Blender Game Engine

CSS = Cascading Style Sheets

10. Bibliografía

[1] Centro de Referencia en reparación de válvula mitral, Monte Sinai:
http://www.reparacionvalvularmitral.org

[2] El blog de cardiología: http://www.elblogdecardiologia.com/tag/valvula-mitral

[3] Medicina UFM, Enfermedad de Barlow:
http://medicina.ufm.edu/index.php/Enfermedad_de_Barlow

[4] Proyección perspectiva vs ortogonal:
http://guti29leber.blogspot.com.es/2012/09/proyeccion-opengl-ortogonal.html

[5] Vpython: http://www.vpython.org/contents/doc.html

[6] Zapata, D. L. (2011, 5). Fisiología cardiovascular:
http://drleaz.wordpress.com/2011/04/05/fisiologa-cardiovascular-clase-4/

[7] Fisher, M. Cloth and Mass-spring model:
http://graphics.stanford.edu/~mdfisher/cloth.html

[8] Ramis, D. L. (2011). El blog de la válvula mitral:
http://elblogdelavalvulamitral.blogspot.com.es/

[9] Ascher, E. B. (2004). *Decomposing Cloth*. *Abstract Decomposing Cloth*

[10] Cotin, M. B.-N. *Real-time Volumetric Deformable Models for Surgery Simulation using Finite Elements and Condensation*. *Computer Graphics Forum(1996)*

[10] Wong, G. B.-K. (2002, 11). *Hardware-Assisted Self-Collision for Deformable Surfaces*. VRST 2002 Proceedings of the ACM symposium on Virtual reality software and technology(2002)