

Escuela Superior de Informática

Universidad de Castilla-La Mancha

CURSO DE EXPERTO EN DESARROLLO

DE VIDEOJUEGOS

TRABAJO FIN DE CURSO

MAKEMAKE

JULIO 2016



Escuela Superior de Informática

Universidad de Castilla-La Mancha

CURSO DE EXPERTO EN DESARROLLO

DE VIDEOJUEGOS

TRABAJO FIN DE CURSO

MAKEMAKE

Calzado Naranjo, Jesús Pedro

Muñoz Fraile, Pamela

JULIO 2016

CURSO DE EXPERTO EN DESARROLLO DE VIDEOJUEGOS

**Calificación Trabajo Fin de Curso**

CONVOCATORIA: Julio 2016

TÍTULO DEL PROYECTO: MAKEMAKE

AUTORES (ORDEN ALFABÉTICO):

CALZADO NARANJO, JESÚS PEDRO

MUÑOZ FRAILE, PAMELA

TRIBUNAL:

Presidente: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Vocal: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Secretario: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

FECHA DE DEFENSA: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CALIFICACIÓN: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |
| --- | --- | --- |
| PRESIDENTE | VOCAL | SECRETARIO |
| Fdo: | Fdo: | Fdo: |

**Índice**

[1. Resumen 9](#_Toc455363095)

[2. Introducción 11](#_Toc455363096)

[3. Objetivos 13](#_Toc455363097)

[**1.** **Objetivos funcionales.** 13](#_Toc455363098)

[**2.** **Objetivos no funcionales** 13](#_Toc455363099)

[4. Arquitectura de la solución 15](#_Toc455363100)

[**1.** **Metodología** 15](#_Toc455363101)

[**2.** **Diagramas de clase** 15](#_Toc455363102)

[**3.** **Patrones** 15](#_Toc455363103)

[**4.** **Herramientas** 16](#_Toc455363104)

[**5.** **Lenguajes de programación** 16](#_Toc455363105)

[**6.** **Bibliotecas** 17](#_Toc455363106)

[**7.** **Proceso de desarrollo** 17](#_Toc455363107)

[a. Diseño 17](#_Toc455363108)

[b. Game concept 17](#_Toc455363109)

[c. Diseño de solución software. 19](#_Toc455363110)

[d. Implementación 19](#_Toc455363111)

[i. main.cpp 22](#_Toc455363112)

[ii. GameManager.cpp 22](#_Toc455363113)

[iii. GameState.h 23](#_Toc455363114)

[v. PlayState.cpp 24](#_Toc455363115)

[vi. ReplayState.cpp 25](#_Toc455363116)

[vii. PauseState.cpp 26](#_Toc455363117)

[viii. EndGameState.cpp 26](#_Toc455363118)

[ix. InputManager.cpp 27](#_Toc455363119)

[x. Actor.cpp 28](#_Toc455363120)

[xi. MakeCamera.cpp 29](#_Toc455363121)

[xii. MyMotionState.cpp 29](#_Toc455363122)

[xiii. Stage.cpp 30](#_Toc455363123)

[xiv. StagesManager.cpp 30](#_Toc455363124)

[xv. SoundFX.cpp 31](#_Toc455363125)

[xvi. SoundFXManager.cpp 31](#_Toc455363126)

[5. Análisis de costes 33](#_Toc455363127)

[6. Manual de usuario 35](#_Toc455363128)

[7. Conclusiones y trabajo futuro 39](#_Toc455363129)

[1. Conclusiones 39](#_Toc455363130)

[2. Trabajo Futuro. 39](#_Toc455363131)

[8. Bibliografía 43](#_Toc455363132)

# Resumen

*MakeMake* es un juego de plataformas tridimensional en el que un personaje, manejado por el jugador mediante un mando de control con “stick” analógico, debe completar numerosos niveles.

El juego se compone de varios mundos y cada mundo de varios niveles. Cada nivel consta de varios puzles para conseguir el pase al siguiente nivel, logrado al alcanzar la única meta por nivel que existe. De este modo, el jugador irá desbloqueando todos los niveles que componen el juego.

Para completar cada nivel, el personaje principal puede interactuar con distintos elementos que aparecerán en pantalla y que le permitirán avanzar.

El jugador puede, mediante el mando de juego, controlar al personaje y además, controlar la cámara de visión. Esto le será muy útil a la hora de descubrir secretos en cada nivel y calcular distancias de objetos. Así pues, el jugador puede rotar la cámara alrededor del nivel y conseguir una visión más clara del puzle que se le propone.

Además de todo esto, el personaje principal cuenta con habilidades como el salto y el arrastre de objetos de tamaño similar a él.

Con este proyecto final se persigue la puesta en escena de todas las habilidades obtenidas durante la duración del curso, para conseguir desarrollar de forma satisfactoria un videojuego completo, entretenido, vistoso y con posibilidad de extensión de contenidos. Además se ha pretendido poner hincapié en la parte técnica, tanto en la arquitectura como en el desarrollo de soluciones para determinadas características.

Por tanto, para el desarrollo de este juego se ha optado por el motor de render Ogre3D, de libre uso y programado en C++. Además se han usado otras librerías para el manejo de sonidos, interfaces de entrada/salida y cálculos físicos así como herramientas de software libre para diseño de elementos 2D y 3D.

# Introducción

El presente documento refleja y describe el proceso de elaboración y desarrollo del juego MakeMake. En este se explicarán todas las fases y retos encontrados durante su tiempo de creación. A continuación se puede encontrar una lista indicando qué encontrará a lo largo de los apartados siguientes.

* **Objetivos**: Antes de comenzar un proyecto es necesario fijar una meta. Un lugar o estado a donde llegar, y que al conseguirlo se dé por concluido el trabajo. Lejos de ser una cosa nimia es vital.
* **Arquitectura de la solución:**
  + *Metodología*: nombra y explica la metodología seguida durante la fase de desarrollo del proyecto.
  + *Diagramas de clase*: es el diagrama de clase creado para el desarrollo
  + *Patrones*: descripción de los patrones implementados.
  + *Herramientas*: enumeración de las aplicaciones utilizadas durante el desarrollo del presente videojuego.
  + *Lenguajes de programación*: lenguajes principales usados en el desarrollo.
  + *Bibliotecas*: para la consecución del juego ha sido necesario el uso de bibliotecas externas que sirven de gran ayuda a la hora de implementar ciertas características contenidas en el juego.
  + *Proceso de desarrollo*: descripción del proceso de desarrollo del presente proyecto. Dentro de este proceso podemos encontrar las siguientes fases:
  + *Diseño*: desarrollo de la idea, diseño y planificación del videojuego.
  + *Bocetos*. Se crean una serie de diseños a mano alzada sobre papel y se comenta en equipo.
  + *Game concept*. Se desarrolla la idea inicial dando como resultado un concepto. Una mecánica de juego que intetará resultar divertida y entretenida de jugar y que suponga un reto para el jugador.
  + *Diseño de solución software*. Esqueleto del juego.
  + *Creación de modelos de primera etapa*. Se diseñan los modelos que se cargaran durante el desarrollo del juego. Estos modelos son creados simplemente para el desarrollo del juego y en fases finales se sustituirán por los definitivos.
* **Implementación:**
  + Creación del esqueleto del juego basado en estados. En esta parte se crea las clases GameState y GameManager, piedra angular del videojuego.
  + Desarrollo de mundo de test e inclusión en el esqueleto inicial. Llegados a este punto se puede desarrollar y hacer pruebas de forma simultánea.
  + Desarrollo de personaje principal. En esta parte se desarrolla el manejo del personaje principal del juego, movimiento y salto.
  + Desarrollo de objetos interactivos. Se crean los diferentes objetos con los que se va a interactuar durante el juego y se desarrolla su comportamiento.
  + Implementación de colisiones. Se desarrollan los comportamientos que tendrán determinados elementos del juego cuando colisionan entre si.
  + Implementación de control por mando de Xbox.
  + Desarrollo de línea argumental y pantallas. Se crea un sistema de carga de escenarios a través de archivos de texto. Esto hace que el juego pueda crecer de forma sencilla.
  + Desarrollo de la cámara del juego. Comportamiento de la cámara muy específico. Se mueve entre 4 puntos. Se anima la transición para dar mejor sensación de juego.
  + Desarrollo de pantalla para conocer el avance del jugador. Mediante esta pantalla accedemos a un mapa donde nos muestra de forma clara y sencilla por donde va el jugador en ese momento.
  + Creación de modelos definitivos de objetos interactivos.
  + Creación de animación para personaje principal. Es necesario que el personaje muestre movimiento al caminar.
  + Creación de 4 mundos ambientados de forma diferente. Se toma especial cuidado en determinados efectos visuales e iluminación de la escena acorde a lo que se presenta.
  + Elaboración de texturas. Se crean las texturas finales del juego. En algunos diseños se ha puesto interés en hacer una simulación de iluminación para estar en consonancia con la escena.
  + Creación de sonidos. Se crean los sonidos que empleará el juego. Los sonidos han sido creados con herramientas gratuitas.
  + Inclusión de músicas. Se incluyen finalmente las músicas del juego. Esto hace del juego una experiencia más inmersiva.
  + Creación de diseños de pantallas de menú. Se añade el diseño definitivo de pantalla de “Press Start” y el mapa. Además se añade un diseño para la pantalla final, cuando el jugador concluya el juego.
* **Análisis de costes:** en esta parte se muestra el esfuerzo en horas desglosado por secciones de desarrollo de los integrantes del grupo.
* **Manual de usuario**: sección en la que se explica la historia detrás del juego y como jugar.
* **Conclusiones y trabajo futuro**. En esta sección se exponen una serie de conclusiones que se han sacado tras la elaboración del presente proyecto y una serie de mejoras que podrán ser incluidas en un futuro.
* **Bibliografia**: listado de fuentes consultadas durante la elaboración de este proyecto.

# Objetivos

## **Objetivos funcionales.**

* Creación de sistema de juego con un personaje situado sobre plataformas. Es la base del juego. Es necesario disponer de un personaje que se pueda mover con soltura y jugabilidad por un mundo cargado en pantalla.
* Desarrollar objetos con los que interactuar para conseguir la meta. Para que el juego empiece a complicarse se desarrollan estos elementos. La mayor parte de las veces sirven de ayuda aunque bien es cierto que otras tantas estorban.
* Creación de mundos diferentes, que den impresión de estar bien integrados haciendo uso de iluminación y efectos, empleando luces y texturas. Así pues se crean efectos, como la lava del mundo castillo haciendo uso de texturas procedurales, partículas y efectos de luz
* Desarrollo de sistema de carga y guardado de partida.
* Implementación de control por mando de Xbox.
* Desarrollo de mapa para control de avance del jugador.
* Desarrollo de puzles simples.
* Creación de varias pantallas que componen el juego, en las cuales se incluyen los puzles creados anteriormente.
* Editor de niveles. Aunque no se ha implementado, es por donde debería seguir el proceso de desarrollo. De este modo, la creación de nuevos niveles se agiliza tanto a la hora de elaborarlos como de probarlos y modificarlos.

## **Objetivos no funcionales**

* Jugabilidad a la altura. No es necesario leer un manual para manejar a un personaje en vista subjetiva. El manejo se ha buscado acercar a cualquier juego de plataformas 3D en el mercado. Sencillo y con soltura.
* Curva de dificultad en lugar de tutoriales. En lugar de crear un tutorial para cada objeto hemos optado por la solución que más nos gusta. Jugar y probar. Cada elemento que se presenta en escena ha pasado por una pantalla inicial donde se le ha propuesto al jugador un puzle muy sencillo, el cual para completarlo debe hacer un uso sencillo del objeto presentado. Tras eso, la cosa se ira complicando poco a poco.
* Variedad en ambientes. Se ha puesto especial hincapié en crear mundos diferentes y que gusten visualmente. De este modo, el jugador ira variando entre mundos y la sensación de estar jugando una aventura es mayor.
* Música y sonidos adaptados. Se ha tratado de completar el juego con un conjunto de sonidos y música que puedan enlazar bien entre ellos.
* Puzles y diversión. Creemos fehacientemente que el enfrentarse a determinados puzles puede ser satisfactorio. Si bien algunos son sencillos se ha tratado de buscar dificultad en otros, de tal modo que cuando el jugador lo supere consiga una gratificación por el tiempo dedicado.

# Arquitectura de la solución

## **Metodología**

Para el desarrollo del presente proyecto se ha optado por emplear scrum, debido a la necesidad de hacer entregas periódicas y se adapta bastante bien para el desarrollo en equipo.

El hecho de tener que realizar entregas cada cierto tiempo y tras ellas realizar modificaciones ha hecho que la naturaleza iterativa e incremental de scrum se ajuste a la perfección.

El rol de coordinador del equipo ha sido asumido por Jesus Pedro, teniendo una visión general del producto final a conseguir y organizando las diferentes tareas a abordar y concluir.

Se han realizado reuniones con periodicidad entre los miembros del equipo. De este modo, las tareas definidas han sido puestas sobre la mesa y se ha ido avanzando en ellas. En esas reuniones se han tratado las tareas concluidas y las que quedan por concluir asi como las que están en marcha. Además se ha puesto especial cuidado en solucionar los posibles conflictos entre desarrollos realizados por cada miembro del grupo.

## **Diagramas de clase**

En fases tempranas, se desarrolló un diagrama de clases que fue variando con el tiempo. En primera instancia es difícil acertar completamente con todo lo que se va a necesitar en el desarrollo. Además, los cambios que puedan surgir por funcionalidad añadida o eliminada hacen que el diagrama inicial vaya variando.

El diagrama final de clases se encuentra en el Anexo 1.

## **Patrones**

Para este proyecto se ha buscado la inclusión de patrones de diseño, si bien es cierto, que la exigencia en los plazos y la presión ha hecho que muchas veces se opte por una solución rápida para conseguir una funcionalidad y llegar a tiempo, dejando para una fase posterior un posible rediseño del código que lo haga más amigable para modificaciones y más sencillo de ampliar por parte de miembros futuros en el grupo.

Así pues los patrones que se han usado son los siguientes:

* ***Singleton***

Este patrón ha sido empleado en el uso del motor gráfico. La implementación que ofrece este es muy completa y hace que el desarrollo se acelere bastante.

La justificación de su uso es conseguir que para todos los elementos del juego exista una única instancia y que todos puedan manejar los mismos recursos.

* ***Patrón State***

El patrón estado ha sido implementado en los diferentes estados del juego. Existe una clase superior llamada GameState de la cual parten para su deficion las diferentes clases que definen los posibles estados del juego, que son Juego, Pausa, Intro, Replay, End.

## **Herramientas**

Durante la elaboración del presente proyecto han sido varias las herramientas que se han usado, para varios ámbitos como son el diseño, la programación, retoque gráfico, desarrollo de modelos 3D, documentación, etc.

A continuación se listan y detallan todas ellas.

***Visual Studio C++ 2012***

Visual Studio es un IDE de desarrollo software para diferentes lenguajes de programación. Dado que el motor de render Ogre3D usado está escrito en C++ se ha optado por este mismo lenguaje de implementación.

Además de las diferentes ayudas en línea a la hora de desarrollar, opciones de autocompletado y detección de errores durante la edición, tiene un depurador muy potente. Todas estas herramientas hacen que el proceso de desarrollo se acelere mucho y se cometan menos errores además de que sea más fácil detectarlos y corregirlos.

***Notepad++***

Editor de texto muy útil a la hora de modificar los diferentes archivos de configuración del motor de render y además, muy usado para crear y modificar archivos clásicos de gráficos de Ogre, como partículas, materiales, overlays, etc

***Blender***

Programa de modelado, iluminación y renderizado animación y creación de gráficos tridimensionales. Además de ser muy potente es software libre.

Se ha empleado en el diseño y desarrollo de modelos empleados en el juego, como la plataforma base, el personaje principal con su animación o los objetos interactivos.

***InkScape***

Inkscape es un editor gráfico de vectores muy potente. Gratuito y de código libre se ha usado para la creación de todos los elementos gráficos 2D del juego. Texturas de modelos, y overlays usan diseños creados a través de esta herramienta.

***GIMP***

Es un software de edición de imágenes digitales en formato mapa de bits. En este caso se ha usado para ciertos retoques en gráficos 2D.

***INNO SETUP***

Software utilizado para generar los instalables.

***ASTAH***

Programa utilizado para el diseño UML.

***OBS***

Programa utilizado para realizar la captura del vídeo del juego.

## **Lenguajes de programación**

***C++***

El lenguaje en que se basa el desarrollo completo del juego es C++ dado el hecho de que es el lenguaje que usa el motor de renderizado. En un lenguaje muy utilizado y esto hace que haya una comunidad muy potente y activa alrededor. De tal forma, cuando encuentras un problema, no tardas en solucionarlo buscando en la web. Además son numerosas las bibliotecas que facilitan el trabajo y aceleran el desarrollo.

Además de todo ello, este lenguaje es el más usado, con diferencia, en el mundo del desarrollo de videojuegos.

## **Bibliotecas**

***Ogre3D***

Del acrónimo ingles Object-Oriented Graphics Rendering Engine es un motor de renderizado 3D orientado a escenas. Escrito en lenguaje C++, es un software libre. Facilita el acceso a capas inferiores de librerías graficas como pueden ser Direct3D y OpenGL. Además es bastante extensible por medio de bibliotecas externas.

***Bullet***

Bullet es una potente biblioteca para cálculo de físicas. Esta, usada en conjunto con ogre3D muestra un mundo en tres dimensiones sobre el que actúan fuerzas físicas resultado muy vistoso y real.

***SDL***

Simple DirectMedia Layer es una biblioteca para manejo de sonidos en C++. Ha sido usada para dotar al juego de efectos de sonido y música ambiental.

***OIS***

Librería para control de dispositivos de entrada. Es muy útil a la hora de incluir el control por parte de ratón, teclado o mandos de juego. Esta biblioteca viene incluida en Ogre de partida.

***STL***

Completa librería incluida en C++. Muy útiles resultan las diferentes implementaciones de estructuras de datos, como las listas, usadas en el desarrollo del presente videojuego.

## **Proceso de desarrollo**

### Diseño

En la fase de diseño de juego se han realizado diversas tareas, que resultan importantes para un futuro cómodo de desarrollo y allanar un camino que se puede volver tortuoso y difícil de completar. Precisamente en el sector del videojuego, no son pocas veces las que ocurre que un desarrollo se alarga en demasía e incluso se cancela por problemas de plazos y por mal diseño inicial.

En esta fase se parte de una idea. ¿Qué tipo de juego queremos hacer? Y se barajaron diferentes propuestas, siendo el desarrollo de un juego de puzles con física la que más nos gustó.

Con la idea concreta aceptada, el equipo se puso a crear bocetos para plasmar de manera visual cómo debía ser el juego y hacia que había que poner rumbo. Han sido muchos los dibujos y diagramas realizados y con ellos la comprensión del juego entre los integrantes ha sido plena.

### Game concept

Una vez se tiene idea de qué mostrar en pantalla se crea un Game Concept en el cual se explica detalladamente cual es la mecánica del juego poniendo énfasis en la jugabilidad y en la diversión del mismo. Se busca con insistencia la idea de que jugar a determinados niveles sea todo un reto para el usuario y le incite a seguir intentándolo una y otra vez para conseguir avanzar.

A continuación se detalla la idea de juego.

***Mundos***

El juego constará de mundos de diferente aspecto. Estos mundos serán creados con efectos característicos para dar variedad a la acción y sumergir más al jugador. Sobre los mundos puede haber variaciones en fuerzas físicas o iluminación que complican la consecución de la meta final.

Los mundos están compuestos de niveles.

***Niveles***

Los niveles son los diferentes retos que se muestran de manera unitaria al jugador. En cada momento el jugador se encontrara jugando un determinado nivel y permanecerá en el hasta completarlo. La solución está en el escenario, un mundo de dimensiones limitadas y del que la única manera de escapar es encontrando la meta que te lleva al siguiente nivel.

Cada nivel pertenece en exclusiva a un único mundo, y éste determina qué condiciones aparecen en pantalla.

Una vez creado un nivel, la intención del equipo es hacer que sea muy sencillo modificar el hecho de que un nivel pertenezca a un mundo u a otro.

***Personaje***

El personaje principal será manejado por un mando de control o teclado. de este modo el jugador se desplazará por el mundo en que se encuentra para descubrir posibles soluciones al puzle. Además tiene la habilidad de saltar. El personaje contará con animación a la hora de caminar.

***Cámara***

La cámara se convierte en otro personaje del juego. Es de vital importancia, porque se necesitará usar para descubrir ciertas cosas que no se ven en el punto inicial, además de ser de una gran ayuda en el cálculo de distancias entre objetos que están en diferente posición de profundidad.

***Historia***

Se crea una pequeña justificación de porque el personaje se encuentra con este problema y que debe hacer para solucionarlo. Por tanto se crea un camino lineal desde la pantalla inicial hasta la última pantalla, de tal modo que el personaje no puede volver a pantallas anteriores y únicamente puede avanzar hasta la consecución del último puzle.

***Atrezo***

Se dota a los mundos de varios objetos inertes que darán morfologías distintas a los distintos niveles. Con estos elementos el personaje interactúa colisionando. No puede traspasarlos pero puede hacer uso de ellos para alcanzar niveles de una altura superior a su capacidad de salto. Estos bloques pertenecen al nivel y jamás se destruyen y tendrán un aspecto acorde con el mundo al que pertenecen.

***Objetos interactivos***

Para la consecución de su meta, el personaje puede y debe usar ciertos elementos del escenario que se le presentan. Puede necesitarlos o no, pero son un arma muy útil para pensar en posibles soluciones. Además se puede dotar al nivel de elementos spare, que no se necesitan, pero puedes hacer las labores de despistar al jugador para que no vea tan fácil la solución.

***Meta***

El objetivo del personaje es conseguir llegar a la meta. Una vez en ella se tele-transportará a la siguiente fase. Normalmente la meta estará claramente mostrada, pero puede ocurrir en escenarios que este oculta y haya que buscarla.

### Diseño de solución software.

Una vez se tiene claro que se quiere alcanzar, se puede ir pensando en dar solución de diseño software. Qué clases habría que diseñar, como interactúan unas con otras, y que funcionalidades son deseables, son preguntas que hay que responder en este momento del desarrollo. No obstante, no hace falta ser muy explícito, ya que los continuos cambios que se esperan en un desarrollo de este tipo puede hacer variar el diseño inicial y hay que estar preparado para ello.

Creación de modelos de primeras etapas. Inicialmente, antes de ponerse a implementar se tiene que contar con los recursos gráficos necesarios para poder desarrollar y hacer pruebas. Estos modelos iniciales son simples cajas cuadradas de colores. Esto es típico del desarrollo de videojuegos o de la animación por computador. Inicialmente se trabaja con elementos en bruto, y sin detallar, para finalmente dar el toque final para que quede de la forma deseada.

### Implementación

La línea temporal de desarrollo ha sido la siguiente:

Esqueleto: Creación del esqueleto del juego basado en estados: Se debe partir de un punto inicial. Para ello se implementa un esqueleto de juego básico con estados en el que no pasa nada. Solo hay movimiento entre estados. Para el cambio de estados se usa el teclado y por ello la librería OIS. Esta librería viene con Ogre, como ya se comentó con anterioridad.

Mundo inicial. Se crea un mundo inicial, compuesto por un cubo de dimensiones suficientemente grandes como para albergar sobre él otros objetos. Las dimensiones serán 15x6x15 como base de unidad la de un cubo estándar que poblara el escenario. Una vez creado se muestra haciendo uso del motor de renderizado.

Personaje: Tras la creación del mundo inicial se añade un cubo que representa al personaje. Este se posará encima del mundo y estará capacitado de movimiento a través de dicho escenario. Para el movimiento se tiene especial cuidado de usar un delta de tiempo para que en todos los sistemas el personaje de la misma sensación a la hora de moverse, y camine a la misma velocidad.

Cámara: Se incluye la funcionalidad de la cámara. Inicialmente la cámara gira sin topes. Puedes hacerla girar y se mueve de un lado a otro.

Se dota al personaje de salto y además se hace el movimiento en relación a la cámara que lo está enfocando, para que siempre podamos mover al personaje para la dirección respecto de nuestra visión.

Se añade la clase Actor, que será la base para construir todos los elementos que aparecen en pantalla tanto si son interactivos como si no lo son.

Se añade MakeCamera para el control de la cámara. Mediante esta clase se dotará al juego de una cámara que se mueve en 4 posiciones que corresponden con las 4 caras de la escena. Además se añade animación entre punto y punto para dar un efecto más llamativo al cambio de cámara.

El movimiento de la cámara lo hace en una circunferencia. Al pulsar el cambio, la cámara se mueve en pequeños ángulos hasta alcanzar un ángulo de 90 que corresponde con el siguiente punto clave.

La cámara siempre estará mirando al centro del escenario, para no perder detalle.

El desarrollo de estos primeros pasos se hace poco vistoso al usar modelos de colores. Así pues, para tener algo con mejor aspecto que mostrar en las reuniones de coordinación, se mejoran las texturas y se añade al personaje modelado. Además se añade un skyBox que será lo que se muestre de fondo en los mundos.

Se generaliza la inclusión de actores para que solo se necesite hacer una llamada para incluirlos en la escena. Una única función se encargara de situar a cada actor, dependiendo de su posición y tipo en la escena.

Se añade la orientación del personaje. Según para donde camine, ahora se orienta el nodo que contiene al personaje principal en la dirección en la que camina.

Se añaden las sombras a la escena. Mejora la sensación de juego y da un aspecto más amigable.

Se añaden las partículas de caminar de Nori. A la hora de andar y de saltar y caer, Nori genera unas partículas de polvo que resultan vistosas y claras.

Estado del personaje. En este momento se incluye el estado para saber si el personaje se encuentra cayendo, saltando, parado o caminando. Será necesario para distintos eventos.

se crea el método updateState en el actor para controlar el estado en que se encuentra cada actor.

Se crean los modelos finales de todos los elementos del juego.

Se añade la funcionalidad para manejo de mando de control de la consola de videojuegos Xbox.

Se crean texturas para los objetos poniendo especial atención en la textura transparente de los objetos invisibles.

Se añade la gestión de colisiones entre elementos. El personaje pulsa interruptores, detecta la meta, destroza bloques al subirse en ellos y mueve piedras. Además los interruptores también son pulsados con piedras y las piedras rompen a las frágiles si se ponen encima.

Se añade una lista de actores para tener claro que hay en la escena.

Se añaden flags para control del tipo de objeto a la hora de colisionar.

Se crean nuevas texturas para nuevos mundos y objetos.

Se añade un atributo a la clase actor, \_actived para determinar cuándo dicho actor espera una determinada acción.

Se añade a la clase Actor un contador para programar eventos, como por ejemplo la destrucción de una piedra frágil.

Se añaden las partículas para la eliminación de las piedras frágiles.

Se añade el método de generación de partículas para un actor. Es el propio actor el que genera sus propias partículas dependiendo de la acción que acontece.

Se hacen diversas modificaciones sobre el material de los bloques transparentes para conseguir una transparencia limpia y sin sombras.

Se añaden las partículas para la meta. Cuando el personaje consigue terminar la pantalla, se generan partículas en vertical como si se tratase de un teletransportador.

Se añade la funcionalidad de Restart. Cuando el jugador crea imposible conseguir la meta, podrá usar esta funcionalidad para resetear el nivel.

Desarrollo de consecución de meta. Cuando el jugador entra en contacto con la meta, se bloquea su control y se genera una animación que indica que se ha conseguido llegar al final del nivel. Además se efectúa un clean en bullet para que el personaje no tome comportamientos extraños al conseguir la meta.

Se añade el método que crea el mundo por el que vamos. Cada mundo contendrá unas características muy claras.

Se añaden las clases Stage y World para controlar y almacenar información de cada nivel y mundo.

Se crea un generador de niveles por medio de matrices 3D. ahora hacer un nivel es sencillo. Hay que rellenar una plantilla de texto.

Se añade la funcionalidad de guardado y carga de progreso del jugador.

Se crea un mapa para que el jugador vea su progreso al inicio y entre fases. Además se muestra al personaje sobre el mundo en el que se encuentra.

Se añade sonido a la escena. En principio se añaden los sonidos para los eventos de juego, como salto, caminar o colisiones con diferentes objetos.

Se añade el método Dead para detección de eliminación del actor y obrar en consecuencia dependiendo de qué tipo de actor sea.

Se crea una animación para la splash screen y se añade una imagen a la pantalla de press start. Estas pantallas corresponden a un mismo entrado, el estado IntroState.

Se pone mucho énfasis en que los Overlays que componen las pantallas aparezcan con su relación de aspecto original y no se deformen o se salgan de los márgenes de la pantalla.

Se añaden todos los mundos con características diferentes cada uno.

Se crean todos los archivos de texturas de los elementos del juego.

Se añade la música final para pantalla press start, para el mapa y la música para cada mundo.

Se modifican los elementos de pantalla como Pausa o información de la pantalla por la que va el jugador en ese momento.

### main.cpp

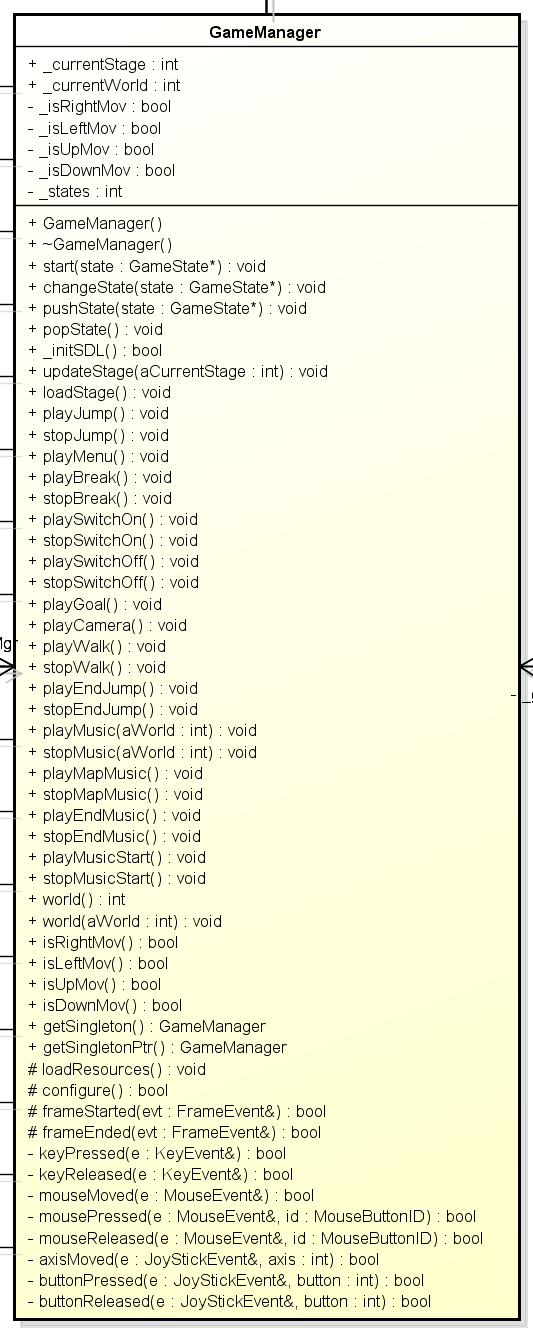
Clase principal del juego donde se crean por primera y única vez los objetos para las clases GameManager, IntroState, PlayState, PauseState, ReplayState y EndGameState al implementar todas ellas el patrón Singleton. De esta manera, cada vez que se desea conocer la única instancia de cada una de ellas, se hará a través de su método estático getSingletonPtr().

Se arranca el juego comenzando por el estado IntroState en la línea:

game->start(IntroState::getSingletonPtr());.

Siendo game un puntero a un objeto de tipo GameManager (GameManager\* game = new GameManager()) y capturando la excepción del tipo Ogre::Exception.

### GameManager.cpp

GameManager es una clase con herencia múltiple que hereda de public Ogre::FrameListener, public Ogre::Singleton<GameManager>, public OIS::KeyListener, public OIS::MouseListener, public OIS::JoyStickListener.

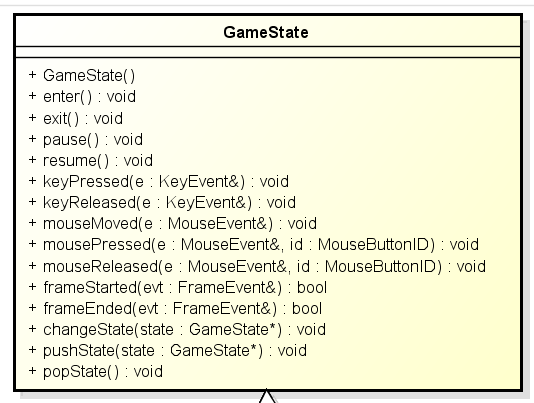
En su método void GameManager::start(GameState\* state) crea el objeto Ogre::Root(), crea el SceneManager, carga los recursos que necesitará, crea los objetos \_inputMgr y \_stagesMgr de tipos InputManager y StagesManager respectivamente, se añade como KeyListener, MouseListener y JoystickListener. Además le dice a root que él mismo es un listener, carga los sonidos y cambia al estado inicial que será IntroState.

Los métodos changeState, pushState y popState se utilizan para manejar los distintos estados por los que puede pasar el juego y de esta manera el flujo de ejecución.

Maneja todos los efectos de sonido y captura los eventos relacionados con el teclado (por ser KeyListener), con el ratón (por ser MouseListener) y con el joystick (por ser JoystickListener).

Cuando detecta un evento de tipo buttonPressed, convierte este evento de joystick para que lance un evento de tipo KeyListener y de esta manera no tener que hacer que hereden el resto de clases de JoystickListener.

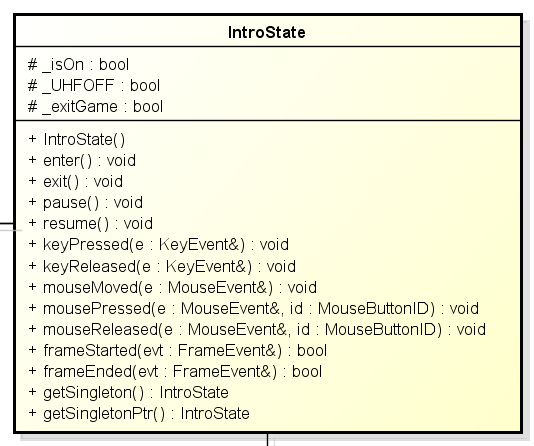
### GameState.h



Clase abstracta que representa un estado básico de la cual heredan IntroState, PlayState, PauseState, ReplayState y EndGameState.

Sus clases hijas deben implementar todos sus métodos, y en concreto, keyPressed y keyReleased para el manejo del teclado; mouseMoved, mousePressed, mouseReleased para el manejo del ratón; frameStarted y frameEnded como si de un FrameListener se tratara y enter,vexit, pause y resume como estados del juego en sí.

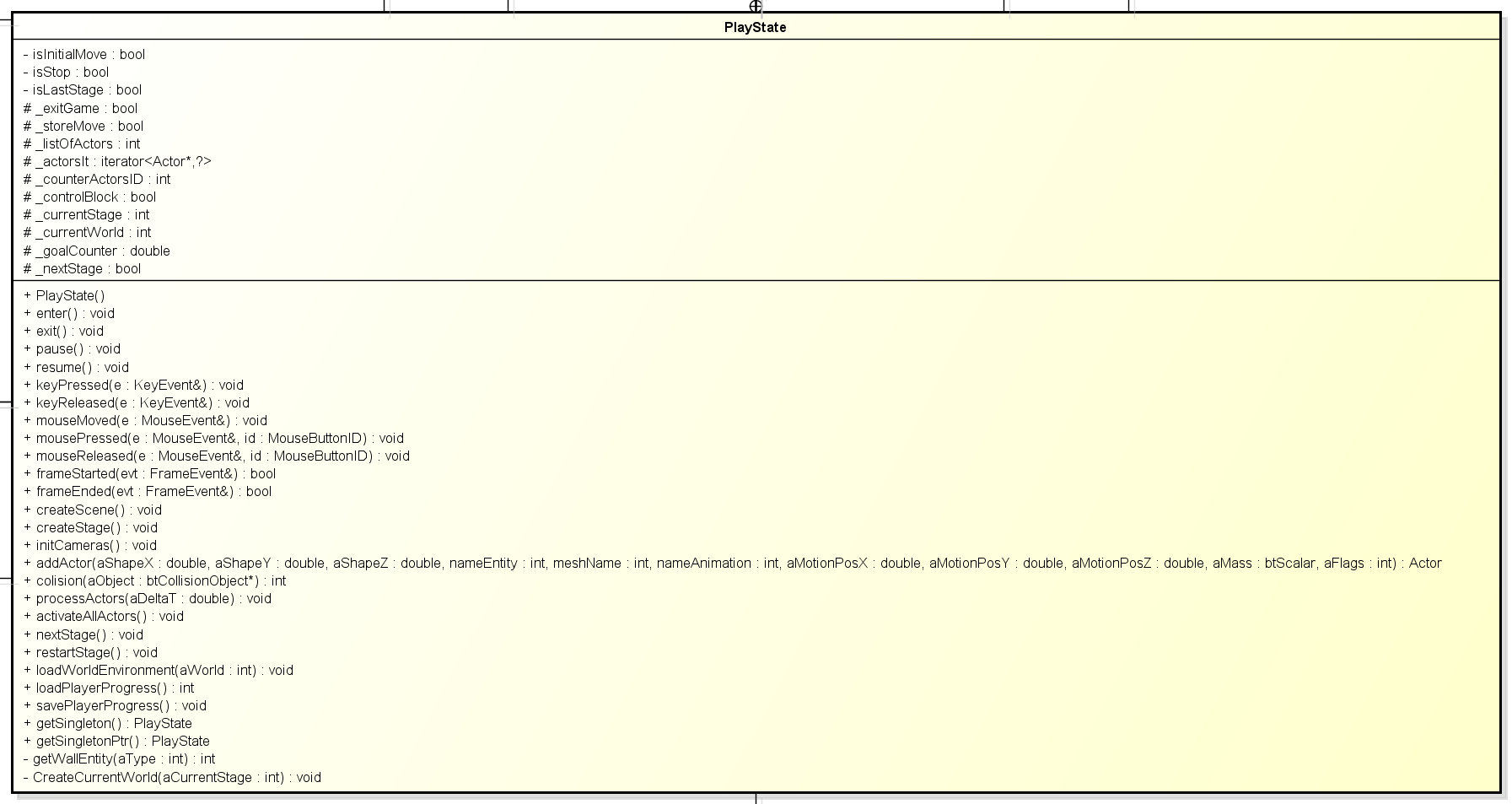
* + 1. **IntroState.cpp**



Primer estado por el que pasará el juego. Aquí se mostrará una pantalla de inicio donde se le pedirá al jugador que pulse START para continuar. Desde este estado, sólo puede irse al estado PlayState, si el usuario pulsa START en el mando o ESPACIO en el teclado, o salir del juego pulsando BACK en el mando o ESC en el teclado.

Implementa los métodos keyPressed, keyReleased, mouseMoved, mousePressed, mouseReleased, frameStarted y frameEnded por heredar de GameState y getSingleton y getSingletonPtr por heredar de Ogre::Singleton.

### PlayState.cpp



Es el estado sobre el que recae la mayoría del peso del juego ya que en él el usuario estará jugando. Desde estado se puede ir al estado ReplayState, pulsando R en el teclado o 4 en el mando, en el que se mostrará la pantalla de avance y desde esta, pulsando ESPACIO en teclado o 1 en el mando se volvería a PlayState a la pantalla por la que se iba jugando.

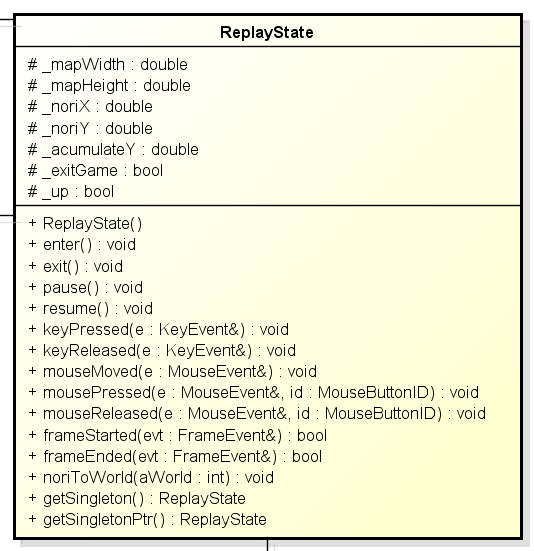
También se puede salir del juego directamente pulsando ESC en el teclado o BACK en el mando.

Se carga el mundo actual en el que se está jugando y se construye el escenario utilizando como dato el Stage actual, leído en StagesManager al comienzo del juego.

Añade los actores Actors y gestiona sus colisiones.

Implementa los métodos keyPressed, keyReleased, mouseMoved, mousePressed, mouseReleased, frameStarted y frameEnded por heredar de GameState y getSingleton y getSingletonPtr por heredar de Ogre::Singleton.

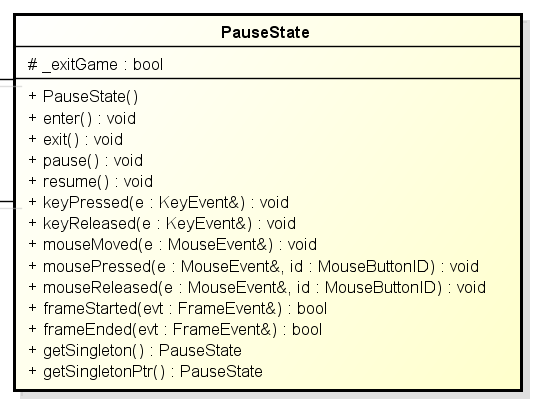
### ReplayState.cpp



A este estado se llegará cuando un usuario pulse la tecla R del teclado o el botón 4 del mando. Se cargará la pantalla de progreso, donde se mostrará a Nori pintado en el mundo por el que se va jugando y tras pulsar ESPACIO del teclado o START o 1 del mando se volverá al estado PlayState.

Implementa los métodos keyPressed, keyReleased, mouseMoved, mousePressed, mouseReleased, frameStarted y frameEnded por heredar de GameState y getSingleton y getSingletonPtr por heredar de Ogre::Singleton.

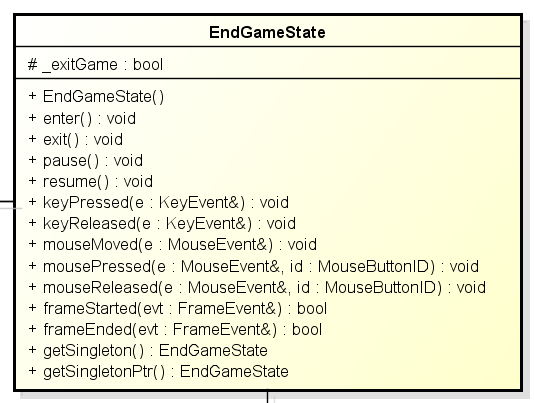
### PauseState.cpp



A este estado se llegará cuando el usuario pulse P en el teclado o 2 en el mando desde el estado PlayState. Pondrá el juego en pausa y pintará en la pantalla un overlay avisando del mismo.

Implementa los métodos keyPressed, keyReleased, mouseMoved, mousePressed, mouseReleased, frameStarted y frameEnded por heredar de GameState y getSingleton y getSingletonPtr por heredar de Ogre::Singleton.

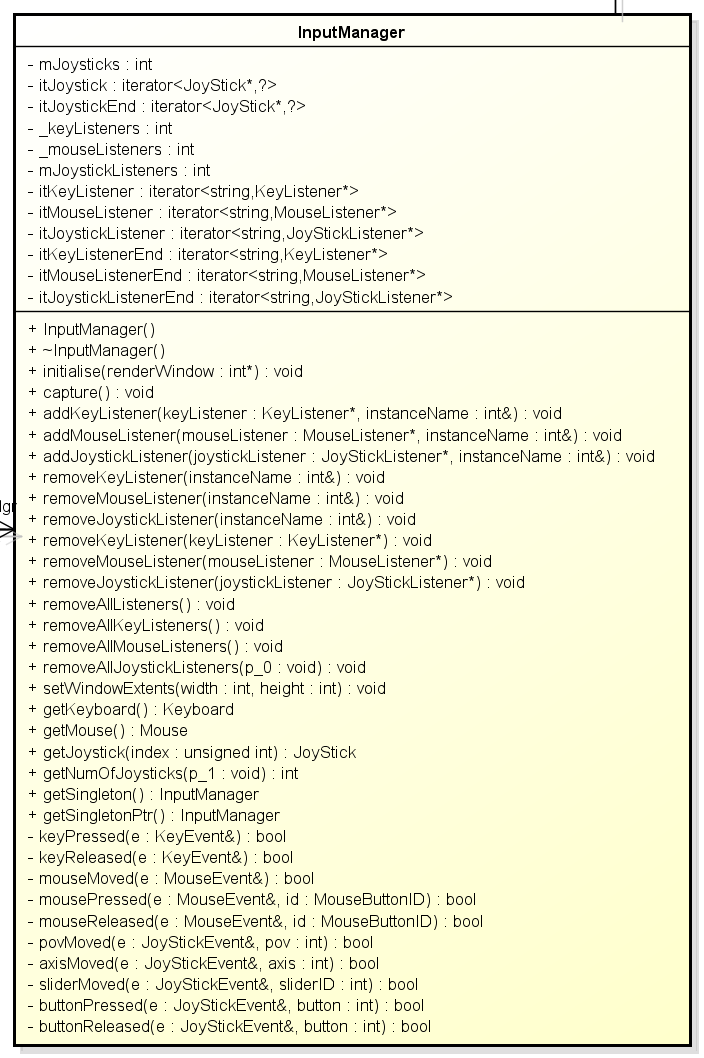
### EndGameState.cpp



A este estado se llega cuando se finaliza la última pantalla con éxito. Muestra una pantalla dando la enhorabuena con música sonando. Desde este estado sólo se puede salir del juego con ESC desde el teclado o con BACK desde el mando.

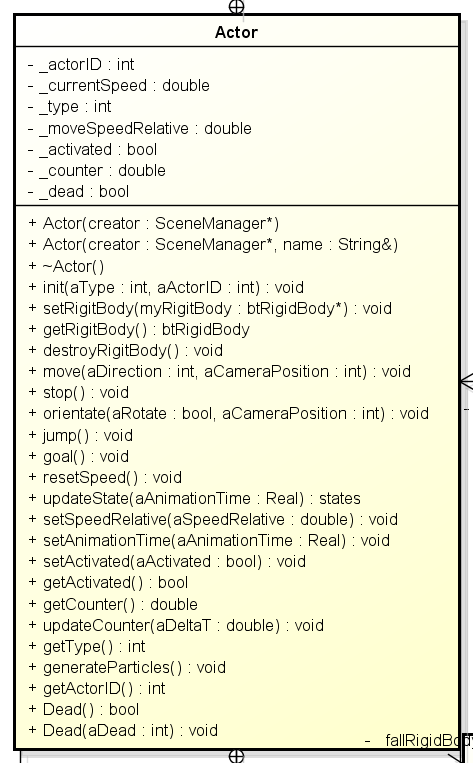
Implementa los métodos keyPressed, keyReleased, mouseMoved, mousePressed, mouseReleased, frameStarted y frameEnded por heredar de GameState y getSingleton y getSingletonPtr por heredar de Ogre::Singleton.

### InputManager.cpp



Clase gestora de entradas. Hereda de forma múltiple de Singleton<InputManager>, OIS::KeyListener, OIS::MouseListener y OIS::JoyStickListener, por lo que implementa los métodos necesarios para el manejo completo del teclado, el ratón y el joystick.

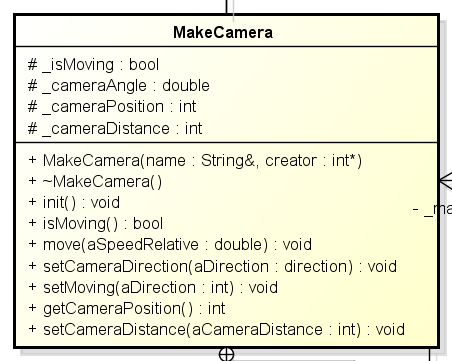
### Actor.cpp



Clase que hereda de Ogre::SceneNode. Todos los elementos de cada mundo serán de tipo Actor y tendrá un tipo asignado en función del cual creará las partículas asociadas a cada uno.

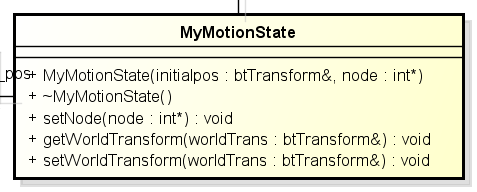
Manejará las colisiones, la física, etc. de cada tipo de actor.

### MakeCamera.cpp



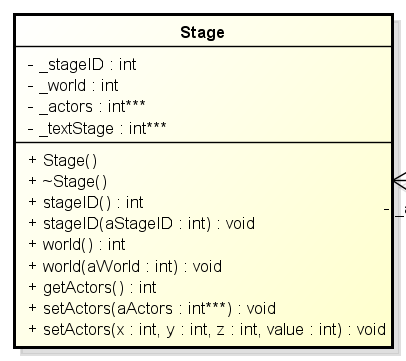
Se utilizará para controlar el movimiento de la cámara. Dicho movimiento se efectúa de 90º en 90º y se puede realizar girando a la derecha o a la izquierda.

### MyMotionState.cpp



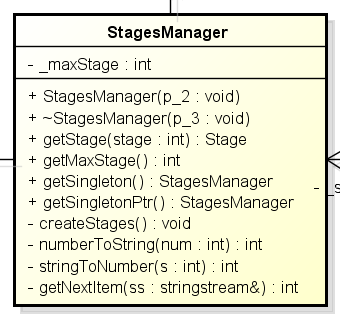
Clase que se utilizará para crear los cuerpos rígidos de Bullet.

### Stage.cpp



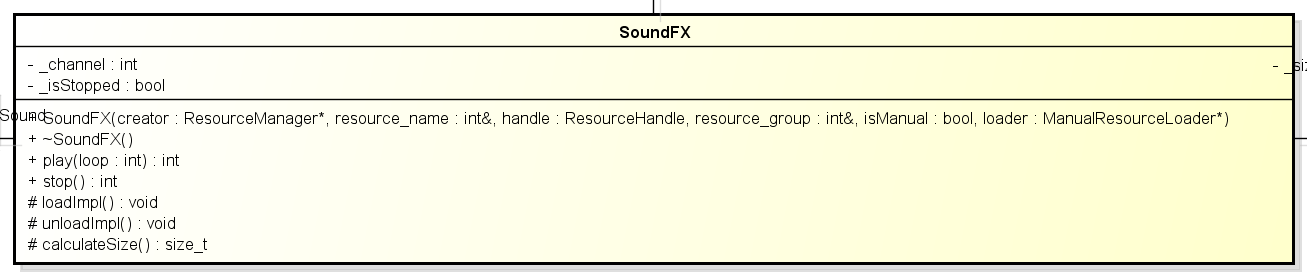
Mantendrá un array de tres dimensiones con la información leída de los ficheros .txt para generar cada escenario. Además guardará el número de Stage a la que corresponde esa información y el mundo al que pertenece la misma.

### StagesManager.cpp



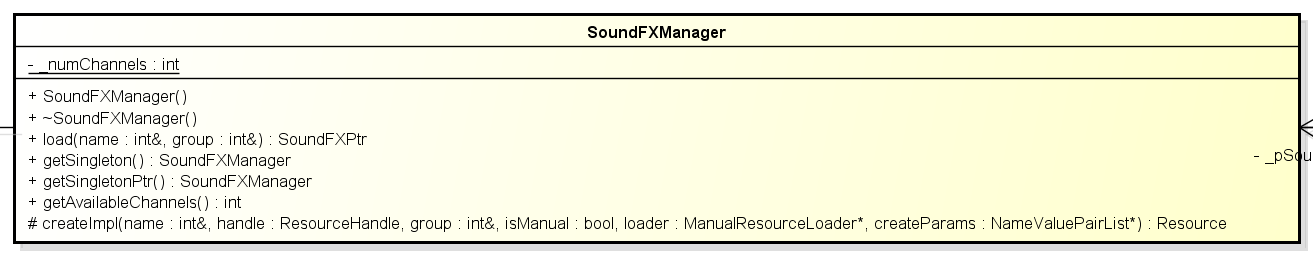
Se encargará de leer los .txt que encuentre en el subdirectorio /stages que se encontrará en la misma ruta del ejecutable. Almacenará el máximo número de Stages que puede cargar y creará objetos de Stages correspondientes una vez interpretados los ficheros de creación de escenarios y los almacenará en un array, de manera que se leerán e interpretarán todos los escenarios al arrancar el juego.

### SoundFX.cpp



Será el encargado de cargar todos los recursos de sonido, reproducirlos y pararlos.

### SoundFXManager.cpp



Su función es hacer de interfaz entre SoundFX y el resto de clases.

# Análisis de costes

En la siguiente tabla se muestra el coste en tiempo del desarrollo del presente proyecto. En ella puede verse desglosado el coste por persona y tarea.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | TAREA | Pedro | Pamela | H/Tarea |
| 1 | Brainstorming y división de tareas | 4 | 4 | 8 |
| 2 | Diseño de arquitectura y solución software | 16 | 16 | 32 |
| 3 | Reuniones | 10 | 10 | 20 |
| 4 | Desarrollo de esqueleto del juego. Estado de juego y transiciones. | 16 | 10 | 26 |
| 5 | Creación de modelos de mundo y tiles | 0 | 16 | 16 |
| 6 | Carga de mundo de test | 5 | 3 | 8 |
| 7 | Funcionalidad de MakeCamera | 25 | 10 | 35 |
| 8 | Creación de modelo de personaje principal | 3 | 16 | 19 |
| 9 | Desarrollo comportamiento de personaje principal | 20 | 10 | 30 |
| 10 | Implementación de control de juego con xbox controller | 5 | 20 | 25 |
| 11 | Implementación de comportamiento de objetos interactivos. | 30 | 25 | 55 |
| 12 | Implementación de sonidos | 16 | 5 | 21 |
| 13 | Generador de escenarios | 15 | 30 | 45 |
| 14 | Carga de escenarios con archivos externos | 15 | 40 | 55 |
| 15 | Carga y guardad de avance | 5 | 3 | 8 |
| 16 | Diseño y desarrollo de pantallas de dialogo | 30 | 15 | 45 |
| 17 | Documentación | 56 | 56 | 112 |
| 18 | Pruebas y validación | 60 | 60 | 120 |
| 19 | Desarrollo de niveles | 40 | 40 | 80 |
| 20 | Sistema de partículas para el juego | 35 | 10 | 45 |
| 21 | Programación de colisiones | 40 | 15 | 55 |
| 22 | Desarrollo de diseños 2d | 30 | 20 | 50 |
| 23 | Creación de modelos de objetos interactivos | 10 | 50 | 60 |
|  | Totales | 486 | 484 | 970 |

# Manual de usuario

MakeMake es un juego sencillo en su manejo, aunque no tanto en la consecución de los puzles. De hecho es la principal premisa. Se ha puesto especial interés en hacer lo más sencillo posible el acceso al juego.

**HISTORIA**

Nori, habitante del pequeño planeta MakeMake, es el único ser de su especie que queda, como consecuencia de una invasión imprevista. Todos los Makelianos, excepto él, fueron capturados en naves Bistecca-Grasso a las órdenes del General Obvious Salame.

Las pretensiones de Salame son claras: vender a granel a todos los habitantes de MakeMake y usar sus tierras para producir el famoso combustible barato Aspro. A bordo de su nave Cetriolo, Salame tuvo la misión casi completa cuando detectó un punto de vida en el planeta. Nori.

Con este panorama, solo hay una posibilidad. Reunir las 4 piezas que completan Lattuga, un objeto milenario que neutraliza las naves modelo Grasso además de bloquear el acceso de habitantes externos al planeta, haciéndolo habitable únicamente por Makelianos.

Nori debe completar 4 mundos, al final de cada cual obtendrá una de las 4 piezas de Lattuga. Una vez completados los mundos, la invasión será neutralizada y todos los habitantes de MakeMake podrán ser rescatados de la nave prisión Carcere Grissini modelo Bistecca-Barattolo.

**El juego**

*Escenario.*

Un escenario consta de una base cubica donde ocurre toda la acción. se encuentra flotando en un mundo con un entorno claramente definido y reconocible. Además el escenario se compone de bloques de aspecto similar al bloque base, con los que se construye toda la escena. Nuestro personaje vive en esa plataforma y no debe salir de ella si no es por el tele transportador que en ella existe. Solo hay uno y puede estar a la vista o no. ¡Explora!

*Nori.*

El personaje principal es Nori. Nori es un bloque de sushi, de la especie MakeMake que habita el planeta homónimo. Un exceso de alga ha provocado a la especie tener patas que facilitan su desplazamiento. Además, las patas le posibilitan saltar para llegar a zonas más altas que él.

*Objetos interactivos*

Para conseguir completar los puzles, nori puede hacer uso de varios objetos que aparecen en la escena. En algunos escenarios es prácticamente obligado usar todos los que salen, pero... ¡cuidado! Puede que ese bloque que mueves, no sea necesario y este ahí solo para despistar.

Los diferentes bloques son:

*Piedra.*

es un elemento muy útil para alcanzar determinados objetivos. Únicamente puedes moverla, pero interactúa con prácticamente todos los elementos. Si la arrastras sobre el suelo, puedes transportarla, pero si la sitúas encima de una piedra frágil, esta se romperá. Puede que antes de hacerlo tengas unos segundos para usarla con el fin de llegar algo más alto. Además, la piedra es útil a la hora de usar los interruptores. sitúala encima y activaras completamente el interruptor.

*Piedra frágil.*

Este elemento permite permanecer sobre el un tiempo, transcurrido el cual se derrumba y desaparece para siempre. Úsalo para llegar a sitios altos y recuerda, que pueden ocultar cosas. Quizá algo que no veas, este tras un muro derrumbable. Ten en cuenta también que algún bloque que derrumbas sin porqué puede ser que lo necesites más adelante para conseguir completar el presente nivel.

*Interruptor y bloques transparentes.*

En el mondo de MakeMake existen unos bloques muy especiales. Son transparentes y no son interactuables. No puedes pisar sobre ellos ni bloquean nada. Pero mediante el uso de un interruptor la cosa cambia. Con el interruptor activado, los bloques transparentes se materializan y se transforman en sólidos. En ese momento, el personaje, o cualquier cosa que haya en el escenario puede hacer uso de los bloques para caminar sobre ellos o bloquear el paso de cierto elemento.

*Meta*

Es el destino final. La razón de Nori. Debe llegar ahí para seguir su aventura. Una vez en la plataforma, Nori emprende un viaje al siguiente escenario y se muestra ante el jugador un nuevo reto.

*Cámara*

La cámara del juego cobra un protagonismo obligado. La escena se presenta en un punto de la cámara que podrás mover. El hecho de poder moverla es necesario, ya que para entender la profundidad de ciertos objetos es vital. Si bien es cierto que las sombras ayudan al jugador a calcular distancias entre objetos, habrá mundos en los que la posición de la luz haga necesario cambiar la cámara para tener claro donde esta cada elemento. Además puede que a simple vista no quede claro donde se encuentra la plataforma de meta y haya que explorar visualmente un poco hasta encontrarla.

La cámara tiene 4 puntos que corresponden a las cuatro caras laterales del mundo cubico. ¡Úsala y “verás”!

*Control*

El control del personaje es muy sencillo. Corresponde con el uso normal de un personaje en un juego de plataformas clásico. Puedes moverte y saltar. Para ello puedes hacer uso de dos tipos de controles:

- Teclado:

* Teclas de dirección: mueven el personaje
* Tecla espacio: salto
* Teclas q y w: movimiento de la cámara de izquierda y derecha respectivamente.
* Tecla r restar
* Tecla p pausa

- Mando de control

Se ha implementado el control mediante mando de la consola xBox o compatible. Los controles son los siguientes:

* Analógico. control del personaje.
* Botón 1 salto
* Botón start comenzar
* Botón 2 pausa
* Botón 4 restart
* Botón back salir
* Botones L y R giro de la cámara

Así mismo para el paso de pantallas en los menús bastara con pulsar el botón 1.

*Juego completado*

Una vez el jugador haya completado todos los puzles propuestos, aparecerá una imagen que felicita del logro y lo anima a rejugarlo, volviéndose al primer nivel y empezando nuevamente la aventura.

* Nota de instalación:
  + Los ejecutables necesitan vcredist\_x86. Se incluye en la instalación de todos, pero puede ocurrir que de un error de instalación diciendo que no encuentra el recurso. A pesar de que lanza el error, se consigue instalar correctamente.
  + Si al arrancar los juegos no aparece nada, ni siquiera un mensaje de error, acuda al fichero plugins.cfg que encontrará en el directorio del juego una vez instalado, y donde aparecen esta líneas Plugin=RenderSystem\_Direct3D9 Plugin=RenderSystem\_Direct3D11, sustituyalas por # Plugin=RenderSystem\_Direct3D9 # Plugin=RenderSystem\_Direct3D11 o instale las directx en su pc
  + No se recomienda la instalación en el directorio “Program Files” debido a los permisos necesarios para escribir en esta ruta.

# Conclusiones y trabajo futuro

## Conclusiones

El desarrollo del presente proyecto ha sido una experiencia muy enriquecedora. Tanto es así que nos hemos quedado con ganas de más, a pesar de la cantidad de horas empleadas y el esfuerzo personal. Aquí no acaba MakeMake.

Durante el desarrollo muchos han sido los problemas que hemos ido encontrando y siempre, aun costando, hemos dado con una solución satisfactoria. Es cierto que al final, con la presión de los plazos, el código no es todo lo bonito que deseamos, y que considero podemos hacer, pero la velocidad con que hemos tenido que desarrollarlo ha hecho que ciertas cosas se quedasen de un modo "solución rápida". Sabemos que esto no es lo ideal, pero hemos buscado que el juego este completo al nivel que inicialmente acordamos, y queda por realizar un trabajo de rediseño general. Con él, el código quedara más legible y usable por parte de cualquiera que entre a formar parte del equipo.

Hemos conocido durante este año varias herramientas, que confluyen todas en el motor de renderizado Ogre3D. Es cierto que al principio es un golpe duro y sin avisar. No es fácil hacerse con el manejo de este motor. Incluso echarlo a andar es todo un reto. Pero una vez que te has familiarizado con el, es muy generoso. Se puede decir que empezamos sufriéndolo y hemos terminado disfrutándolo.

## Trabajo Futuro.

Makemake nos gusta. Estamos contentos con el resultado final. Esto no quiere decir que no queden cosas por hacer. Si bien es cierto que el juego se acerca mucho a lo que en un principio imaginamos, hemos quedado con ganas de más, de mejorar e incluir muchas funcionalidades que harán al juego más entretenido y juzgable, y puede que con ellas, sea el jugador sea el que quiera más.

A continuación describimos varias de las características que más importancia tienen a la hora de abordar nuevos trabajos en este proyecto:

*Editor de niveles*

Una de las cosas que echaremos de menos al jugar es que podamos crear nuestros propios niveles para compartirlos con algún amigo. En el actual desarrollo se pueden crear nuevos niveles, pero no es una forma amigable. Actualmente para crear nuevos niveles se puede crear o modificar un archivo de texto plano con el formato conocido y añadir el número de matrices que componen un escenario (como se dijo anteriormente un escenario esta compuesto por una matriz tridimensional de 15x6x15 posiciones).

Lo ideal es crear un editor de niveles que permita varias cosas:

- colocación de elementos en el espacio tridimensional.

- colocación de elementos de atrezo del escenario. Bloque sin funcionalidad.

- colocación de personaje principal y objeto meta.

- carga de niveles disponibles.

- guardado de modificaciones.

- prueba del nivel durante la edición.

*Enemigos. IA y modelos.*

En el concepto inicial del juego aparecían personajes que te hacían más difícil conseguir pasar de nivel. Estos enemigos deambulan por el escenario y si consiguen tocarte, reinicias el nivel. Además, pueden estar provistos de armas, lo cual puede hacer aún más complicado superar el reto.

Los enemigos deben estar provistos de IA, para que sea más laborioso escapar de ellos.

*Mayor número de niveles.*

Un punto del juego que nos ha quedado corto es el número de niveles a conseguir. El desarrollo de estos ha sido manual. Debido al elevado coste de tiempo que ha supuesto la creación del juego y los plazos, el número de niveles, y la dificultad de los mismos se ha quedado corta. No obstante, teniendo ya los elementos para construcción de niveles, se pueden ir añadiendo sin problema. Hay que tener en cuenta que la máxima en la creación de este juego ha sido tener completo la mecánica del juego separando la edición de niveles. De este modo se puede subcontratar la creación de ellos o incluso que los jugadores puedan subirlos a un servidor del que puedan actualizarse el resto de usuarios.

*Detección de caminos.*

Cuando se crea un escenario, o cuando estás jugando en él, sería interesante que el juego te indicase si es posible llegar a la meta, con la disposición actual de elementos en el escenario y dependiendo de la posición de nori y la meta. De este modo, el escenario se reiniciaría y empezarías en la posición inicial. Esta característica, si bien supone un reto interesante en lo que programación se refiere, Pensamos que es algo menos "romántico". Quizá es más interesante que el propio jugador, tras deambular por el escenario, termine dándose cuenta de que no tiene solución. Esto hace que pueda estudiar mejor cómo se consigue descifrar el puzle al que se enfrenta.

*Nuevos objetos*

En este aspecto, el juego admite mucho margen de ampliación. Para esta ocasión y teniendo en cuenta el plazo, se han implementado un número limitado de elementos interactivos. Ahora bien, en el concepto inicial del juego se tuvieron en cuenta un número mayor de dichos elementos del juego. A la piedra móvil, la piedra frágil, el interruptor con activación de objetos se le unirán los siguientes elementos:

- Saltador. Objeto que tras caer en el permite ejecutar un salto el doble de altura que con el salto normal de Nori. De este modo el personaje puede llegar a sitios a los que sería imposible llegar.

- Puertas de única dirección. Este elemento puede ser útil a la hora de proponer retos al jugador. Así pues se puede añadir en un lugar del escenario del que no se pueda volver y hacer que el jugador haya tenido que "hacer cosas" en un lado, antes de pasar al otro, como por ejemplo lanzar una piedra o romper algún elemento.

*Powers up para el personaje.*

Otro de los elementos que no se han podido añadir ha sido los poderes. Mediante estos items, el personaje puede hacer uso de habilidades que normalmente no tiene, como por ejemplo un salto mayor, o la posibilidad de traspasar algún muro. Además, la idea es que estos elementos vayan a parar a una mochila que lleva el jugador y pueda guardarlos para futuros escenarios, si no los necesita en el actual. así pues, puede que en un escenario futuro necesites haber hecho ciertas cosas en otro anterior. Incluso quizá debas rejugar niveles.

*Selección de niveles.*

En el desarrollo actual, el destino esta guiado y escrito. Es decir, la ruta es lineal y va desde el nivel 1 hasta el último. Este modo está bien, pero lo ideal es poder rejugar los niveles que consideremos más entretenidos, e incluso cargarlo para que alguien a quien se lo recomiendes pueda jugarlo. Para un futuro se espera tener una pantalla con un mapa más grande por el que se pueda navegar y entrar en cada mundo. Una vez dentro, la idea se cargar todas las pantallas que pertenecen a ese mundo.

*Lista de achievements.*

Se pretende añadir un conjunto de logros al juego. Esto completa la experiencia de juego y da más vida al mismo. Los logros pueden ir desde completar el conocimiento de manejo del jugador a completar determinado puzle en algún escenario o la recolección de determinados elementos. Además, estos logros pueden ser conectados con un servidor para consultar que logos ha conseguido cada jugador y poder cargar los tuyos en cualquier lugar.

*Varias cuentas de juego.*

En esta entrega existe un único jugador. Se comienza el reto desde el principio y se va avanzando, pero no existe la posibilidad de arrancar un juego desde el principio. Para mejorar queda dotar al juego de varias cuentas de juego, y poder navegar entre ellas. Además, estas cuentas pueden hacer uso de servicios en la nube de los cuales obtener determinada información y además guardar avance y logros.

*Nuevos mundos.*

MakeMake en su primera edición tiene 4 mundos que ya se han comentado. Como trabajo futuro queda completar un número importante de ellos. Además, en estos mundos queda por añadir diferentes modificaciones como:

- nuevos modelos 3d para mundos más complejos.

- efectos sobre las condiciones de juego. Cosas como un mundo helado andando sobre hielo escurridizo, viento que actué sobre el personaje y elementos del mundo, movimientos de luz, terrenos poco estables, estanques, efectos climáticos, inclinaciones y demás efectos podrían completar el desarrollo.

*Efectos de transición entre pantallas.*

En el presente desarrollo se ha puesto especial detalle en hacer una experiencia amigable y vistosa. El menú es muy sencillo, ya que solo te lleva a jugar y hay poco donde fallar. En este apartado, la mejora pasa por hacer un efecto fade-out/fade-in en la transición entre diferentes pantallas de menú y juego y componer los elementos en pantalla con animaciones. Si bien es un cambio menor, puede que de mayor empaque al conjunto.

# Bibliografía

* Ogre3D: http://www.ogre3d.org/
* Ogre3D Foro: http://www.ogre3d.org/forums
* Blender: http://www.blender.org/
* Wikipedia: http://www.wikipedia.org/
* Stackoverflow: http://stackoverflow.com/
* Tutorial Ogre + Bullet: http://oramind.com/ogre-bullet-a-beginners-basic-guide/

ANEXO 1: Diagrama de Clases

