Proyecto fin de carrera. Facultad de informática.
Universidad de Las Palmas de Gran Canaria

Desarrollo de un concepto de juego por computador para la programación de técnicas de Inteligencia Artificial

ENRIQUE GONZÁLEZ RODRÍGUEZ
Las Palmas de Gran Canaria, 10 de Abril del 2008

Proyecto fin de carrera de la Facultad de Informática de la Universidad de Las Palmas de Gran Canaria presentado por el alumno:

ENRIQUE GONZÁLEZ RODRÍGUEZ

Título del Proyecto: Desarrollo de un concepto de juego por computador para la programación de técnicas de Inteligencia Artificial

Tutor: Modesto Castrillón Santana

ÍNDICE

1. Introducción

A lo largo de la historia, los videojuegos han supuesto una de las posibles salidas para los Ingenieros Informáticos. Sin embargo, antaño este sector no estaba del todo desarrollado condicionado principalmente por un hardware poco evolucionado, el cual no permitía pasar más allá de los gráficos 2D. No sólo el hardware condicionaba el producto final, sino la deficiencia existente en los procesos ingenieriles empleados.

A parte de las carencias tecnológicas contempladas en los videojuegos de la época, se observaban comportamientos anómalos de los personajes que formaban parte del videojuego. El motivo fundamental de estos comportamientos era la falta de una programación orientada a la Inteligencia Artificial, ya que por esa época, el desarrollo de la I.A. estaba en sus inicios.

En las últimas décadas se ha ido solventando este hecho, dando lugar, junto con técnicas avanzadas de programación e ingeniería del software, la creación de juegos en 3D. A fecha actual, el desarrollo de la Inteligencia Artificial está en crecimiento en diversas ramas de la informática, siendo una de ella, los videojuegos, esto es posible gracias a los avances del hardware que permiten a los desarrolladores aplicar I.A. más sofisticadas.

El objetivo principal de este proyecto es desarrollar un videojuego donde se combinen técnicas de desarrollo software, gráficos por ordenador y que sirva de plataforma para el análisis y desarrollo de comportamientos basados en inteligencia artificial. La idea surge como una propuesta del alumno con el fin de aplicar ramas específicas dentro de la Informática, las cuales tienen un gran interés en el género al que pertenece el videojuego (acción en primera persona).

El desarrollo de disciplinas en la Informática, principalmente los Gráficos por Computadores y la Inteligencia Artificial, han permitido evolucionar a la industria del videojuego para pasar de unos juegos gráficamente simples con un comportamiento muy elemental, a juegos con unos niveles de detalles gráficos muy elaborados (iluminación, sombreado, realismo…) y que incluyen además reglas de comportamiento sofisticadas basadas en la Inteligencia Artificial.

El mundo de los videojuegos está en auge y es evidente que se requiere de ingenieros informáticos capaces de aportar de sus conocimientos para mejorar la productividad. De este contexto nace la idea de la creación de un videojuego, como propuesta del alumno.

La motivación personal reside en la construcción de un juego a nivel de programación haciendo uso de la ingeniería del software. Por eso partiremos de un juego existente denominado, Quake, del cual obtendremos las partes que no son labor de un ingeniero informático como sería la construcción de escenarios, entidades, generación de música y efectos u otras tareas. Sin embargo, nosotros llevaremos a cabo de nuevo la generación del código del Quake, con la salvedad de que seguiremos las pautas de la ingeniería del software para integrar todos los módulos que componen el juego; Inteligencia Artificial, Audio, Gráficos, Sistema de Colisiones. A este nuevo juego lo denominaremos QuakeGarage.

De manera resumida, en el presente documento encontraremos los objetivos que se proponen para este proyecto. Como aspecto previo a lo que se estipula en los objetivos se ofrece un documento de diseño del juego (apéndice), donde se describe todos los elementos que participan en el videojuego.

La primera fase se basa en los modelos de análisis y diseño visto en la ingeniería del software, se trata del trabajo que se debe realizar antes de la programación. En el modelo de análisis podremos encontrar las necesidades de recursos externos y del usuario, una lista enumerada de todos los requisitos para llevar a cabo el juego, estos requisitos se agruparán en componentes y por último ofreceremos un conjunto de casos de usos. En el modelo de diseño, podremos observar la construcción de nuestro software desde la perspectiva del diseño arquitectónico y el diseño en componentes.

La segunda fase se basa en la programación, se describirán los programas y tecnologías utilizadas con una comparativa de los motores gráficos y juegos contemplados. Los tres grandes bloques de la programación son la Inteligencia Artificial, los gráficos por computador y la implementación del juego.

Por último, aportaremos conclusiones y posibles extensiones como trabajo futuro.

2. Estado actual del tema

La industria de los juegos por ordenador se ha mostrado muy exitosa en años recientes, siendo evidente su interés comercial. Este interés se ve reflejado en las disputas comerciales que distintas grandes compañías tienen en la actualidad por ganar la cuota del mercado.

El desarrollo de un producto de este tipo hace uso de la colaboración multidisciplinar, si bien, es de destacar la importancia de distintas materias contempladas en la titulación de un Ingeniero en Informática, tales como: Ingeniería del Software, Arquitectura de Computadores, Procesadores de Lenguajes, Gráficos por Computador, Inteligencia Artificial, Multimedia, etc. De esta forma, podemos combinar conocimientos adquiridos en distintos contextos para la consecución de un producto.

En algunas ramas específicas como la Inteligencia Artificial y su aplicación a la Robótica Móvil, se están haciendo uso de entornos de juegos como simuladores para analizar comportamientos antes de realizarlo con costosos robots físicos. Este hecho influye notablemente para que se desarrollen videojuegos que incluyan una interfaz de programación de la aplicación (API). Es decir, dispondremos de una herramienta docente para la simulación de estrategias de comportamientos inteligentes.

3. Objetivos

Como ya sabemos, nuestro objetivo principal es crear un juego por computador donde se puedan aplicar técnicas de inteligencia artificial y de gráficos por computador. Podemos descomponer el objetivo del proyecto en los siguientes apartados:

a. Diseño de un concepto de juego FPS (acción 3D en primera persona).
   Se construirá una arquitectura del juego haciendo uso de los patrones de diseño de la ingeniería del software.

b. Desarrollo de una plataforma para el desarrollo de estrategias de inteligencia artificial para diseñar Bots.
   Se resolverá las necesidades de un jugador artificial o Bot para simular su comportamiento dentro de un entorno

c. Visualización de un entorno 3D aplicando algunas técnicas de rendering.
   Se modificará un determinado motor gráfico, a la que aplicaremos técnicas de gráficos por computador, con el objetivo de visualizar un escenario en 3D.

El desarrollo de este proyecto se basará en el uso de la ingeniería del software, y se le dará una gran importancia al uso de algunas herramientas de código abierto como puede ser el motor gráfico, el motor de audio o la base de datos. La adquisición de este tipo de herramientas, nos dará mayor flexibilidad para modificarlo e incorporarlo a nuestro proyecto.

Además, se realizará un estudio comparativo donde se pretende evaluar el uso de nuestra herramienta principal, el motor gráfico. De esta manera, se analizará la cuota del mercado referente a este tipo de aplicaciones.

Otro aspecto esencial es la inteligencia artificial. Dado que se construirá un juego, lo aprovecharemos como una interfaz para la programación de técnicas de inteligencia artificial. Es decir, el juego, QuakeGarage, que construiremos a lo largo del proyecto, nos servirá de API (interfaz para la programación de aplicaciones) como medio de comunicación para que nuestro agente autónomo interaccione con el entorno. Asimismo, se dará como ejemplo de aplicabilidad, la implementación de un BOT o agente artificial. Esto nos permitirá ver el potencial de desarrollo de las herramientas que la API proporciona.

4. Modelo de Análisis

Descripción

La especificación de requisitos Software (SRS) proporciona una lista con la descripción de especificaciones funcionales y no funcionales para los componentes software de QuakeGarage. Es derivado, en parte, del documento de diseño de QuakeGarage (GDD, se incluye en el anexo de este documento) y es suplementado por la especificación de diseño Software (SDS) y el plan de test. Los requisitos de Software proporcionan las siguientes vistas:

- Una lista de los requisitos presentado numéricamente.
- Una agrupación preliminar de componentes.
- Exploración de los casos de usos.

El documento tiene la intención de establecer la meta inicial para el desarrollo del software.

Perspectiva

Este juego está dentro de la categoría FPS (first-person shooter), con un modo de un único jugador para los usuarios del ordenador. Un FPS es un videojuego que renderiza el mundo desde la perspectiva visual del carácter, y prueba la habilidad del jugador; en la puntería con las armas y su capacidad de movimiento ya sea para atacar o defenderse.

En esta sección, el lector ganará una sensación de derivación, intención y alcance del producto. Se tiene la intención de ilustrar los procesos y metodologías empleadas en la ingeniería del software y el desarrollo de videojuegos. QuakeGarage incorpora características que son comunes en los juegos de ordenador, pero difiere en la metodología empleada para su desarrollo. Se trata de una descomposición del software en componentes, que nos darán una visión clara de las partes que conforman el producto, permitiendo una implementación guiada de las características.

Descomposición de las funciones del sistema

Las componentes funcionales preliminares del producto son las siguientes:

- Componente 1: Apertura del juego.
- Componente 2: Gráficos.
  - Componente 2.1: Objetos GUI en 2D.
  - Componente 2.2: Constructor de niveles.
  - Componente 2.3: Constructor de entidades.
  - Componente 2.4: Efectos y animaciones.
- Componente 3: Sonido.
- Componente 4: Configuración del jugador.
- Componente 5: Opciones del juego.
- Componente 6: Navegación en el nivel.
- Componente 7: Unidades físicas.
- Componente 8: Ítems.
- Componente 9: Combates.
- Componente 10: Inteligencia Artificial.
- Componente 11: Vista de las estadísticas.
- Requisitos no funcionales.

Los componentes mencionados son explorados más profundamente en el apartado “Vista en componentes de los requisitos funcionales”.

Características del usuario

El juego está diseñado para personas mayores de 12 años. Tiene un único nivel de dificultad y está pensado para realizar partidas cortas, sin necesidad de cargar y guardar estados del juego. Por otro lado, sus características ofrecen oportunidades interesantes, y está dirigida a gente interesada en combates expertos contra Bots, especializados en diversos mapas.

Restricciones

Existen claras limitaciones que se aplican al juego. Se trata de un videojuego de un solo jugador que no hace uso de Internet, ni dispone de una modalidad multijugador (aunque se podría extender en el futuro).

En ningún caso se puede esperar un juego con el nivel de complejidad que se ofrece actualmente en el mercado, ya que tan sólo se trata de una versión con fines académicos. Tampoco se debe esperar una ejecución satisfactoria en todos los equipos. Se ejecutará en los ordenadores que mejor respondan a determinados aspectos de rendimiento.

Asumciones y dependencias

El desarrollo del sistema depende de la utilización de funciones de DirectX y Win32 que maneja el motor gráfico 3D Irrlicht y también de las librerías de sonido IrrKlang y la base de datos MySQL 5.0.

Un paquete de instalación acompañará al software, que integrará las diferentes partes que conforman el juego y se podrán arrancar a través de un ejecutable.

Necesidades de recursos externos

Los asuntos tratados en esta sección del documento se refieren a los requisitos de usuario y del sistema que afectan el funcionamiento del juego dentro de un determinado sistema.

Interfaz de usuario

El juego corre en un ordenador personal en el cuál, la interfaz de usuario está basada en componentes creados usando Direct3D y DirectInput. El mensaje estándar de Windows se utiliza para eventos de teclado. La interfaz como mínimo debe apoyarse de 16 bits y de 32 bits de resolución. La resolución principal debe ser 800x600, pero también soporta mayores resoluciones.

La interfaz descrita, está contenida en el motor gráfico 3D Irrlicht, del cual nos hemos apoyado para el desarrollo del juego.

Interfaz de hardware

El juego se ejecuta en un PC equipado con un Intel Core 2 Duo Procesador 2.0 Ghz, una tarjeta gráfica NVIDIA Geforce Go 7300 con 384 MB de memoria y compatible con Direct3D, una memoria RAM de 1 GB y una apropiada tarjeta de sonido.

Interfaz de software

El juego requiere del sistema operativo Windows XP Service Pack 2 y de la librería DirectX 9.c o posteriores para el correcto funcionamiento del juego.

Interfaz de comunicación

El usuario interactúa con el juego usando el ratón y el teclado.

Vistas de los requisitos funcionales

Esta sección describe los tres puntos de vista de los requisitos funcionales para QuakeGarage.

- Vista de la lista primaria de los requisitos funcionales. Contiene una lista organizada secuencialmente.
- Vista de los requisitos funcionales agrupadas de acuerdo al componente. Un componente representa una coherente agrupación de funcionalidades.
- Vista de casos de usos para las necesidades. Los casos de uso sirven de complemento a la lista de requisitos. Nos proporcionan un número de escenarios (un contexto de uso) para conocer mejor los requisitos funcionales.

Lista primaria de los requisitos funcionales

1. El software soportará un puntero de ratón personalizado.
2. El software soportará paneles, botones, menús e imágenes.
3. El software tendrá un menú principal con las siguientes opciones:
   - Game Mode
   - Bot Mode
   - Setup
   - Credit

- Exit

4. El software tendrá la capacidad de recuperarse de errores e informar de lo ocurrido.
5. El software se ejecutará sin el modo pantalla completa (windowed).
6. El software se ejecutará sólo en modo pantalla completa (fullscreen).
7. El software estará capacitado de interactuar con el ratón y el teclado.
8. El software soportará las siguientes configuraciones:
   - Movimientos
   - Uso de armas
   - Ver resultado
   - Uso de ítem especial
   - Salir de la partida.
   - Configuración del jugador (nombre, modelo, piel)

9. El software deberá cargar el mapa en no más de 20 segundos.
10. El software será capaz de cargar una partida del juego con la configuración establecida por el usuario.
11. El software soportará canales alfa (transparencia).
12. El software permitirá al usuario elegir cualquier nivel disponible.
13. El software tendrá su propio cargador de niveles o escenarios.
14. El software implementará caracteres, objetos y construcciones, cualquier objeto 3-D con mallas.
15. El software soportará diversos formatos de mallas:
    - Quake 3 levels (.bsp)
    - Quake 2 models (.md2)

16. El software soportará diversos formatos de texturas:
    - JPEG File Interchange Format (.jpg)
    - Portable Network Graphics (.png)
    - Truevision Targa (.tga)
    - Windows Bitmap (.bmp)
    - Zsoft Paintbrush (.pcx)

17. El software incluirá efectos visuales:
    - Superficies de agua realistas
    - Luces dinámicas
    - Sombras dinámicas
    - Objetos transparentes
    - Sistema de partículas: humo, fuego,…
    - Animación de texturas
    - Animación de mallas
    - Cielo

18. El software podrá deshabilitar efectos visuales.
19. El software soportará las siguientes opciones del juego:
    - Gráficos
    - Red (pendiente)
    - Sonido (Mono, estéreo)

- Volumen
- Cambiar la resolución
- Nivel de detalles de las mallas
- Nivel de detalles de las partículas
- Nivel de detalles de las texturas

20. El software soportará la reproducción de MP3.
21. El software permitirá al usuario seleccionar su propia música.
22. El software operará correctamente en ausencia de archivos de música.
23. El software soportará DirectSound y una variedad de formatos de sonidos, particularmente .wav y .mp3
24. El software soportará 16 canales.
25. El software permitirá al usuario deshabilitar la música de fondo.
26. El software podrá deshabilitar efectos de sonido.
27. El software permitirá la libre navegación a través la cámara del sistema.
28. El software permitirá al usuario tomar el control de la cámara del sistema.
29. El software fijará la cámara del sistema en una unidad física.
30. El software permitirá al usuario tomar el control de una unidad física del sistema.
31. El software permitirá a la IA tomar el control de una unidad física del sistema.
32. El software restringirá el movimiento de la unidad física dentro del mapa o escenario.
33. El software tendrá la capacidad de regular el movimiento de unidades físicas basado en la velocidad, gravedad, terreno, etc.…
34. El software permitirá al usuario ejecutar acciones de la unidad física haciendo uso de los diferentes botones o teclas.
35. El software permitirá que las unidades físicas adquieran armaduras, armas, vida, munición e ítems especiales.
36. El software controlará la existencia de ítems en el escenario.
37. El software controlará el inventario de cada unidad física
38. El software permitirá el uso de ciertos ítems, en concreto, las armas y los ítems especiales.
39. El software controlará el tiempo de disparo, daño producido y demás eventos del combate, actualizando los estados de cada contendiente.
40. El software permitirá reaparecer a unidad física que esté abatida.
41. El software soportará el modo singular donde el usuario jugará contra la IA.
42. El software tendrá una modalidad de juego donde pueda jugar contra sí mismo.
43. El software controlará datos de interés de la partida.
44. El software deberá caber en un solo disco junto con la documentación.
45. El software incluirá una guía de usuario
46. El software será implementado en C++/Win32.
47. El software soportará las siguientes prestaciones software/hardware:
    - Windows XP
    - Intel Core 2 Duo 2.0 Ghz
    - RAM 512 MB
    - NVIDIA Geforce Go 73000 384 MB
    - DirectX 9.c

48. El software soportará diferentes lenguajes para programar shaders:
    - Pixel y Vertex Shaders 1.1 to 3.0
    - HLSL
    - GLSL

Vista de componentes

Este punto de vista de los requisitos es compartido en mayor detalle con la descripción de diseño software para QuakeGarage. El diseño software de QuakeGarage proporciona una vista de componentes de las clases que implementan la funcionalidad. El desglose se entiende sólo como una primera exploración de las metas del juego.

Componente 1: Apertura del juego

Requisitos números: 1, 2, 3, 4, 6, 7, 10

- El software soportará un puntero de ratón personalizado
- El software soportará paneles, botones, menús e imágenes.
- El software tendrá un menú principal con las siguientes opciones.
  - Game Mode
  - Bot Mode
  - Setup
  - Credit
  - Exit
- El software tendrá capacidad de recuperarse de errores e informar de lo ocurrido.
- El software se ejecutará sólo en modo pantalla completa (fullscreen).
- El software estará capacitado de interactuar con el ratón y el teclado.
- El software será capaz de cargar una partida del juego con la configuración establecida por el usuario.

Componente 2: Gráficos

Componente 2.1: Objetos GUI en 2D

Requisitos números: 2, 11, 16

- El software soportará paneles, botones, menús e imágenes.
- El software tendrá soporte para los canales alfa (transparencia).
- El software soportará diversos formatos de texturas:
  - JPEG File Interchange Format (.jpg)
  - Portable Network Graphics (.png)
  - Truevision Targa (.tga)
  - Windows Bitmap (.bmp)
  - Zsoft Paintbrush (.pcx)

Componente 2.2: Constructor de niveles

Requisitos números: 6, 9, 13, 15

- El software se ejecutará sólo en modo pantalla completa (fullscreen).
- El software se deberá cargar el mapa en no más de 20 segundos.
- El software tendrá su propio cargador de niveles o escenarios.
- El software soportará diversos formatos de mallas:
  - Quake 3 levels (.bsp)
  - Quake 2 models (.md2)

Componente 2.3: Constructor de entidades

Requisitos números: 14, 15

- El software implementará caracteres, objetos y construcciones, cualquier objeto 3-D con mallas.
- El software soportará diversos formatos de mallas:
  - Quake 3 levels (.bsp)
  - Quake 2 models (.md2)

Componente 2.4: Efectos y animaciones

Requisitos números: 17, 18, 48

- El software incluirá efectos visuales:
- Superficies de agua realistas
  - Luces dinámicas
  - Sombras dinámicas
  - Objetos transparentes
  - Sistema de partículas: humo, fuego,…
  - Animación de texturas
  - Animación de mallas
  - Cielo
- El software podrá deshabilitar efectos visuales.
- El software soportará diferentes lenguajes para programar shaders:
  - Pixel y Vertex Shaders 1.1 to 3.0
  - HLSL
  - GLSL

Componente 3: Sonido

Requisitos números: 20, 21, 22, 23, 24, 25, 26

- El software soportará la reproducción de MP3.
- El software permitirá al usuario seleccionar su propia música.
- El software operará correctamente en ausencia de archivos de música.
- El software soportará DirectSound y una variedad de formatos de sonidos, particularmente .wav y .mp3
- El software soportará 16 canales.
- El software permitirá al usuario deshabilitar la música de fondo.
- El software podrá deshabilitar efectos de sonido.

Componente 4: Configuración del jugador

Requisitos números: 1, 2, 6, 7, 8

- El software soportará un puntero de ratón personalizado
- El software soportará paneles, botones, menús e imágenes.
- El software se ejecutará sólo en modo pantalla completa (fullscreen).
- El software estará capacitado de interactuar con el ratón y el teclado.
- El software soportará las siguientes configuraciones:
  - Movimientos
  - Uso de armas
  - Ver resultado
  - Uso de ítem especial
  - Salir de la partida
  - Configuración del perfil (nombre, modelo, piel)

Componente 5: Opciones del juego

Requisitos números: 1, 2, 6, 7, 19

- El software soportará un puntero de ratón personalizado
- El software soportará paneles, botones, menús e imágenes.
- El software se ejecutará sólo en modo pantalla completa (fullscreen).
- El software estará capacitado de interactuar con el ratón y el teclado.
- El software soportará las siguientes opciones del juego:
  - Gráficos
  - Red (pendiente)
  - Sonido (Mono, estéreo)

- Volumen
- Cambiar la resolución
- Nivel de detalles de las mallas
- Nivel de detalles de las partículas
- Nivel de detalles de las texturas

Componente 6: Navegación en el nivel

Requisitos números: 1, 6, 7, 11, 27, 28

- El software se ejecutará sólo en modo pantalla completa (fullscreen).
- El software soportará un puntero de ratón personalizado.
- El software estará capacitado de interactuar con el ratón y el teclado.
- El software tendrá soporte para los canales alfa (transparencia).
- El software permitirá la libre navegación a través la cámara del sistema.
- El software permitirá al usuario tomar el control de la cámara del sistema.

Componente 7: Unidades físicas

Requisitos números: 1, 2, 7, 11, 29, 30, 31, 32, 33, 34, 35, 38

- El software soportará un puntero de ratón personalizado.
- El software soportará paneles, botones, menús e imágenes.
- El software estará capacitado de interactuar con el ratón y el teclado.
- El software tendrá soporte para los canales alfa (transparencia).
- El software fijará la cámara del sistema en una unidad física.
- El software permitirá al usuario tomar el control de una unidad física del sistema.
- El software permitirá a la IA tomar el control de una unidad física del sistema.
- El software restringirá el movimiento de la unidad física dentro del mapa o escenario.
- El software tendrá la capacidad de regular el movimiento de unidades físicas basado en la velocidad, gravedad, terreno, etc.…
- El software permitirá al usuario ejecutar acciones de la unidad física haciendo uso de los diferentes botones o teclas.
- El software permitirá el uso de ciertos ítems a las unidades físicas, en concreto, las armas y los ítems especiales.

Componente 8: Ítems

Requisitos números: 35, 36, 37

- El software permitirá que las unidades físicas adquieran armaduras, armas, vida, munición e ítems especiales.
- El software controlará la existencia de ítems en el escenario.
- El software controlará el inventario de cada unidad física.

Componente 9: Combate

Requisitos números: 39, 40

- El software controlará el tiempo de disparo, daño producido y demás eventos del combate, actualizando los estados de cada contendiente.
- El software permitirá reaparecer a unidad física que esté abatida.

Componente 10: Inteligencia Artificial

Requisitos números: 31, 41, 42

- El software permitirá a la IA controlar una unidad física del sistema.
- El software soportará el modo singular donde el usuario jugará contra la IA.
- El software tendrá una modalidad de juego donde pueda jugar contra sí mismo.

Componente 11: Vista de las estadísticas

Requisitos números: 2, 6, 7, 11, 43

- El software se ejecutará sólo en modo pantalla completa (fullscreen).
- El software soportará paneles, botones, menús e imágenes.
- El software estará capacitado de interactuar con el ratón y el teclado.
- El software tendrá soporte para los canales alfa (transparencia).
- El software controlará datos de interés de la partida.

Requisitos no funcionales

Requisitos números: 44, 45, 46, 47

- El software deberá caber en un solo disco junto con la documentación.
- El software incluirá una guía de usuario
- El software soportará las siguientes prestaciones software/hardware:
  - Windows XP
  - Intel Core 2 Duo 2.0 Ghz
  - RAM 512 MB
  - NVIDIA Geforce Go 73000 384 MB
  - DirectX 9.c
- El software será implementado en C++/Win32.

Vista de casos de usos

Se presenta un conjunto completo de casos de uso para los requisitos funcionales de QuakeGarage. Cada caso de uso sirve para determinar la validez de un requisito o conjunto de requisitos y para recopilar información útil en la elaboración de un plan de test. Los casos de uso también proporcionan un punto de partida en el desarrollo del diseño.

Casos de Uso para el *usuario*

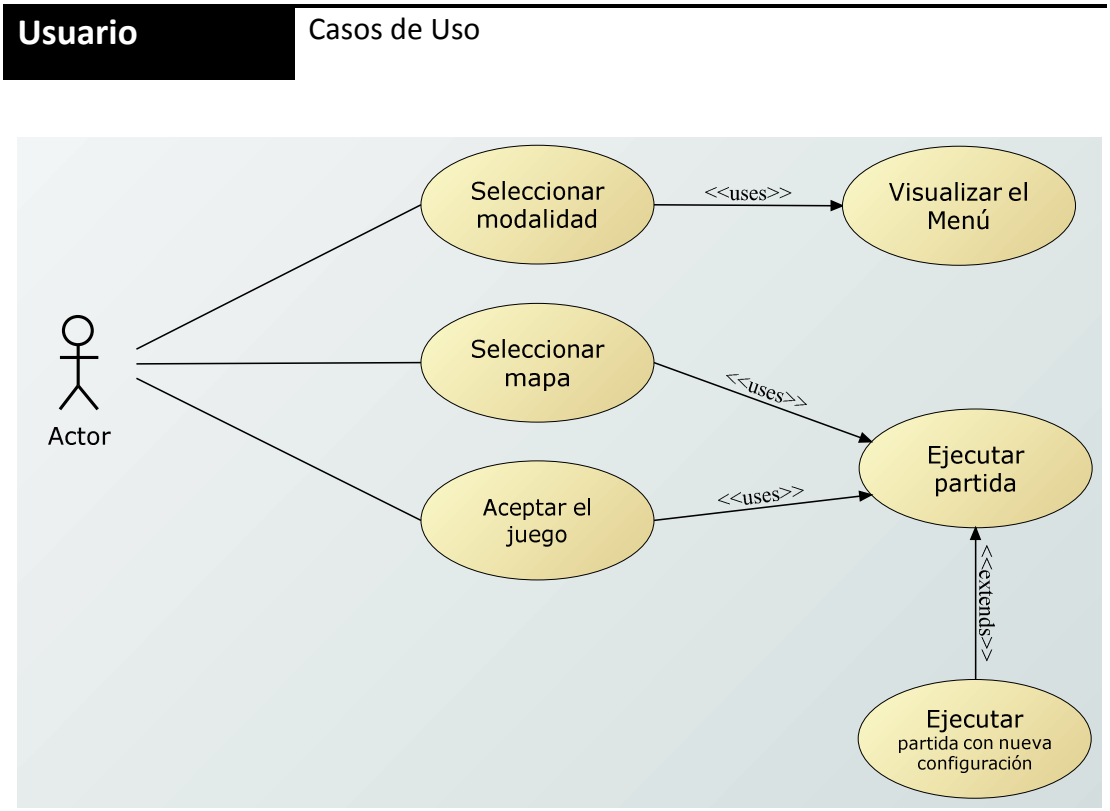


Figura 1. Casos de Uso para un usuario

Usuario	Casos de Uso

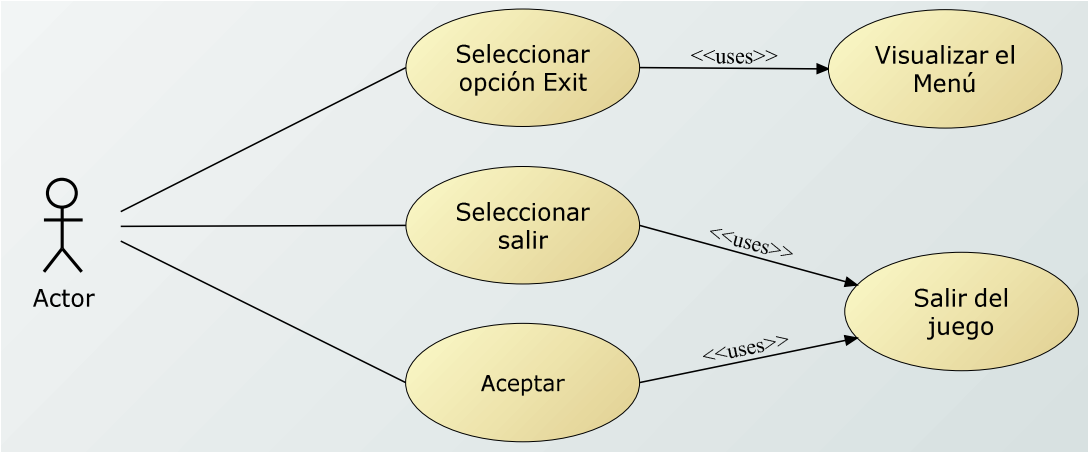


Figura 2. Casos de Uso para un usuario

Usuario	Casos de Uso

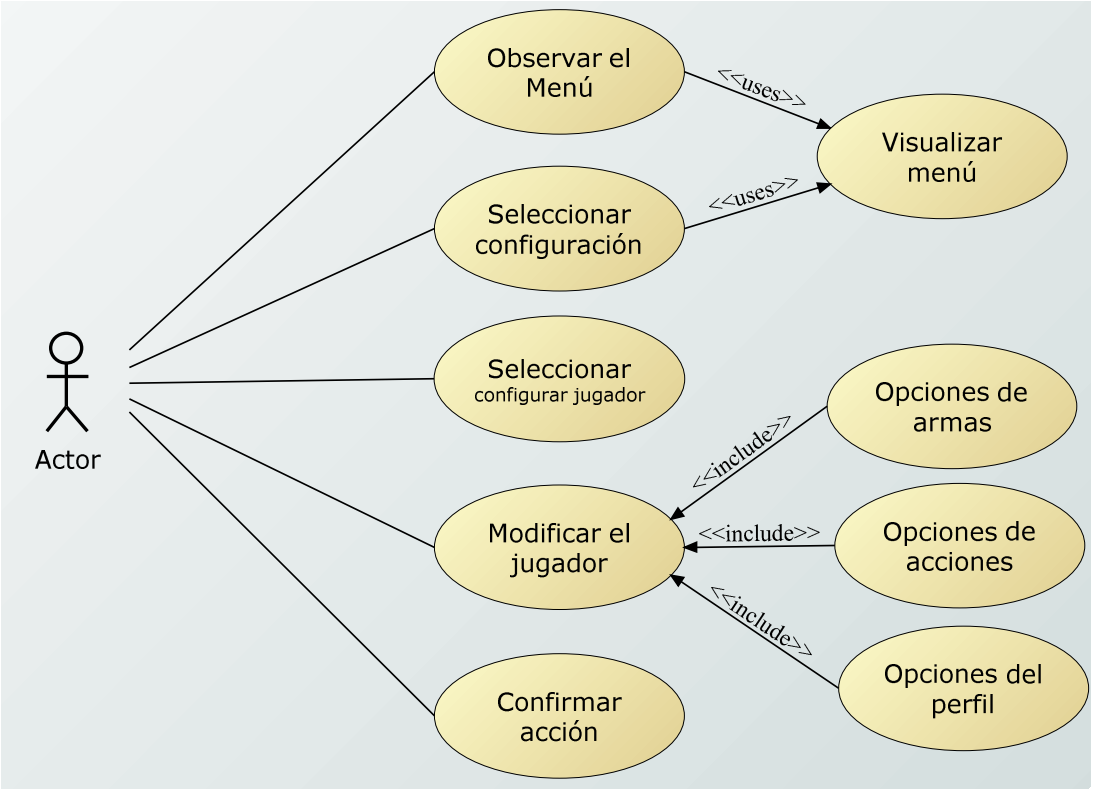


Figura 3. Casos de Uso para un usuario

Usuario	Casos de Uso

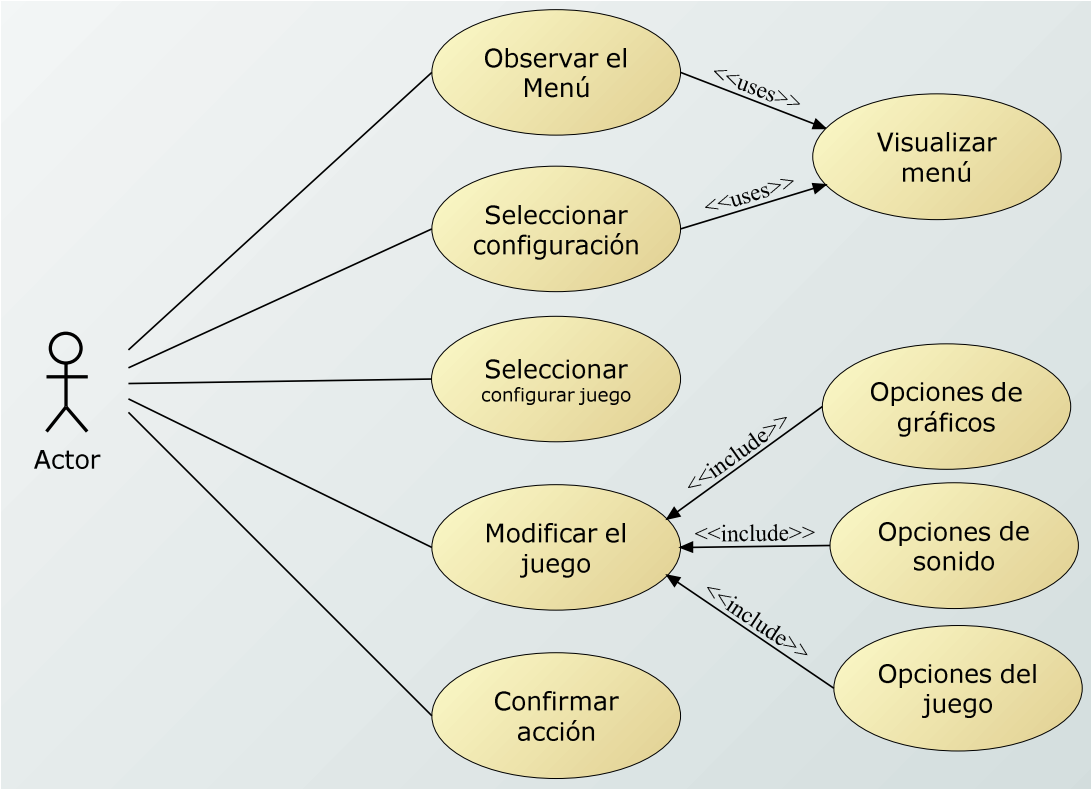


Figura 4. Casos de Uso para un usuario

Usuario | Casos de Uso

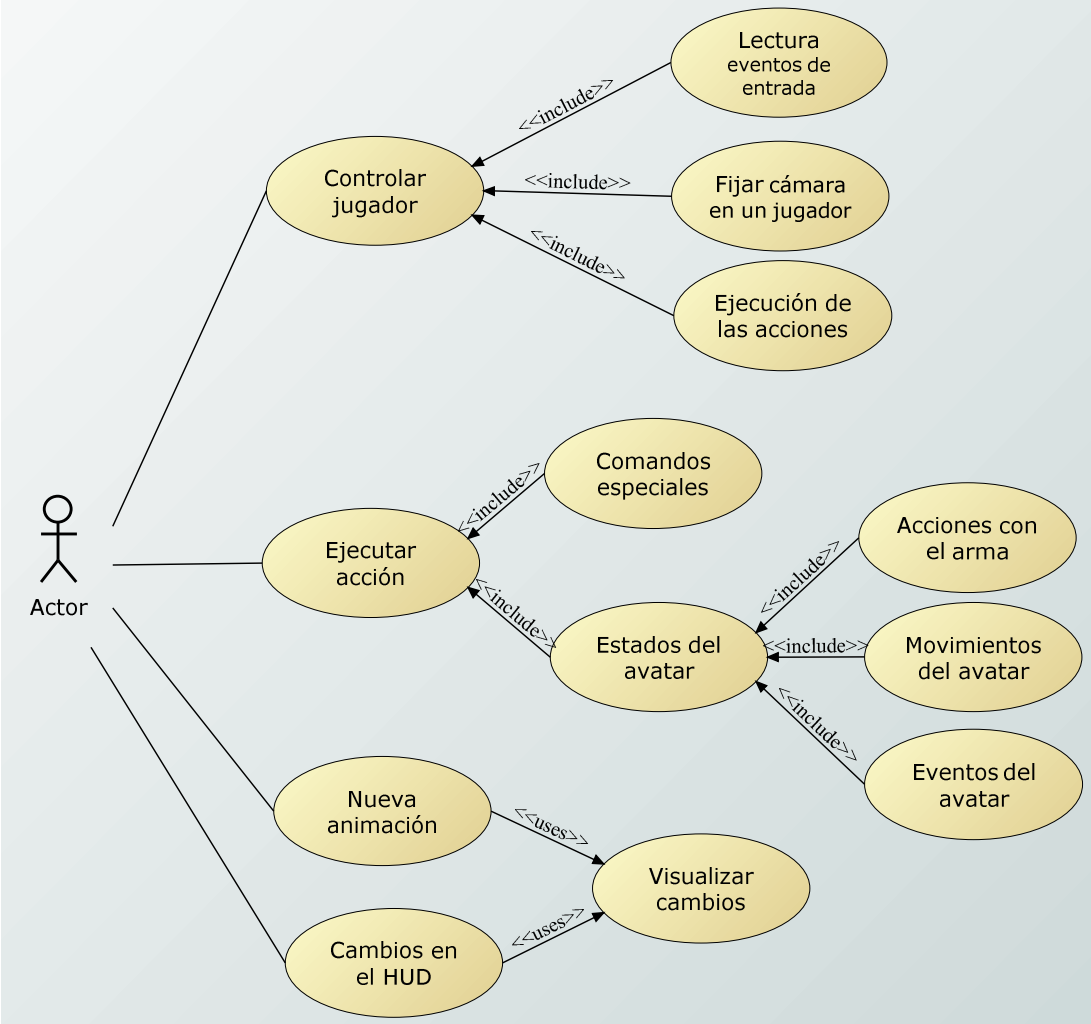


Figura 5. Casos de Uso para un usuario

Usuario Casos de Uso

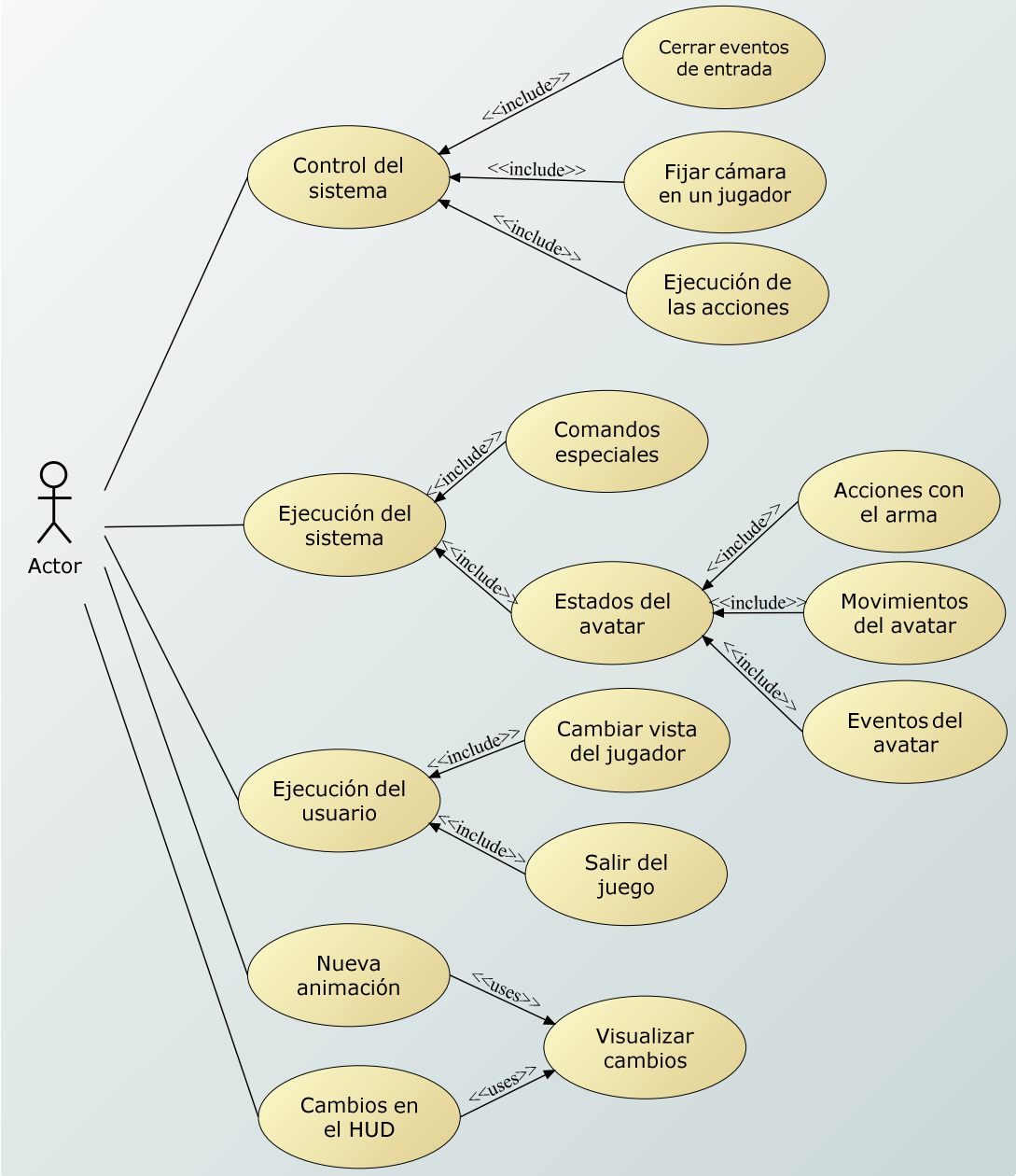


Figura 6. Casos de Uso para un usuario

Descripción de los casos de uso

Caso de uso 1: El usuario observa la pantalla no completa (windowed)
Requisitos explorado: #3, #5
Jugador (Actor) Contexto (Papel): Jugador
Precondición: El juego está en el menú
Poscondición: La pantalla del juego no ocupa toda la resolución
Cursos principales: 1.) El juego se carga. 2.) El juego fija la resolución a un tamaño adecuado que no ocupe toda la vista.
Cursos alternativos:
Cursos excepcionales: 1a1.). Los archivos del juego no existen o están corruptos. 1a2.). El ordenador no tiene una tarjeta 3D aceleradora.

Caso de uso 2: El usuario usa el puntero del sistema
Requisitos explorado: #1, #3
Jugador (Actor) Contexto (Papel): Jugador
Precondición: El juego está en el menú
Poscondición: El puntero del ratón es visible
Cursos principales: 1.) El usuario ve un puntero de ratón personalizado.
Cursos alternativos:
Cursos excepcionales: 1a.) Los gráficos para el puntero se perdieron.

Caso de uso 3: El usuario observa los objetos 2D-GUI
Requisitos explorado: #2, #11, #16
Jugador (Actor) Contexto (Papel): Jugador
Precondición: El juego ha sido iniciado
Poscondición: El usuario ve los objetos GUI
Cursos principales: 1.) El usuario observa paneles, botones, menús e imágenes.
Cursos alternativos:
Cursos excepcionales: 1a.) Los gráficos para los objetos 2D-GUI se perdieron.

Caso de uso 4: El usuario carga una partida
Requisitos explorado: #1, #2, #3, #6, #7, #9, #10,#12
Jugador (Actor) Contexto (Papel): Jugador
Precondición: El juego está en el menú
Poscondición: El usuario inicia una partida
Cursos principales: 1.) El sistema visualiza el menú principal del juego: Game mode, Bot mode, Setup, Credit, Exit 2.) El usuario selecciona una modalidad de juego (Game mode, Bot mode) 3.) El sistema reconoce la opción y le muestra los mapas de juego 4.) El usuario elige un mapa determinado y solicita empezar la lucha 5.) El sistema reconoce la petición y carga la partida con la configuración actual.
Cursos alternativos: 4a.) El usuario solicita volver atrás (al menú principal). 4b.) El sistema reconoce la orden y muestra el menú principal.
Cursos excepcionales:

Caso de uso 5: El usuario sale del juego
Requisitos explorado: #1, #2, #3, #6, #7
Jugador (Actor) Contexto (Papel): Jugador
Precondición: El juego está en el menú
Poscondición: El usuario acaba el juego
Cursos principales: 1.) El sistema visualiza el menú principal del juego: Game mode, Bot mode, Setup, Credit, Exit. 2.) El usuario selecciona la opción Exit. 3.) El sistema reconoce la opción y espera confirmación del usuario. 4.) El usuario confirma la elección. 5.) El sistema finaliza.
Cursos alternativos: 4a.) El usuario opta no salir del juego. 4b.) El sistema reconoce la orden y muestra el menú principal.
Cursos excepcionales:

Caso de uso 6: El usuario observa los créditos
Requisitos explorado: #1, #2, #3, #6, #7
Jugador (Actor) Contexto (Papel): Jugador
Precondición: El juego está en el menú
Poscondición: El usuario ve los créditos
Cursos principales: 1.) El sistema visualiza el menú principal del juego: Game mode, Bot mode, Setup, Credit, Exit. 2.) El usuario selecciona la opción Credit. 3.) El sistema reconoce la opción y muestra la pantalla de créditos. 4.) El usuario permanece en los créditos
Cursos alternativos: 4a.) El usuario vuelve al menú principal. 4b.) El sistema reconoce la orden y muestra el menú principal.
Cursos excepcionales:

Caso de uso 7: El usuario observa las configuraciones
Requisitos explorado: #1, #2, #3, #6, #7
Jugador (Actor) Contexto (Papel): Jugador
Precondición: El juego está en el menú
Poscondición: El usuario ve las opciones de configuración
Cursos principales: 1.) El sistema visualiza el menú principal del juego: Game mode, Bot mode, Setup, Credit, Exit. 2.) El usuario selecciona la opción Setup. 3.) El sistema reconoce la opción y muestra la pantalla de configuración. 4.) El usuario elige alguna opción de configuración
Cursos alternativos: 4a.) El usuario opta volver al menú principal. 4b.) El sistema reconoce la orden y muestra el menú principal.
Cursos excepcionales:

Caso de uso 8: El usuario carga un mapa
Requisitos explorado: #6, #9, #13, #15
Jugador (Actor) Contexto (Papel): Jugador
Precondición: El escenario está elegido
Poscondición: El usuario observa el mapa cargado
Cursos principales: 1.) El sistema carga la malla del mapa elegido en el formato .bsp 2.) El sistema almacena la geometría del mapa en una estructura Octree. 3.) El sistema almacena los shaders del mapa. 4.) El sistema añade el cielo. 5.) El sistema almacena la información de las entidades dentro del mapa. 6.) El sistema renderiza el escenario. 7.) El usuario puede contemplar el escenario con la pantalla completa.
Cursos alternativos:
Cursos excepcionales: 1a.) Los archivos del juego no existen o están corruptos. 6a1.) El ordenador no tiene una tarjeta 3D aceleradora. 6a2.) La tarjeta gráfica esta rota.

Caso de uso 9: El usuario observa las mallas de objetos
Requisitos explorado: #14, #15
Jugador (Actor) Contexto (Papel): Jugador
Precondición: La partida se está ejecutando
Poscondición: El usuario observa objetos 3D
Cursos principales: 1.) El sistema carga la malla de un objeto en el formato .md2 2.) El sistema almacena la geometría del objeto. 3.) El sistema almacena las texturas del objeto. 4.) El sistema renderiza el objeto. 5.) El usuario puede contemplar el objeto.
Cursos alternativos:
Cursos excepcionales: 1a.) Los archivos del juego no existen o están corruptos. 4a1.) El ordenador no tiene una tarjeta 3D aceleradora. 4a2.) La tarjeta gráfica esta rota.

Caso de uso 10: El usuario observa efectos y animaciones
Requisitos explorado: #16, #18, #46
Jugador (Actor) Contexto (Papel): Jugador
Precondición: El jugador está dentro de la partida
Poscondición: El usuario ve determinados efectos
Cursos principales: 1.) El sistema carga un emisor de partículas, algún efecto o animación. 2.) El sistema renderiza el efecto cargado. 3.) El usuario contempla el efecto.
Cursos alternativos:
Cursos excepcionales: 1a.) Los archivos del juego no existen o están corruptos. 2a1.) El ordenador no tiene una tarjeta 3D aceleradora. 2a2.) La tarjeta gráfica esta rota.

Caso de uso 11: El usuario reproduce música
Requisitos explorado: #20, #21, #22, #23, #25
Jugador (Actor) Contexto (Papel): Jugador
Precondición: el juego está iniciado. El sistema soporta una tarjeta de sonido.
Poscondición: El usuario escucha música de fondo.
Cursos principales: 1.) El usuario dispondrá de un directorio para poner MP3. 2.) El usuario escuchará música de fondo durante la partida.
Cursos alternativos: 2a1.) El usuario no escucha música porque no hay archivos MP3. 2a2.) El usuario no escucha música porque ha desactivado la música de fondo.
Cursos excepcionales: 2a.) Un archivo MP3 corrupto causa el colapso del juego.

Caso de uso 12: El usuario escucha distintos sonidos
Requisitos explorado: #22, #23, #24, #26
Jugador (Actor) Contexto (Papel): Jugador
Precondición: La partida está iniciada. Las DirectX están instaladas.
Poscondición: El sonido del juego se reproduce
Cursos principales: 1.) El sistema usa DirectSound para cargar un archivo de audio. 2.) El sistema usa DirectSound para reproducir el sonido. 3.) El sistema mezcla el sonido en 16 canales. 4.) El usuario escucha sonido(s).
Cursos alternativos: 2a.) El sistema ha deshabilitado el efecto de sonido 4a.) El usuario ha silenciado el sonido.
Cursos excepcionales: 1a.) Las DirectX fallan (habrá que reinstalar). 1b.) La tarjeta de sonido no es soportado por DirectX.

Caso de uso 13: El usuario no escucha música de fondo
Requisitos explorado: #25
Jugador (Actor) Contexto (Papel): Jugador
Precondición: La partida está iniciada.
Poscondición: El juego no tiene música de fondo.
Cursos principales: 1.) El sistema intenta reproducir música de fondo. 2.) El sonido de fondo está deshabilitado en las configuraciones del jugador. 3.) El sistema no reproduce música de fondo.
Cursos alternativos: 2a.) El sonido de fondo no está deshabilitado pero el usuario ha silenciado el sonido 2b.) El sistema reproduce música de fondo pero no se escucha nada.
Cursos excepcionales:

Caso de uso 14: El usuario configura al jugador
Requisitos explorado: #1, #2, #6, #7, #8
Jugador (Actor) Contexto (Papel): Jugador
Precondición: El juego está en las configuraciones (Setup) del menú
Poscondición: El usuario cambia la configuración
Cursos principales: 1.) El sistema visualiza el menú de configuración: Player, Controls, System, Game option, Default 2.) El usuario elige una opción de configuración del jugador: Player o Controls. - El perfil del jugador se configura con la opción Player. - Los movimientos, las armas u otras configuraciones con la opción Controls 3.) El sistema reconoce la elección y visualiza el contenido 4.) El usuario cambia las opciones.
Cursos alternativos: 2a.) El usuario cambia la configuración a la que estaba por defecto (Default). 2b.) El sistema pide confirmación. 2c1.) El usuario acepta, la configuración se modifica y se vuelve al menú Setup. 2c2.) El usuario no acepta, la configuración no se modifica y se vuelve al menú Setup. 4a.) El usuario opta volver atrás, al menú de configuración (Setup). 4b.) El sistema reconoce la orden y muestra el menú de configuración.
Cursos excepcionales:

Caso de uso 15: El usuario configura las opciones de juego
Requisitos explorado: #1, #2, #6, #7, #19
Jugador (Actor) Contexto (Papel): Jugador
Precondición: El juego está en las configuraciones (Setup) del menú
Poscondición: El usuario cambia la configuración
Cursos principales: 1.) El sistema visualiza el menú de configuración: Player, Controls, System, Game option, Default 2.) El usuario elige una opción de configuración del juego: System o Game option. - Los gráficos, la red y el sonido se configura con la opción System. - Las demás se configura con la opción Game option. 3.) El sistema reconoce la elección y visualiza el contenido 4.) El usuario cambia las opciones.
Cursos alternativos: 2a.) El usuario cambia la configuración a la que estaba por defecto (Default). 2b.) El sistema pide confirmación. 2c1.) El usuario acepta, la configuración se modifica y se vuelve al menú Setup. 2c2.) El usuario no acepta, la configuración no se modifica y se vuelve al menú Setup. 4a.) El usuario opta volver atrás, al menú de configuración (Setup). 4b.) El sistema reconoce la orden y muestra el menú de configuración.
Cursos excepcionales:

Caso de uso 16: El usuario navega por el nivel
Requisitos explorado: #1, #6, #7, #11, #27, #28
Jugador (Actor) Contexto (Papel): Jugador
Precondición: La partida se está ejecutando en modo observador
Poscondición: El usuario observa las diferentes partes del escenario
Cursos principales: 1.) El sistema permitirá la libre navegación de la cámara por el escenario 2.) El sistema le dará el control de la cámara al usuario. 3.) El usuario usará el cursor como guía para navegar a través del nivel 4.) El usuario podrá ver el mapa en diferentes perspectiva de la cámara
Cursos alternativos:
Cursos excepcionales: 4a.) Los objetos no revelan diferentes perspectivas.

Caso de uso 17: El usuario controla una unidad física (jugador)
Requisitos explorado: #7, #28, #29, #32, #33, #34
Jugador (Actor) Contexto (Papel): Jugador
Precondición: La partida se ha iniciado.
Poscondición: El usuario controla las acciones de una unidad física
Cursos principales: 1.) El sistema permitirá al usuario tomar el control de una unidad física. 2.) El sistema fijará la cámara en la unidad física del usuario. 3.) El usuario usará el ratón para mover la cámara y los botones o teclas para realizar diferentes acciones: - Movimiento (saltar, agacharse, desplazamiento) - Armas (cambio de armas, disparar) - Usar item especial. 4.) El sistema controlará la ejecución de cada acción, según la lógica implementada.
Cursos alternativos:
Cursos excepcionales: 3a.) La interfaz de entrada (teclado y ratón) no responde.

Caso de uso 18: El usuario hace cambiar el arma del jugador
Requisitos explorado: #2, #7, #11, #30, #34
Jugador (Actor) Contexto (Papel): Jugador
Precondición: La partida se ha iniciado y el usuario controla una unidad física.
Poscondición: El jugador cambia el arma.
Cursos principales: 1.) El usuario acciona la tecla de cambio de arma. 2.) El sistema reconoce la orden y procede a cambiar el arma. 3.) El sistema desatiende algunas ordenes hasta la conclusión del proceso. 4.) El usuario puede observar un arma diferente y los objetos GUI-2D del nuevo arma.
Cursos alternativos: 2a.) El sistema no puede cambiar el arma. 2b.) El jugador sigue en el mismo estado, nada ha cambiado.
Cursos excepcionales:

Caso de uso 19: El usuario hace disparar el arma del jugador
Requisitos explorado: #2, #7, #11, #30, #34
Jugador (Actor) Contexto (Papel): Jugador
Precondición: La partida se ha iniciado y el usuario controla una unidad física.
Poscondición: El jugador realiza un disparo.
Cursos principales: 1.) El usuario acciona la tecla de disparo del arma. 2.) El sistema reconoce la orden y procede a lanzar el “proyectil”. 3.) El sistema desatiende algunas ordenes hasta la conclusión del proceso. 4.) El usuario puede observar la munición en las imágenes GUI. 5.) El sistema controlará el recorrido del proyectil hasta su fin.
Cursos alternativos: 2a.) El sistema no puede disparar porque no hay suficiente munición. 2b.) El jugador sigue en el mismo estado, nada ha cambiado.
Cursos excepcionales:

Caso de uso 20: El usuario hace agachar al jugador
Requisitos explorado: #7, #30, #32, #33, #34
Jugador (Actor) Contexto (Papel): Jugador
Precondición: La partida se ha iniciado y el usuario controla una unidad física.
Poscondición: El jugador se agacha
Cursos principales: 1.) El usuario acciona la tecla de disparo de agacharse 2.) El sistema reconoce la orden y ejecuta el movimiento. 3.) El sistema desatiende algunas ordenes hasta la conclusión del proceso. 4.) El sistema controla las colisiones con el escenario, a la vez que ejecuta el movimiento. 5.) El usuario puede observar el cambio de posición.
Cursos alternativos: 2a.) El sistema no ejecuta el movimiento porque ya se encuentra en ese estado. 2b.) El jugador sigue en el mismo estado, nada ha cambiado.
Cursos excepcionales:

Caso de uso 21: El usuario hace saltar al jugador
Requisitos explorado: #7, #30, #32, #33, #34
Jugador (Actor) Contexto (Papel): Jugador
Precondición: La partida se ha iniciado y el usuario controla una unidad física.
Poscondición: El jugador salta
Cursos principales: 1.) El usuario acciona la tecla de salto. 2.) El sistema reconoce la orden y ejecuta el movimiento. 3.) El sistema desatiende algunas ordenes hasta la conclusión del proceso. 4.) El sistema controla las colisiones con el escenario, a la vez que ejecuta el movimiento. 5.) El usuario puede observar el cambio de posición.
Cursos alternativos: 2a.) El sistema no ejecuta el movimiento porque ya se encuentra en ese estado. 2b.) El jugador sigue en el mismo estado, nada ha cambiado.
Cursos excepcionales:

Caso de uso 22: El usuario hace usar un ítem especial al jugador
Requisitos explorado: #2, #7, #11, #30, #34, #38
Jugador (Actor) Contexto (Papel): Jugador
Precondición: La partida se ha iniciado y el usuario controla una unidad física.
Poscondición: El jugador cambia de estado
Cursos principales: 1.) El usuario acciona la tecla de usar un ítem. 2.) El sistema reconoce la orden y ejecuta la acción. 3.) El sistema desatiende algunas ordenes hasta la conclusión del proceso. 4.) El usuario puede observar el cambio de estado en las imágenes GUI.
Cursos alternativos: 2a.) El sistema no puede accionar el ítem porque no tiene. 2b.) El jugador sigue en el mismo estado, nada ha cambiado.
Cursos excepcionales:

Caso de uso 23: El usuario ve el resultado del juego
Requisitos explorado: #2, #7, #11, #34, #43
Jugador (Actor) Contexto (Papel): Jugador
Precondición: La partida se ha iniciado.
Poscondición: El usuario observa el resultado
Cursos principales: 1.) El usuario acciona la tecla para ver el resultado 2.) El sistema reconoce la orden y ejecuta la acción. 3.) El sistema desatiende algunas ordenes hasta la conclusión del proceso. 4.) El usuario puede observar las puntuaciones en las imágenes GUI.
Cursos alternativos: 2a.) El sistema comprueba que el usuario presionó la tecla para ver los resultados. 2b.) El sistema quita los resultados. 2c.) El usuario continúa con la pantalla original del juego.
Cursos excepcionales:

Caso de uso 24: El usuario sale de la partida
Requisitos explorado: #2, #7, #11, #34
Jugador (Actor) Contexto (Papel): Jugador
Precondición: La partida se ha iniciado.
Poscondición: El usuario accede al menú principal.
Cursos principales: 1.) El usuario acciona la tecla para salir de la partida 2.) El sistema reconoce la orden y ejecuta la acción. 3.) El sistema desatiende algunas ordenes hasta la conclusión del proceso. 4.) El usuario puede observar el menú del juego.
Cursos alternativos:
Cursos excepcionales:

Caso de uso 25: El usuario hace que el jugador adquiera alguna entidad
Requisitos explorado: #35, #36, #37
Jugador (Actor) Contexto (Papel): Jugador
Precondición: La partida se ha iniciado y el usuario controla una unidad física.
Poscondición: El jugador adquiere algún elemento.
Cursos principales: 1.) El usuario colisiona con algún ítem (escudo, arma, vida, munición, ítem especial) 2.) El sistema verifica la colisión con la entidad y comprueba que jugador a colisionado. 3.) El sistema hace desaparecer el ítem y controla su nueva aparición. 4.) El sistema actualiza el inventario del jugador con la nueva entidad. 5.) El usuario puede observar en las imágenes GUI, su nueva adquisición
Cursos alternativos: 3a.) El sistema no modifica el ítem ya que el usuario no lo puede coger. 3b.) El inventario del jugador sigue en el mismo estado, nada ha cambiado.
Cursos excepcionales:

Caso de uso 26: El usuario observa al jugador siendo disparado
Requisitos explorado: #2, #11, #30, #39
Jugador (Actor) Contexto (Papel): Jugador
Precondición: La partida se ha iniciado y el usuario controla una unidad física.
Poscondición: El jugador recibe un disparo.
Cursos principales: 1.) El usuario colisiona con algún proyectil o está dentro del radio de colisión. 2.) El sistema verifica la colisión con el proyectil y comprueba que jugador a colisionado. 3.) El sistema hace desaparecer el proyectil con alguna animación 4.) El sistema actualiza el daño producido y la reacción del jugador. 5.) El usuario puede observar su nuevo estado en las imágenes GUI.
Cursos alternativos:
Cursos excepcionales:

Caso de uso 27: El usuario ve al jugador morir y reaparecer
Requisitos explorado: #2, #7, #11, #30, #39
Jugador (Actor) Contexto (Papel): Jugador
Precondición: La partida se ha iniciado y el usuario controla una unidad física.
Poscondición: El jugador muere y reaparece en el mapa.
Cursos principales: 1.) El sistema verifica que la salud del jugador es menor o igual a 0 2.) El sistema inicializa la muerte del jugador y le quita el control al usuario. 3.) El usuario observa la muerte del jugador. 4.) El sistema espera unos segundos y a continuación permite reaparecer al jugador. 5.) El usuario acciona la tecla para que el jugador reaparezca. 6.) El sistema inicializa el estado del jugador, lo posiciona en un lugar arbitrario del mapa, y le devuelve el control al usuario.
Cursos alternativos: 5a1.) El usuario permanece ocioso y el jugador sigue en estado de muerte. 5a2.) El usuario ejecuta el comando para salir de la partida. 5a3.) El usuario ejecuta el comando para ver los resultados.
Cursos excepcionales:

Caso de uso 28: El usuario se enfrenta a la I.A.
Requisitos explorado: #31, #41
Jugador (Actor) Contexto (Papel): Jugador
Precondición: La partida se ha iniciado y el usuario controla una unidad física.
Poscondición: El usuario intenta vencer en el juego.
Cursos principales: 1.) El usuario emplea las estrategias de combate contra la I.A. 2.) El sistema controla la I.A. y se encargará de responder.
Cursos alternativos:
Cursos excepcionales:

Caso de uso 29: El usuario observa un combate I.A. vs I.A.
Requisitos explorado: #29, #30, #31, #41, #42
Jugador (Actor) Contexto (Papel): Jugador
Precondición: La partida se ha iniciado y el sistema controla el juego.
Poscondición: El usuario observa una partida.
Cursos principales: 1.) El sistema permitirá al usuario observar a una unidad física. 2.) El sistema fijará la cámara en la unidad física que el usuario eligió. 3.) El usuario usará el ratón y las teclas para cambiar las posiciones de la cámara. 4.) El sistema controlará la ejecución de cada acción, según la lógica implementada. 5.) Las I.A. combaten entre ellas.
Cursos alternativos:
Cursos excepcionales:

Restricciones de diseño

En esta sección se enumera las restricciones de diseño. Las restricciones de diseño son aquellos factores que afectan a la meta del proyecto o que pueden impactar en la planificación de desarrollo para el producto.

Disponibilidad

Un único paquete de instalación permite al usuario instalar el juego. Después de instalarlo el juego estará disponible para el usuario del sistema en todo momento.

Seguridad

No hay herramientas de seguridad.

Mantenimiento

El juego dispondrá de un servicio de mantenimiento para que se pueda actualizar. Los elementos de mantenimiento estarán disponibles para su descarga desde la página web.

Si un parche o una expansión son usados, el ejecutable puede ser sobrescrito. Es por ello, que los usuarios deben saber que los archivos de datos no deben ser sobrescritos durante las actualizaciones.

5. Modelo de diseño

Diseño del software

Es la fase donde se desarrollan las actividades de diseño para construir sistemas estables. De esto depende el éxito de la realización del software y la facilidad del mantenimiento.

En este documento se describe el diseño de un videojuego. La presentación del documento le proporciona una manera de ver cómo está construido el software, en términos de su diseño arquitectónico y su construcción en componentes.
Además, se tendrá presente los conceptos de diseño que garantizan la calidad; Abstracción, refinamiento, modularidad, arquitectura del software, jerarquía de control, ocultación de información, independencia funcional, cohesión y acoplamiento.

Diseño arquitectónico

En esta sección se descomponen y describen las diferentes partes que forman la arquitectura del sistema. Se trata del proceso de diseño inicial donde se identifican los subsistemas y se establece el marco de trabajo para el control y comunicación de los mismos. Las actividades principales son:

- Arquitectura del videojuego. Estructuración del sistema en varios subsistemas principales.
- Modelado del control.
- Descomposición modular: Cada subsistema se descompone en módulos interconectados

Arquitectura del videojuego

Con el aumento de actitudes hacia el uso de diferentes motores de juego, se puede observar que muchos, en el mercado, comparten un conjunto definido de funcionalidades, y en su mayoría los patrones de diseño. El objetivo de este apartado es obtener una arquitectura de referencia, específicamente para juegos con perspectiva 3D en primera persona (FPS). Una arquitectura de referencia es útil para el desarrollo de nuevos motores de juego manteniendo la coherencia entre la fase de diseño y la fase de implementación.

Antecedentes

Los juegos han dado el primer mercado para las técnicas avanzadas de gráficos. El costo para desarrollar, cada vez más realista, simulaciones ha crecido tanto que los desarrolladores de videojuegos, ya no pueden confiar en la recuperación de su inversión con un único juego. Esto ha dado lugar a la aparición de motores de videojuegos, que fueron creados con propósitos lo suficientemente generales como para ser usado por una familia de juegos similares. De hecho, el motor de un juego es como un motor de un automóvil. Se puede sacar del coche y construir otro para el mismo. El motor del juego es una colección de código de simulación que no especifica directamente el comportamiento del juego o el entorno. El motor incluye módulos como el de entrada / salida, los gráficos en 2D o en 3D, física, detección de colisiones, IA, sonido, y bases de datos. Diferentes juegos tendrá todos o algunos de los módulos mencionados.

Un típico FPS es un juego de disparo, con una perspectiva en primera persona que se caracteriza porque pueden ser extremadamente inmersivo, es por ello que se utilizan motores 3D que permitan crear juegos tan realistas como sea posible.

Básicamente, un juego es una base de datos en tiempo real con una bonita interfaz. Principalmente se compone de un bucle infinito, que procesa todos los objetos en el juego y a continuación, dibuja el siguiente marco (frame) de animación. Esto se repite a una frecuencia adecuada para el ojo humano, es lo que se conoce como los "frames per second". En la primera etapa de un bucle, el juego arranca y se inicializa todas sus variables. El programa también configura las estructuras de datos, asigna memoria y carga los archivos de gráficos y de audio.

- En el menú principal, el usuario podrá seleccionar o cambiar las diferentes opciones. Una vez que se haya iniciado el juego, el programa cargará la nueva configuración de la partida, así como también los archivos gráficos y de audio que requieran la misma partida.
- En el bucle principal del juego, la lógica del juego se reinicia para el próximo marco (frame). En este punto, el programa almacena la entrada del jugador (eventos de ratón y teclado). En respuesta a las acciones del jugador el programa ejecuta la lógica interna del juego, esto es, mueve todos los objetos, procesa la IA, lleva a cabo la detección de colisiones, y así sucesivamente. En esta etapa, el marco (frame) de la animación actual está listo para ser dibujado "renderizado". El programa dibuja todos los objetos en una parte invisible de la memoria (backbuffer). Previamente se volcó los datos que contenían el backbuffer en el frontbuffer, ya que el frontbuffer es la parte visible de la memoria que mostrará por pantalla el estado actual del juego.

En resumen, se podría decir que existe un BUCLE PRINCIPAL de la aplicación, cuyos pasos más relevantes son:

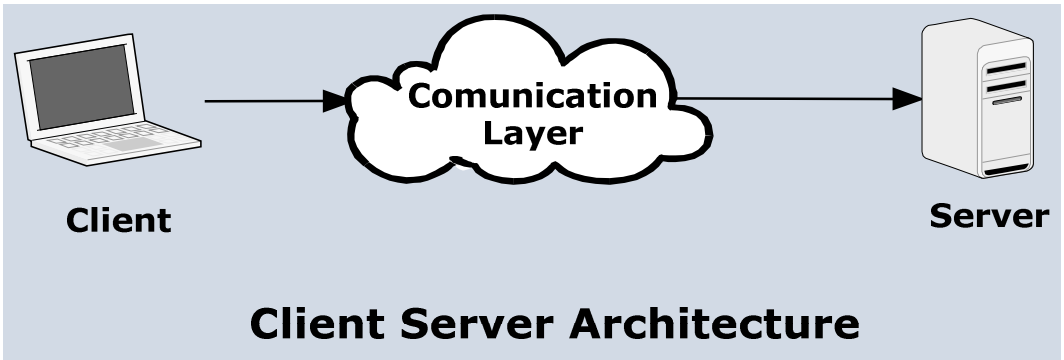
1. Leer entrada usuario
2. Actualizar la lógica del juego
   a. Procesar los comandos pendientes
   b. Actualizar las entidades
   c. Atender los eventos (de la partida) pendientes
3. Dibujar (GUI).

Estudio de una arquitectura existente

A continuación se analizara una arquitectura existente de un juego FPS, en concreto, el Quake II. Posee una arquitectura muy presente en todos los juegos, cliente-servidor. Todas las comunicaciones entre el cliente y el servidor se realizan a través de la capa de comunicaciones. Esta capa de abstracción es conveniente para permitir la extensibilidad de futuros protocolos. Cuando el modo de jugador es único, la capa de comunicaciones está integrada en el mismo computador, el cliente y el servidor. En modo multijugador, toda la comunicación se realiza a través de un protocolo de red.

La separación entre cliente y servidor es de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni contiene un sólo programa. Mediante la arquitectura Cliente-Servidor se consigue una centralización del control, de tal forma que los accesos, los recursos y la integridad de los datos son controlados por el servidor.

Así mismo, aumenta notablemente la escalabilidad del sistema, junto con sus prestaciones, y por tanto, la capacidad de los clientes y el servidor se pueden aumentar por separado. Al estar distribuidas las funciones y responsabilidades entre varios ordenadores independientes, es posible reemplazar, reparar, actualizar, o incluso trasladar un servidor, mientras que sus clientes no se verán afectados por ese cambio (o se afectarán mínimamente). Esta independencia de los cambios se conoce como encapsulación.

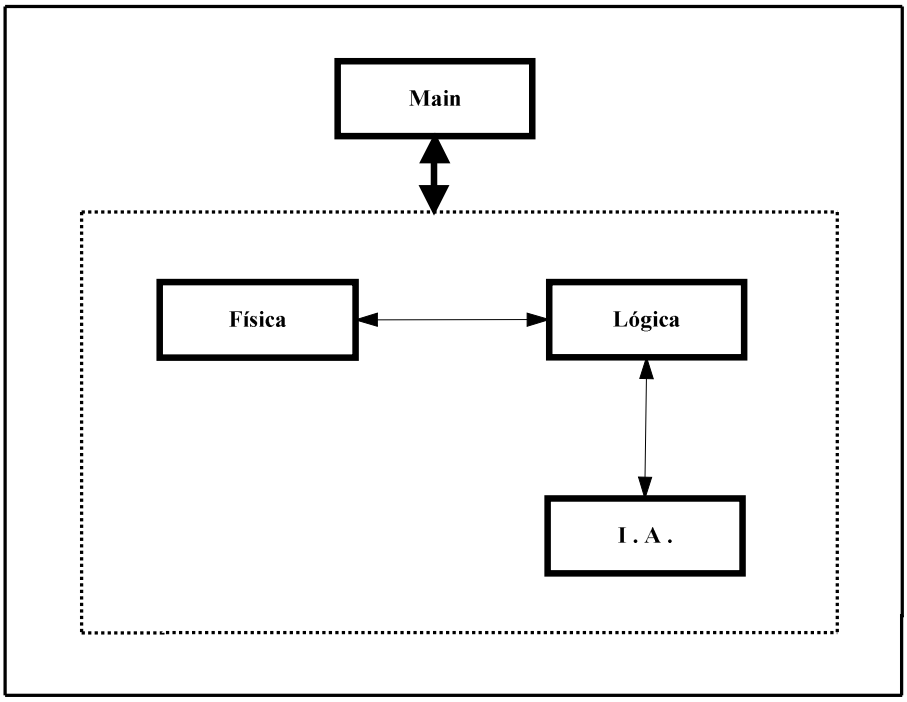


Arquitectura del Servidor Quake II

El servidor Quake II mantiene la base del tiempo y estados de los juegos, calcula el movimiento de los objetos, la física y ejecuta la IA para todos los personajes controlados por el ordenador. Se puede dividir en el motor del juego y la parte específica del juego. En Quake II los subsistemas de Física e Inteligencia Artificial son considerados como materiales específicos de juego y se encuentran en el código del juego. Esta abstracción permite a los desarrolladores el uso del motor del Quake II para desarrollar juegos con diferentes IA y reglas de física, por tanto, es flexible y reutilizable. El código específico del juego se compila y se enlaza en una DLL separada, llamada game.dll. Cualquier interacción entre esta parte del servidor y la parte del motor se realiza mediante el interfaz estándar de la DLL. Esto se asemeja al uso del patrón de diseño Facade para este subsistema. Una estructura global de punteros a funciones es la manera estándar de interacción entre el motor y parte específica del juego.

El análisis de la arquitectura revela los siguientes subsistemas:

- Subsistema de la lógica: responsable de almacenar la información referente a mapas, entidades y niveles.
- Subsistema de la física: regula la física de todo el mundo. Normas como la velocidad, fricción y la gravedad son fijados por este subsistema en el interior del entorno virtual.
- Subsistema de la IA



Quake 2 Server Architecture

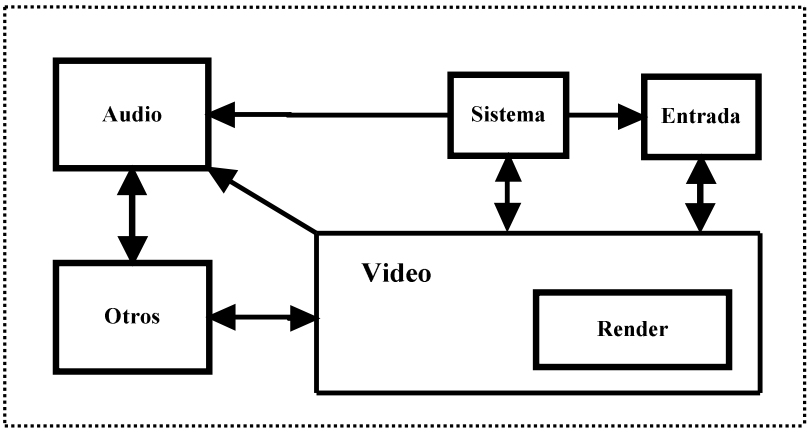
Arquitectura del Cliente Quake II

Una arquitectura concreta del cliente de Quake II se muestra en la siguiente figura. Se trata del componente principal de interacción con el usuario. Es responsable de la inmersión del jugador a través del audio y de la estimulación visual. En cada frame, cada cliente recoge la entrada del usuario (por ejemplo, un teclado y un ratón) y los envía al servidor.

El acoplamiento del renderizado en el interior del subsistema de vídeo se hace del siguiente modo. En Quake II el renderizado es compilado y vinculado como una DLL independiente. Todas las llamadas a este subsistema se pasan a través de la interfaz estándar DLL, esto permite la sustitución del módulo de renderizado. El usuario puede cambiar de sistema de renderizado de OpenGL a Software durante el juego.

El análisis de la arquitectura revela los siguientes subsistemas:

- Subsistema de entrada: tiene la responsabilidad de recoger los eventos del teclado y del ratón en cada frame.
- Subsistema de audio: genera la música o efectos de sonido.
- Subsistema de vídeo: es responsable de visualizar cada fotograma en la pantalla.



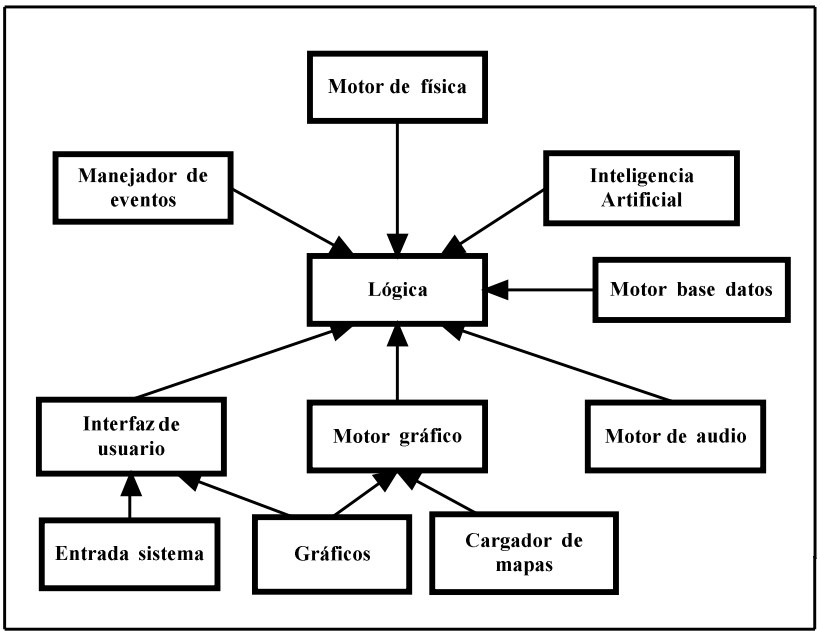
Quake 2 Client Architecture

Tras la comparación, se puede observar que el Quake II se ajusta a la arquitectura el modelo cliente-servidor. Algunas de las diferencias notables es la inclusión de un subsistema común que incluye todas las funciones y rutinas utilizadas tanto por el Cliente como el Servidor. La inclusión de subsistemas en el Juego es el punto de modularidad, que permite al Quake II ser cada vez más atractivo y personalizable para otros desarrolladores de software. Otra característica muy importante es la vinculación de todos los elementos específicos del juego en librerías dinámicas DLL.

Propuesta de arquitectura

Mediante el análisis de arquitecturas concretas para juegos FPS como, el Quake II, es bastante inherente que existan elementos comunes en el diseño de cada juego. A pesar de la mentalidad de que cada género requiere su propio diseño específico, se puede extraer una arquitectura general, como se muestra en la figura de abajo. Cada subsistema se describe a continuación:

- Manejador de eventos: es responsable de la gestión de eventos que desencadenan en un nuevo estado (la entrada del jugador, objetivos del juego, etc…)
- Motor de física: dicta la simulación de la física que rige cada jugador dentro del juego
- Motor de lógica: contiene todos los dominios específicos de información del juego, tales como gráficos, niveles, mapas, etc... También es responsable de la inteligencia artificial de cada personaje dentro del juego
- Motor de entrada: es responsable de mantener un registro de entrada del usuario y enviarlos al manejador de evento
- Motor de gráfico: renderiza toda la información a la pantalla utilizando las capacidades disponibles del hardware
- Cargador de mapas
- Motor de audio: es responsable de la reproducción de audio a través del hardware.
- Motor de base de datos: contiene información acerca de las entidades.



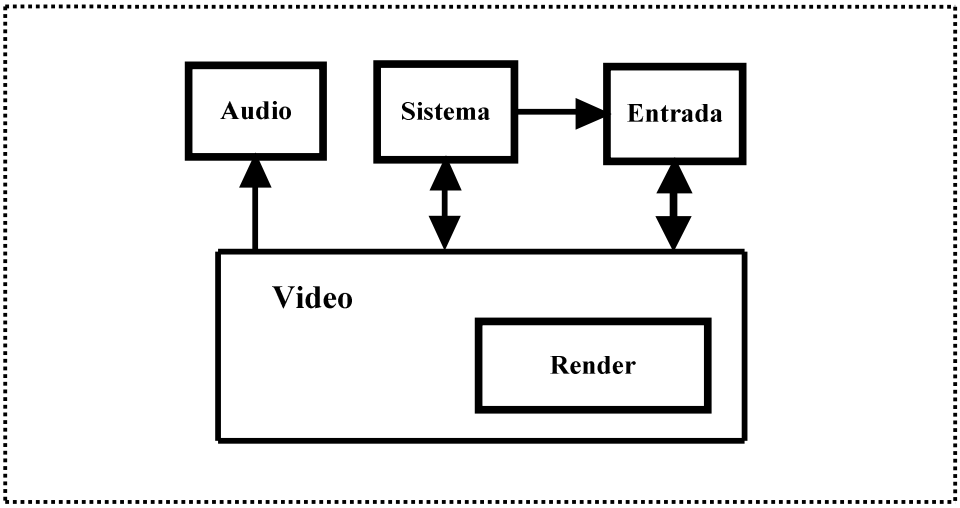
Propuesta de arquitectura

El análisis de todos los componentes de un videojuego demuestra que podemos organizar todos los subsistemas en dos dominios consistentes para el código específico del juego, y para el código no específico del juego. Es importante tener en cuenta la abstracción entre componentes específicos y los componentes que no son del juego para garantizar la máxima reutilización.

Módulo 1

Este primer módulo capta todos los componentes no específicos del juego y se vincula a una o varias DLL. Estos componentes suelen depender del hardware, por lo que la abstracción en el módulo es un paso lógico. Cada subsistema se describe a continuación.

- Subsistema de audio: responsable de la salida de audio a través de los equipos informáticos.
- Subsistema de entrada: mantiene un registro de todos los eventos recibidos por los usuarios y se encarga de enviarlos al manejador de eventos.
- Subsistema de vídeo: muestra gráficos a los usuarios, y abstrae el hardware subyacente para facilitar su uso a cualquier tipo de usuario.



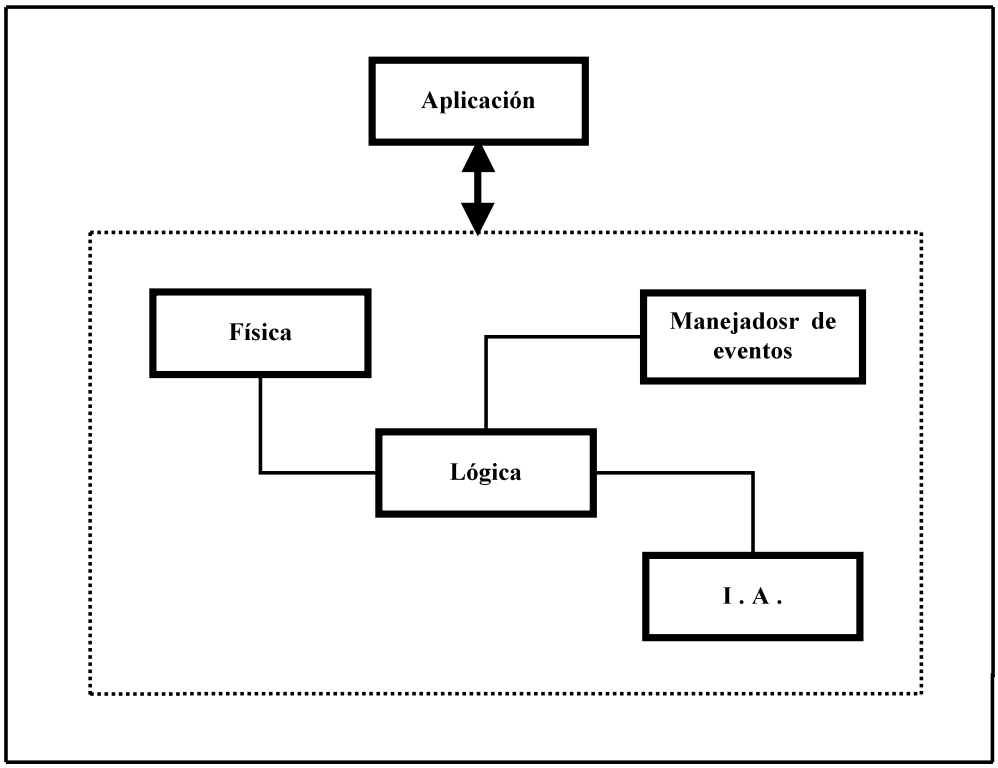
Arquitectura del módulo 1

La responsabilidad de los sistemas hardware específicos, permite una mayor reutilización. Optimizaciones específicas del hardware se ejecutan siempre en este módulo para aumentar el rendimiento.

Módulo 2

Este segundo módulo contiene todos los componentes específicos del videojuego. A diferencia del módulo 1, esta parte se integra dentro de un ejecutable <exe>. Cada subsistema se describe a continuación.

- Subsistema Aplicación: cabe destacar el Menú y el Juego. Es específico y responsable de que la Interfaz de usuario (UI) sea accesible por los jugadores.
- Subsistema Manejador de eventos: responsable de analizar el evento y determinar la acción a efectuar
- Subsistema Lógica: actualiza el estado actual del juego y contiene información específica, personajes, armas, mapas, gráficos, etc…
- Subsistema Inteligencia Artificial: definen el comportamiento de cada personaje en el juego, este subsistema suele incluirse dentro del subsistema de la lógica.
- Subsistema Física: regula la forma en que cada jugador interactúa dentro del mundo.



Arquitectura del módulo 2

Los juegos basados en un determinado motor por lo general tienen la misma apariencia. La figura “Propuesta de arquitectura” muestra los componentes de un Motor de juego.

Modelo de Control

El modelo de control empleado en el desarrollo de ésta aplicación, es un modelo de control **centralizado**. Se designa a un controlador del sistema, que es la aplicación del juego. Tiene la responsabilidad de administrar la ejecución del resto de los subsistemas.

El control es compartido por los subsistemas, pero se espera que se devuelva la responsabilidad de control a la aplicación principal. Se entiende que el modelo de control centralizado usado es el de llamada-retorno. El control se inicia en la parte superior de una jerarquía y, por medio de llamadas a subrutinas, pasa a los niveles inferiores.

Descomposición modular

Después de diseñar la arquitectura estructural, la siguiente etapa del proceso de diseño arquitectónico es descomponer el sistema en módulos.

Para el sistema que se está implementando, se usará un **modelo orientado a objetos**, en la que el sistema se descompone en un conjunto de objetos que se comunican entre ellos. Los módulos son objetos con estado privado y operaciones definidas sobre ese estado.

En el siguiente apartado se identificarán los diferentes módulos o componentes existentes en el sistema. Se partirá de los identificados en la fase de análisis y se irán completando a medida que se desarrolle el diseño del videojuego. Cada uno de los componentes mostrados contendrá un conjunto de objetos organizados según el modelo orientado de objetos comentado anteriormente.

Diseño de componentes

Un componente es una colección de clases, operaciones globales, y otros elementos que comparten el mismo conjunto de responsabilidades. También puede referirse a una sola clase. En el desarrollo de un determinado componente se incorporan un conjunto de funcionalidades, junto con una descripción del punto en el que se encuentra del proceso de desarrollo. Los componentes están representados utilizando hasta cinco puntos de vista del sistema. Cada componente debe ser completo de tal manera que se pueda identificar una funcionalidad específica. Esto nos ahorrará muchos esfuerzos en procesos de mantenimiento y actualización.

El sistema se presenta en cinco "vistas" diferentes. Un punto de vista es una manera de ver el sistema. En algunos casos, las "vistas" son creadas usando diagramas UML. En otros casos, son usados las listas u objetos genéricos de UML. A continuación se muestran las vistas:

- Requisitos (lista y CRC).
- Conceptual (diagrama de objetos). Esta es una representación del sistema usando o bien un caso de uso general o un objeto gráfico para describir el sistema. Si se usa el diagrama de objetos, las cajas representan a la clase de objetos, funciones globales, objetos COM, tablas de base de datos, y otros elementos. Una vista conceptual es opcional. Se recomienda su uso durante las primeras etapas del diseño, o en los puntos del proyecto que se necesite mostrar algunos de los trabajos preliminares que llevan a decisiones de diseño.
- Comportamiento (secuencia y colaboración). Una vista del comportamiento muestra cómo los objetos que integran el sistema interactúan a través de mensajes o de las operaciones. Las vistas del comportamiento puede presentarse utilizando diagramas UML de secuencia y de colaboración.
- Lógico (clase). Una representación del sistema en la que se muestra las clases y sus relaciones estáticas. La vista de la lógica del sistema se representa mediante los diagramas de clases. En los diagramas de clase se incluyen los atributos y operaciones de las clases. Los casos de uso se crean tanto inductiva como deductivamente. Inductivamente, los casos de uso se derivan de los diagramas de secuencia y de colaboración. Deductivamente, son tomadas directamente del análisis de los requisitos funcionales del sistema.
- Componente (interno y externo).
  - La visión externa representa a la totalidad del sistema construido en un punto dado del desarrollo. En última instancia, representa la forma en que el sistema está desplegado. Desde que se tiene la intención de representar a un sistema global durante el esfuerzo de desarrollo de software, la vista del componente externo se implica más.
  - El componente interno tiene la intención de mostrar una vista aislada del sistema. Se puede o no representar a todo un sistema, pero todas las dependencias del sistema relevantes al componente deben ser incluidas. Se tiene la intención de representar a una parte del sistema general exponiendo

las características específicas del sistema de un determinado componente y no la representación del sistema en su conjunto.

Descripción de componentes

Esta sección contiene las descripciones y los esquemas de cada uno de los componentes de software del sistema. En general, es una representación del sistema que abarca tanto la funcionalidad del sistema y el lugar en el proceso de desarrollo en la que la funcionalidad se implementa.

Si está efectuando una actualización del sistema, la representación por componentes del sistema le permitirá entender el orden en que se desarrollaron los componentes y el tipo de dependencias que se existen durante el proceso de desarrollo.

Lista de Componentes

Los componentes que se presentan a continuación, están expuestos en el orden en el que el sistema se puede construir. Los componentes del juego son:

- Componente 1: Apertura del juego.
- Componente 2: Gráficos.
  - Componente 2.1: Objetos GUI en 2D.
  - Componente 2.2: Constructor de niveles.
  - Componente 2.2: Constructor de entidades.
  - Componente 2.4: Efectos y animaciones.
- Componente 3: Sonido.
- Componente 4: Configuración del jugador.
- Componente 5: Opciones del juego.
- Componente 6: Navegación en el nivel.
- Componente 7: Unidades físicas.
- Componente 8: Ítems.
- Componente 9: Combates.
- Componente 10: Inteligencia Artificial.
- Componente 11: Vista de las estadísticas.
- Requisitos no funcionales.

Componente 1 – Apertura del juego

Este componente es la primera implementación del juego. Proporciona un marco básico en el que pone a prueba la ventana de juego y la interfaz gráfica con la que el usuario interactuará.

Componente 1 - Vista de requisitos

Esta sección identifica los requisitos y los casos de uso derivados del documento de requisitos del Software.

Componente 1 - Requisitos aplicables

La siguiente lista identifica los requisitos abordados:
Requisitos: 1, 2, 3, 4, 6, 7, 10

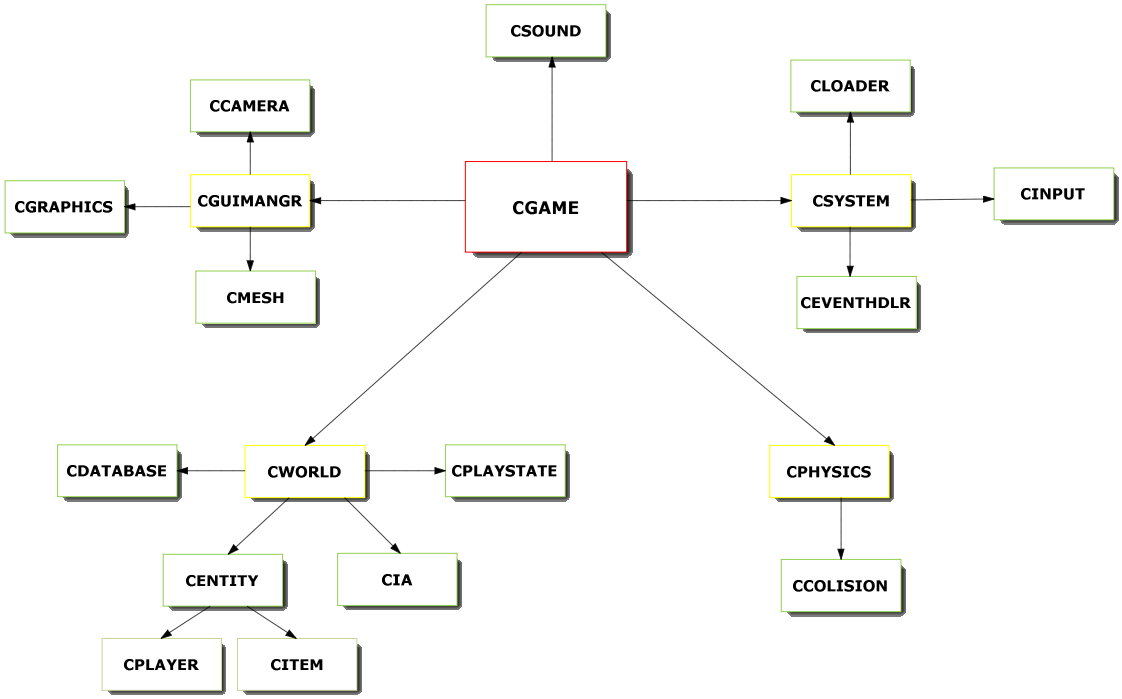
Componente 1 - Vista de casos de uso

Los siguientes casos de usos proporcionan una narración de una secuencia de acciones que pueden ser utilizados para situar el comportamiento del componente en su contexto.
Casos de uso: 1, 2, 3, 4, 5, 6, 7

Componente 1 - Vista conceptual generalizada

Esta sección proporciona una vista conceptual (opcional) generalizada. El siguiente diagrama de contexto es muy útil para las primeras de iteraciones del ciclo de desarrollo, como un medio para que los miembros del equipo puedan comprender las principales divisiones o componentes del sistema.

El diagrama se basa en las interacciones derivadas de la utilización de los casos de uso, los requisitos, y una primera conceptualización del sistema. Este punto de vista se utiliza sólo en este componente, con fines de análisis general. En posteriores, el sistema adquiere una gravedad que hace que esta vista suplemental sea innecesaria.



Componente 1 - Vista del comportamiento – Interacciones

Esta sección describe las interacciones en el comportamiento del componente.

Componente 1 - Vista de la secuencia

La secuencia muestra la vista de las interacciones básicas, junto con la identidad genérica de las clases que interactúan.

Usuario Diagrama Secuencia

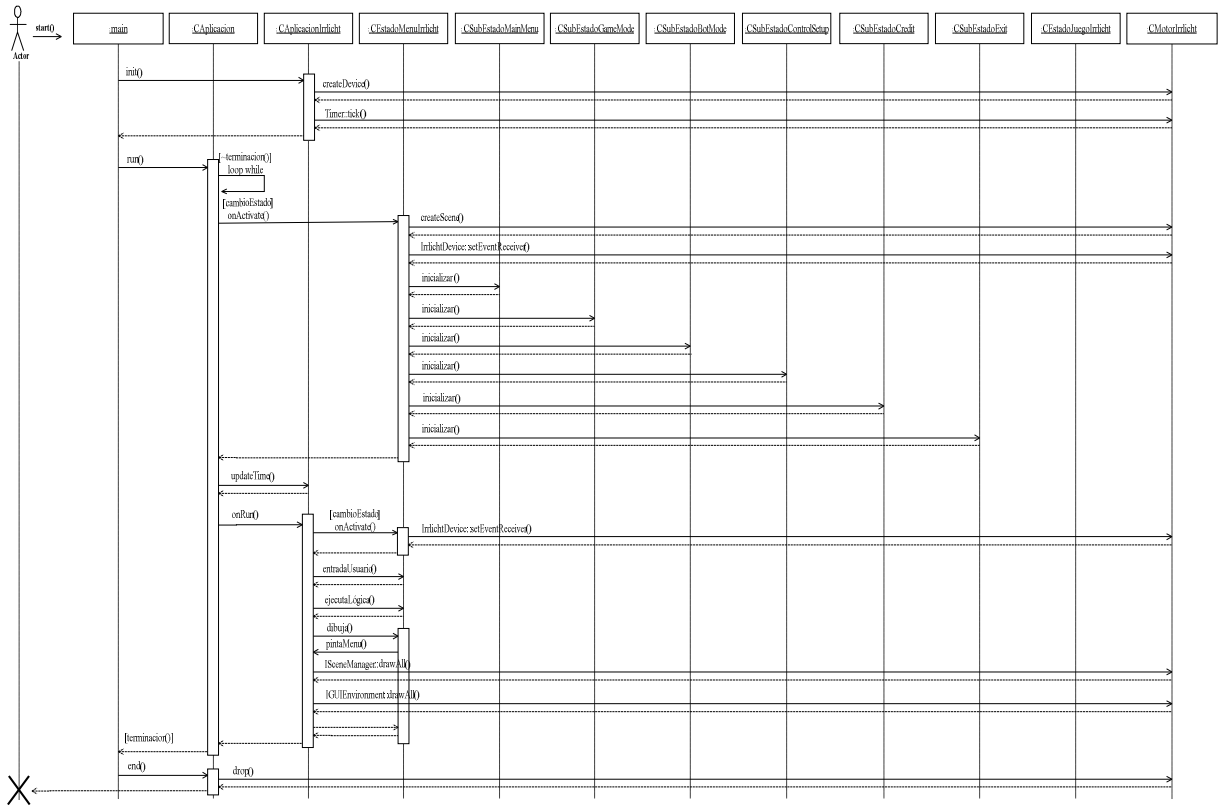
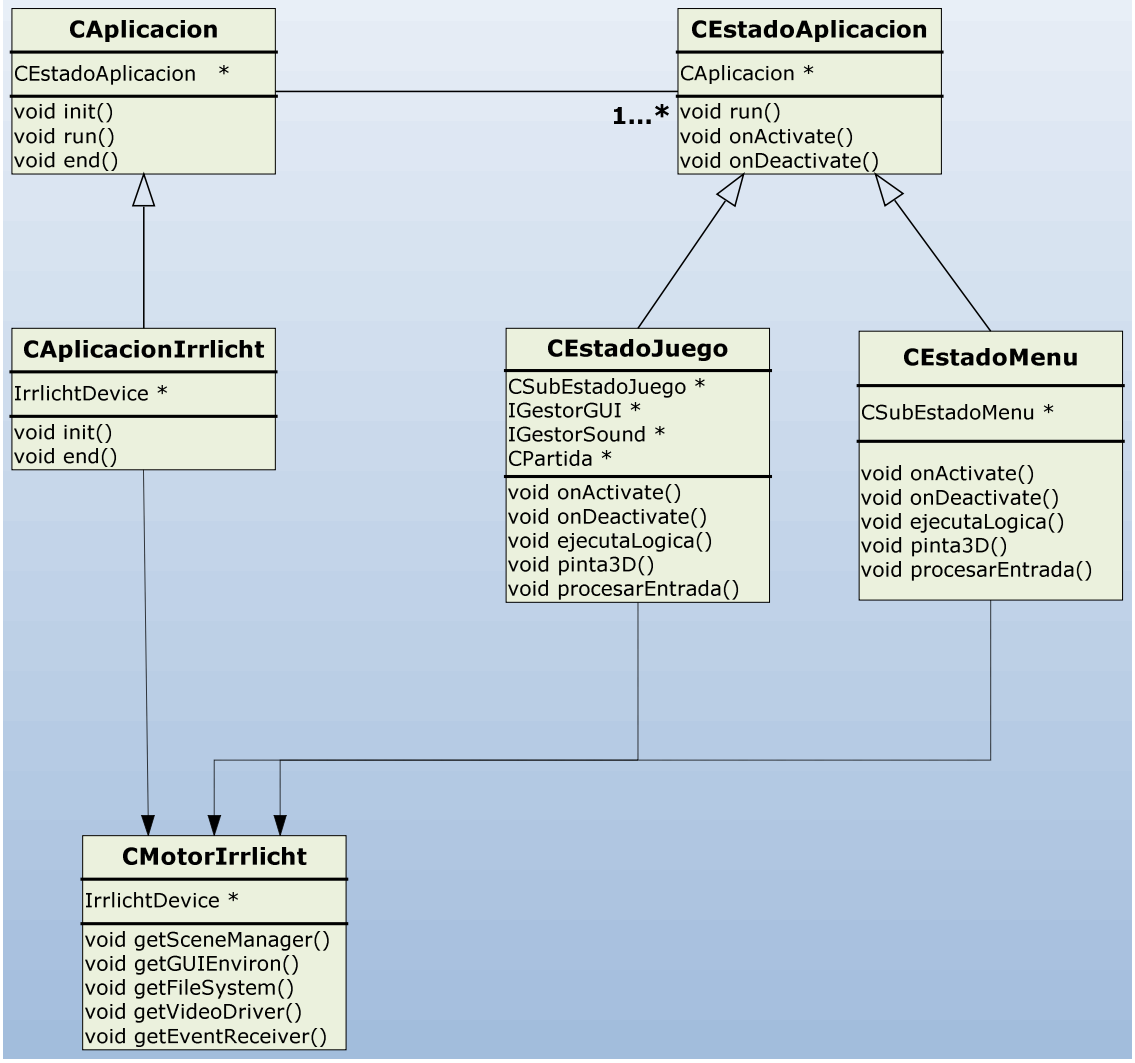


Figura 1.

Componente 1 - Vista de la lógica – Diagrama de clase

Esta sección describe las clases que se utilizan. Las operaciones definidas en las clases reflejan el comportamiento que se produce en este y otros componentes, pero aún no podrá representar la definición completa de la clase. Hay que tener en cuenta que en los siguientes componentes, no utilizamos el listado de operaciones. Esto es sólo por conveniencia, las operaciones deberían listarse cuando sea necesario.



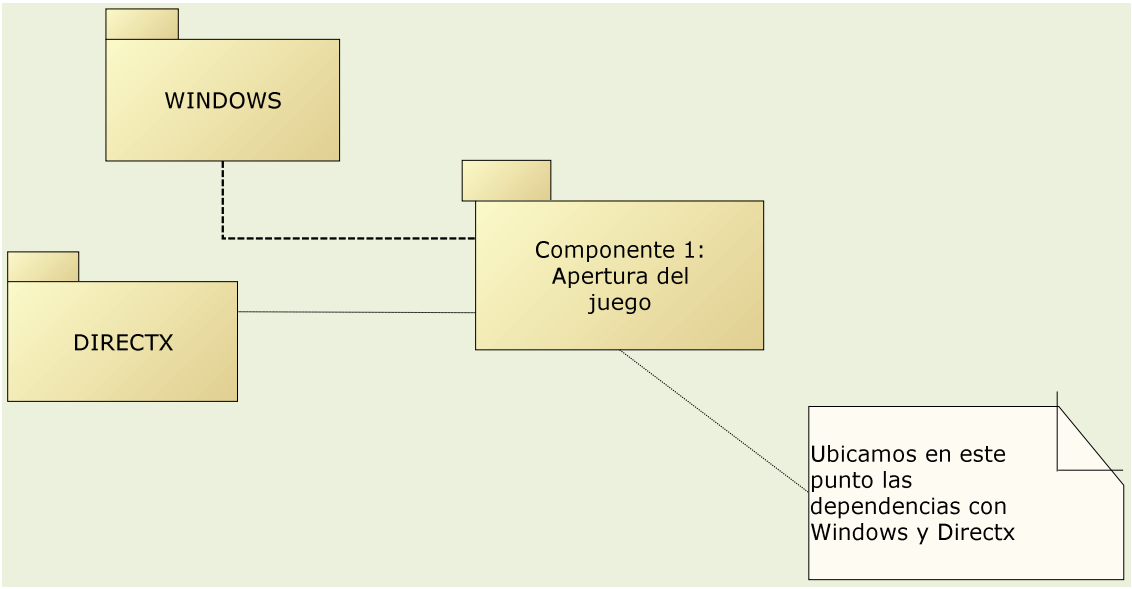
Componente 1 - Vista de componentes

Esta sección representa la vista de componentes.

Se representará un diagrama de componentes de alto nivel en el que se muestra junto con todos los demás componentes activos en el sistema. El propósito de la vistas de componentes es mostrar las dependencias entre componentes.

Componente 1 - Vista del sistema

Esta sección proporciona una vista de los componentes activos del sistema. En este momento no existen dependencias con otros componentes.



Componente 2 – Gráficos

- Componente 2.1 – Objetos GUI en 2D
- Componente 2.2 – Constructor de niveles
- Componente 2.3 – Constructor de entidades
- Componente 2.4 – Efectos y animaciones

Componente 2.1 – Objetos GUI en 2D

Componente 2.1 - Vista de requisitos

Esta sección identifica los requisitos y los casos de uso derivados del documento de requisitos del Software.

Componente 2.1 - Requisitos aplicables

La siguiente lista identifica los requisitos abordados:
Requisitos: 2, 11, 16

Componente 2.1 - Vista de casos de uso

Los siguientes casos de usos proporcionan una narración de una secuencia de acciones que pueden ser utilizados para situar el comportamiento del componente en su contexto.
Casos de uso: 3

Componente 2.1 - Vista del comportamiento – Interacciones

Esta sección describe las interacciones en el comportamiento del componente.

Componente 2.1 - Vista de la secuencia

La secuencia muestra la vista de las interacciones básicas, junto con la identidad genérica de las clases que interactúan.

USUARIO Diagrama Secuencia

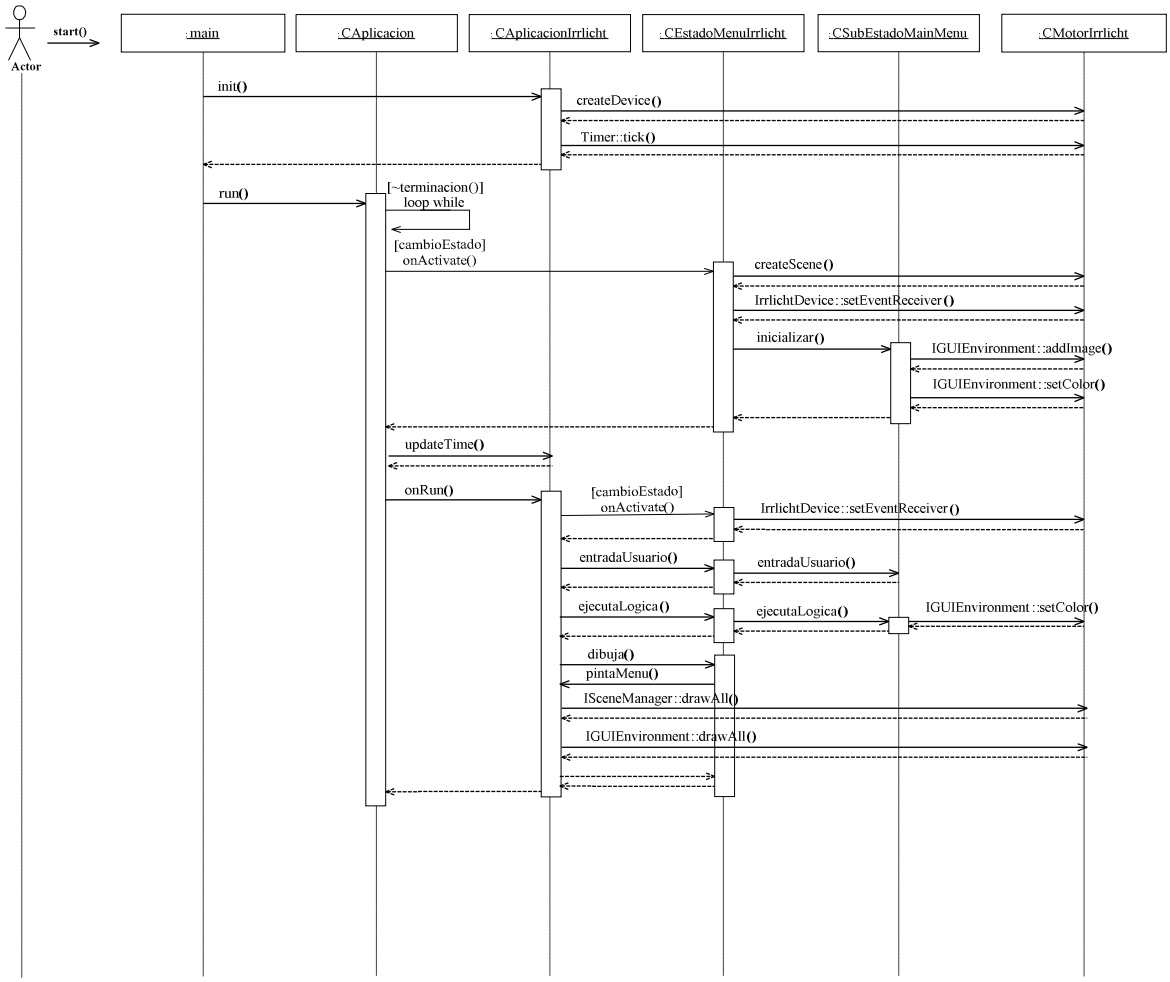
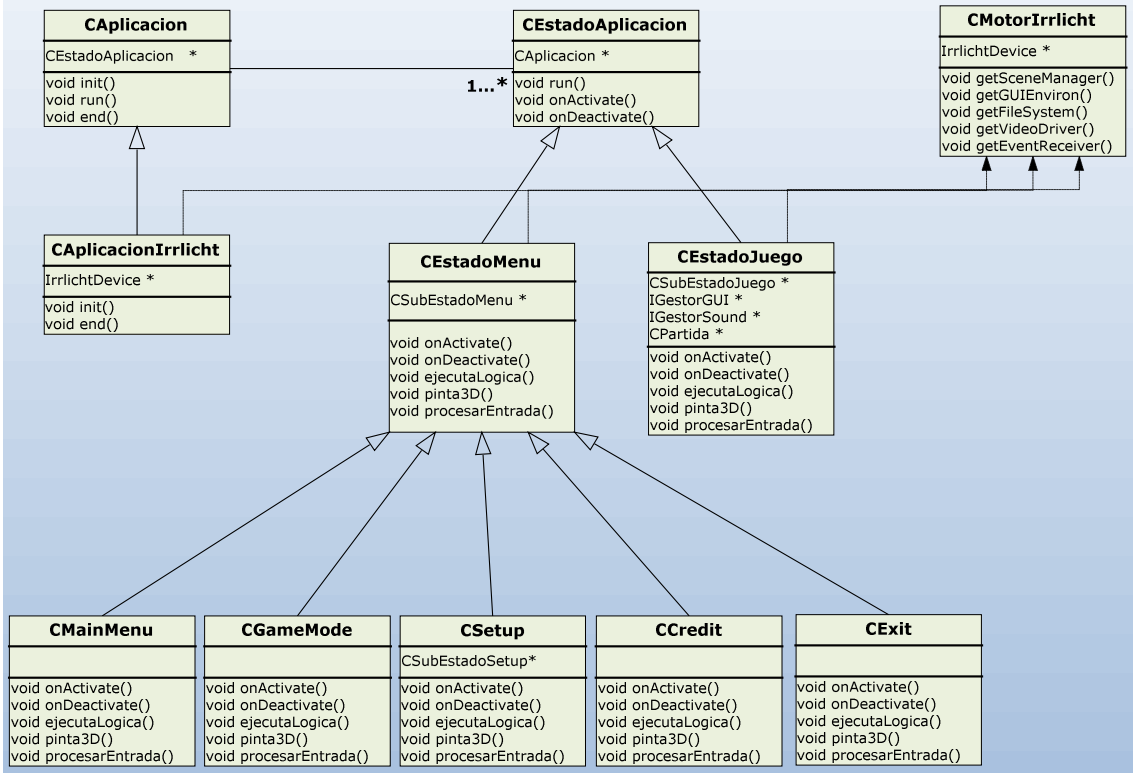


Figura 2.

Componente 2.1 - Vista de la lógica – Diagrama de clase

Esta sección describe las clases que se utilizan. Las operaciones definidas en las clases reflejan el comportamiento que se produce en este y otros componentes, pero aún no podrá representar la definición completa de la clase.



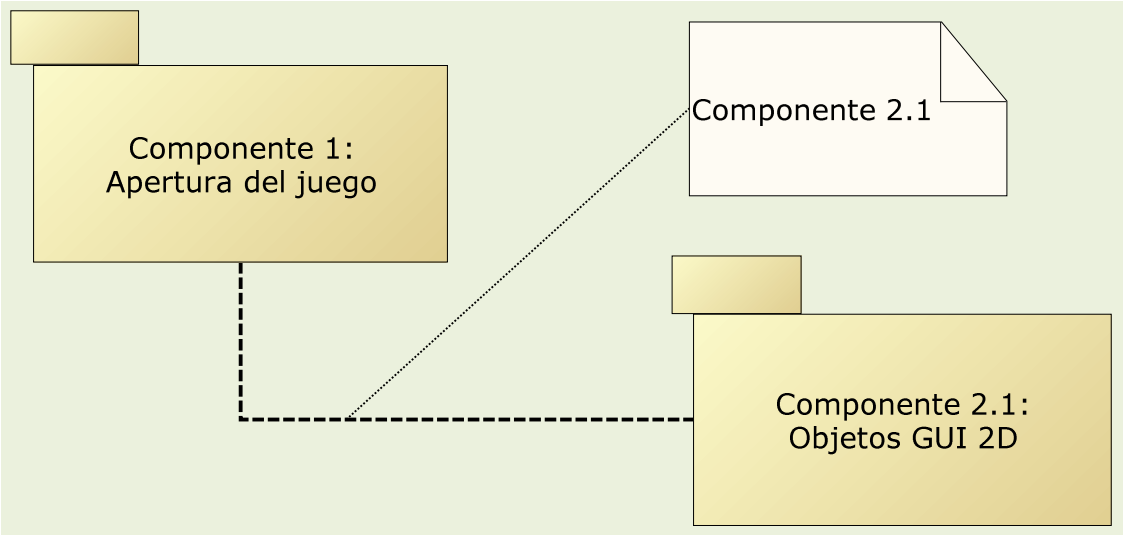
Componente 2.1 - Vista de componentes

Esta sección representa la vista de componentes.

Se representará un diagrama de componentes de alto nivel en el que se muestra junto con todos los demás componentes activos de la actualidad en el sistema. El propósito de la vistas de componentes es mostrar las dependencias entre componentes.

Componente 2.1 - Vista del sistema

Esta sección proporciona una vista de los componentes activos del sistema.



Componente 2.2 – Constructor de niveles

Componente 2.2 - Vista de requisitos

Esta sección identifica los requisitos y los casos de uso derivados del documento de requisitos del Software.

Componente 2.2 - Requisitos aplicables

La siguiente lista identifica los requisitos abordados:
Requisitos: 6, 9, 13, 15

Componente 2.2 - Vista de casos de uso

Los siguientes casos de usos proporcionan una narración de una secuencia de acciones que pueden ser utilizados para situar el comportamiento del componente en su contexto.
Casos de uso: 8

Componente 2.2 - Vista del comportamiento – Interacciones

Esta sección describe las interacciones en el comportamiento del componente.

Componente 2.2 - Vista de la secuencia

La secuencia muestra la vista de las interacciones básicas, junto con la identidad genérica de las clases que interactúan.

USUARIO

Diagrama Secuencia

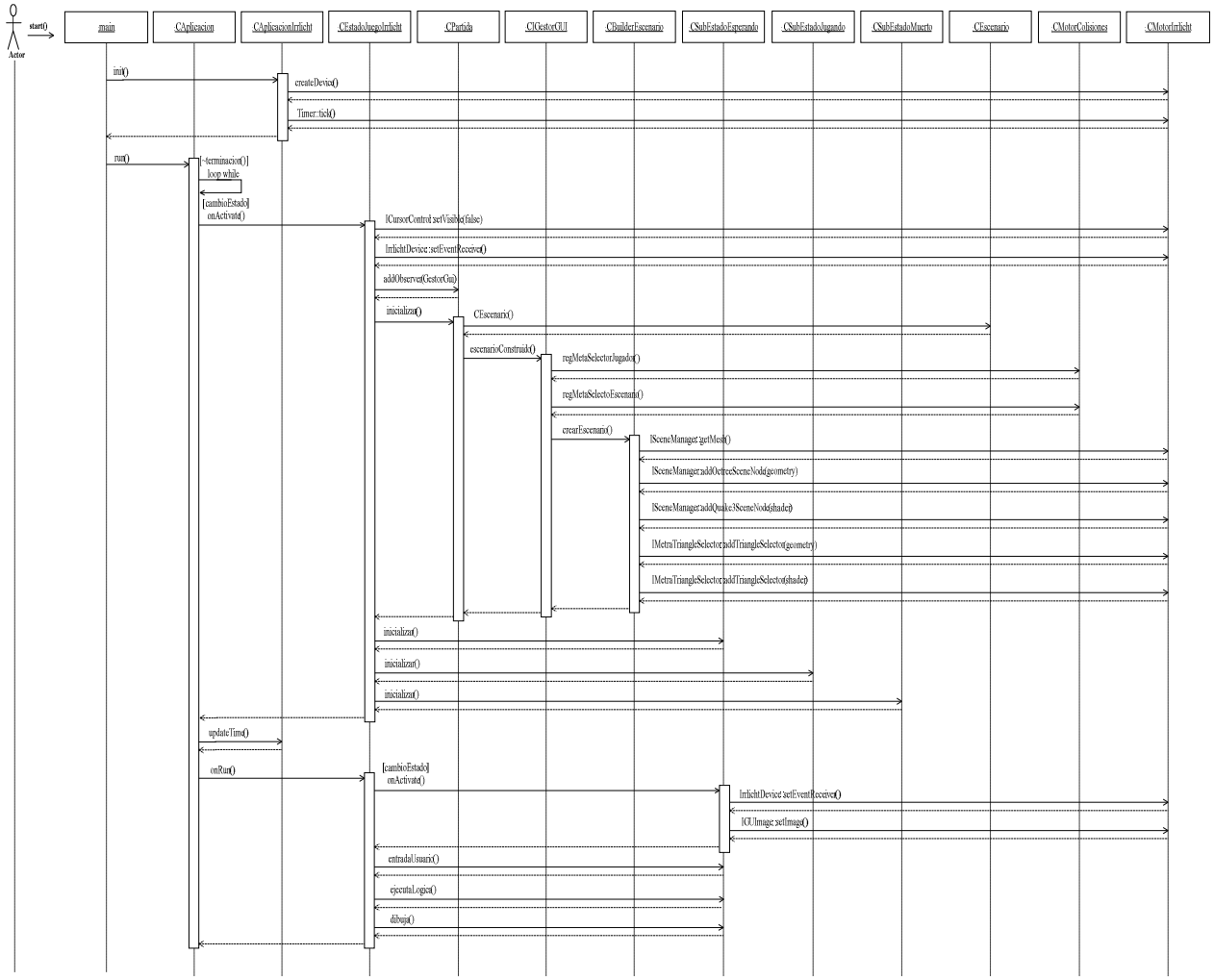
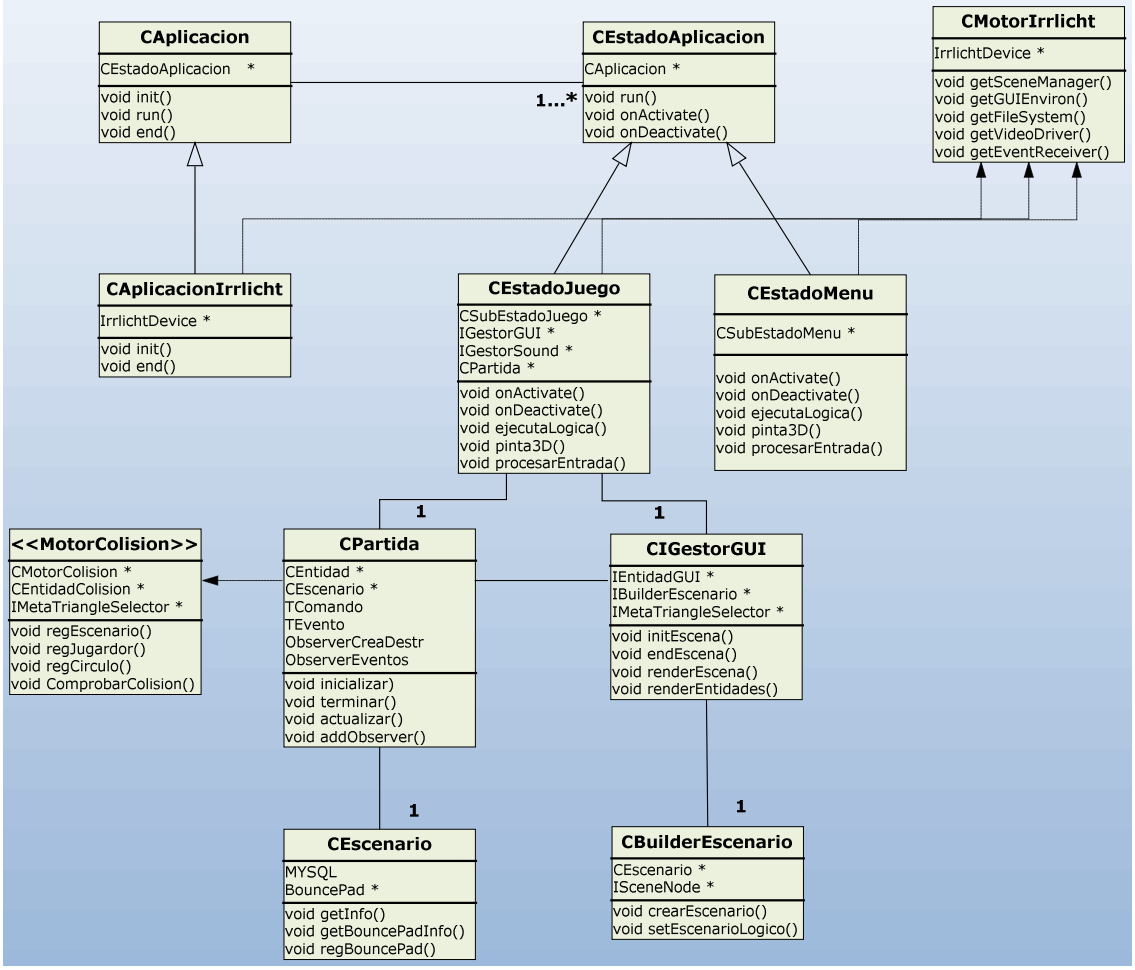


Figura 3.

Componente 2.2 - Vista de la lógica – Diagrama de clase

Esta sección describe las clases que se utilizan. Las operaciones definidas en las clases reflejan el comportamiento que se produce en este y otros componentes, pero aún no podrá representar la definición completa de la clase.



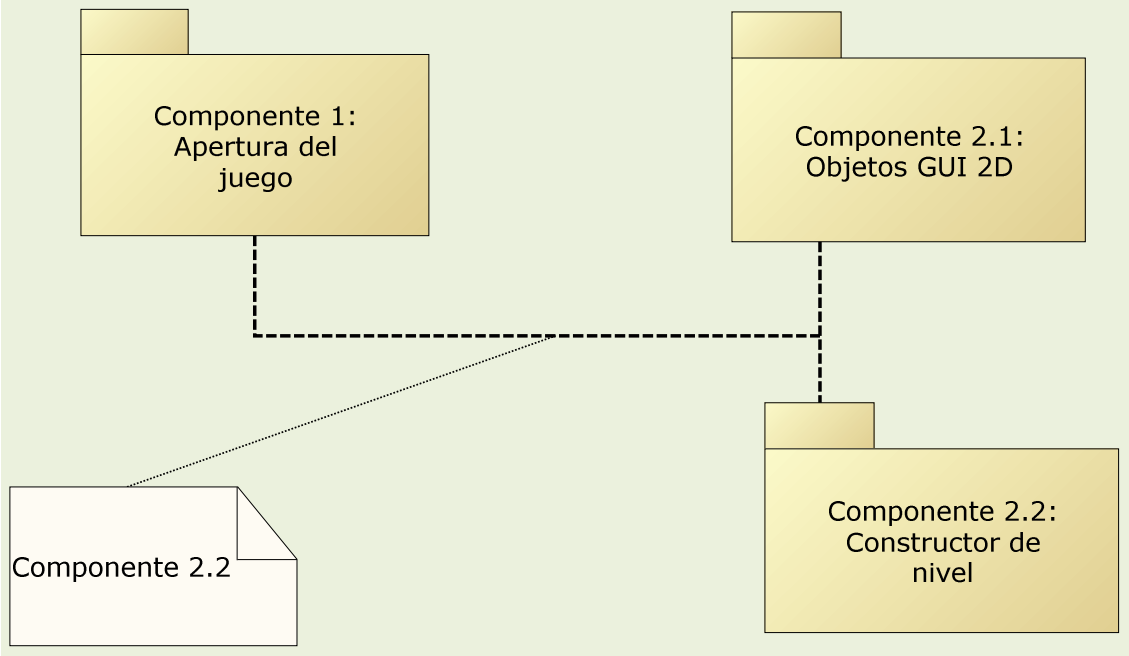
Componente 2.2 - Vista de componentes

Esta sección representa la vista de componentes.

Se representará un diagrama de componentes de alto nivel en el que se muestra junto con todos los demás componentes activos de la actualidad en el sistema. El propósito de la vistas de componentes es mostrar las dependencias entre componentes.

Componente 2.2 - Vista del sistema

Esta sección proporciona una vista de los componentes activos del sistema.



Componente 2.3 – Constructor de entidades

Componente 2.3 - Vista de requisitos

Esta sección identifica los requisitos y los casos de uso derivados del documento de requisitos del Software.

Componente 2.3 - Requisitos aplicables

La siguiente lista identifica los requisitos abordados:
Requisitos: 14, 15

Componente 2.3 - Vista de casos de uso

Los siguientes casos de usos proporcionan una narración de una secuencia de acciones que pueden ser utilizados para situar el comportamiento del componente en su contexto.
Casos de uso: 9

Componente 2.3 - Vista del comportamiento – Interacciones

Esta sección describe las interacciones en el comportamiento del componente.

Componente 2.3 - Vista de la secuencia

La secuencia muestra la vista de las interacciones básicas, junto con la identidad genérica de las clases que interactúan.

USUARIO | Diagrama Secuencia

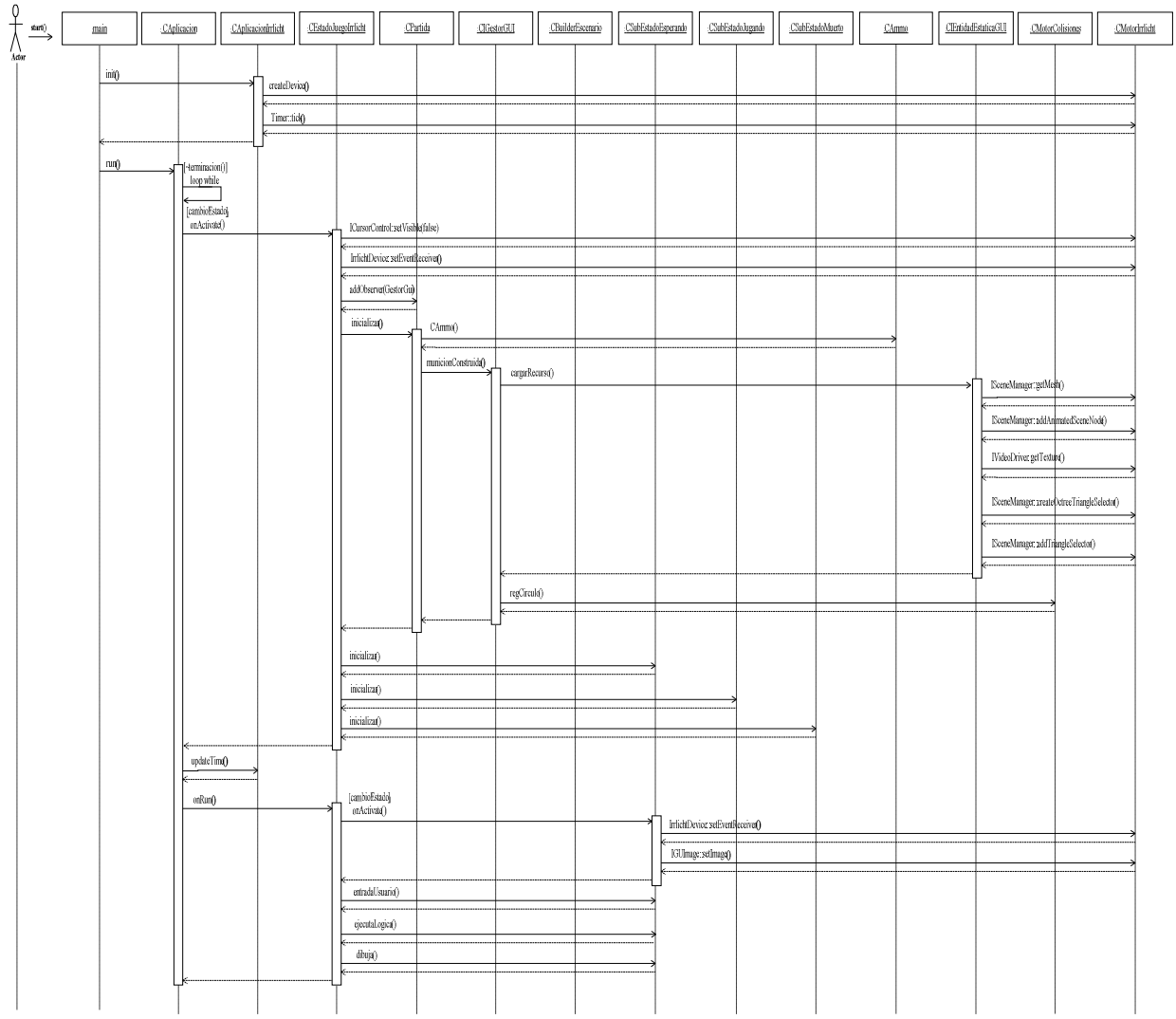
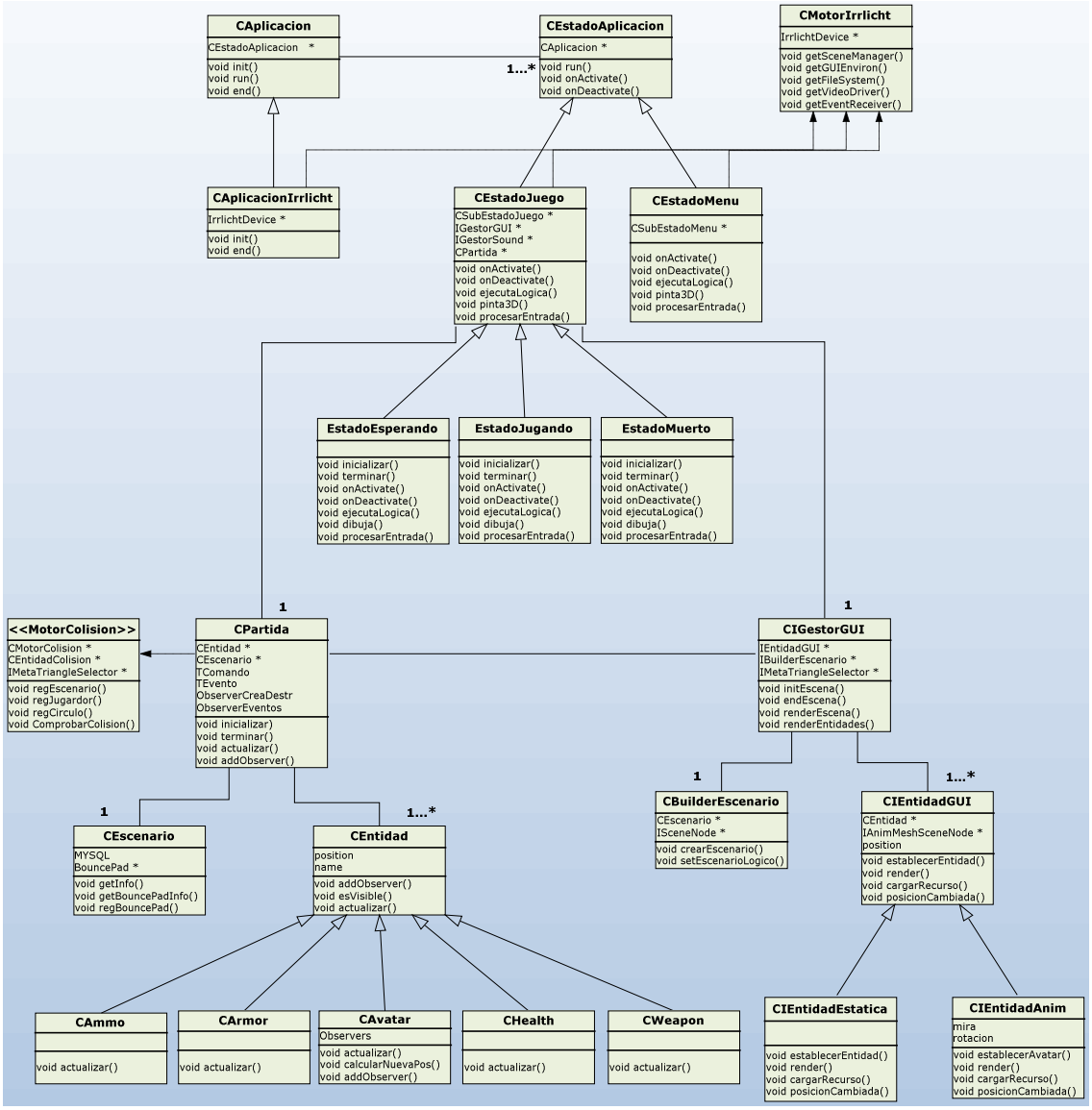


Figura 4.

Componente 2.3 - Vista de la lógica – Diagrama de clase

Esta sección describe las clases que se utilizan. Las operaciones definidas en las clases reflejan el comportamiento que se produce en este y otros componentes, pero aún no podrá representar la definición completa de la clase.



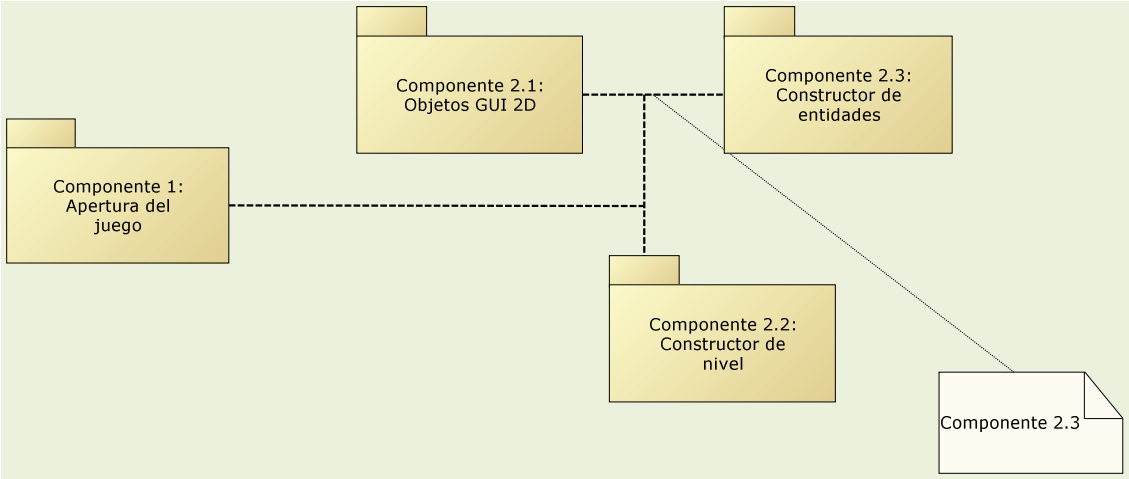
Componente 2.3 - Vista de componentes

Esta sección representa la vista de componentes.

Se representará un diagrama de componentes de alto nivel en el que se muestra junto con todos los demás componentes activos de la actualidad en el sistema. El propósito de la vistas de componentes es mostrar las dependencias entre componentes.

Componente 2.3 - Vista del sistema

Esta sección proporciona una vista de los componentes activos del sistema.



Componente 2.4 – Efectos y animaciones

Componente 2.4 - Vista de requisitos

Esta sección identifica los requisitos y los casos de uso derivados del documento de requisitos del Software.

Componente 2.4 - Requisitos aplicables

La siguiente lista identifica los requisitos abordados:
Requisitos: 17, 18, 48

Componente 2.4 - Vista de casos de uso

Los siguientes casos de usos proporcionan una narración de una secuencia de acciones que pueden ser utilizados para situar el comportamiento del componente en su contexto.
Casos de uso: 10

Componente 2.4 - Vista del comportamiento – Interacciones

Esta sección describe las interacciones en el comportamiento del componente.

Componente 2.4 - Vista de la secuencia

La secuencia muestra la vista de las interacciones básicas, junto con la identidad genérica de las clases que interactúan.

USUARIO

Diagrama Secuencia

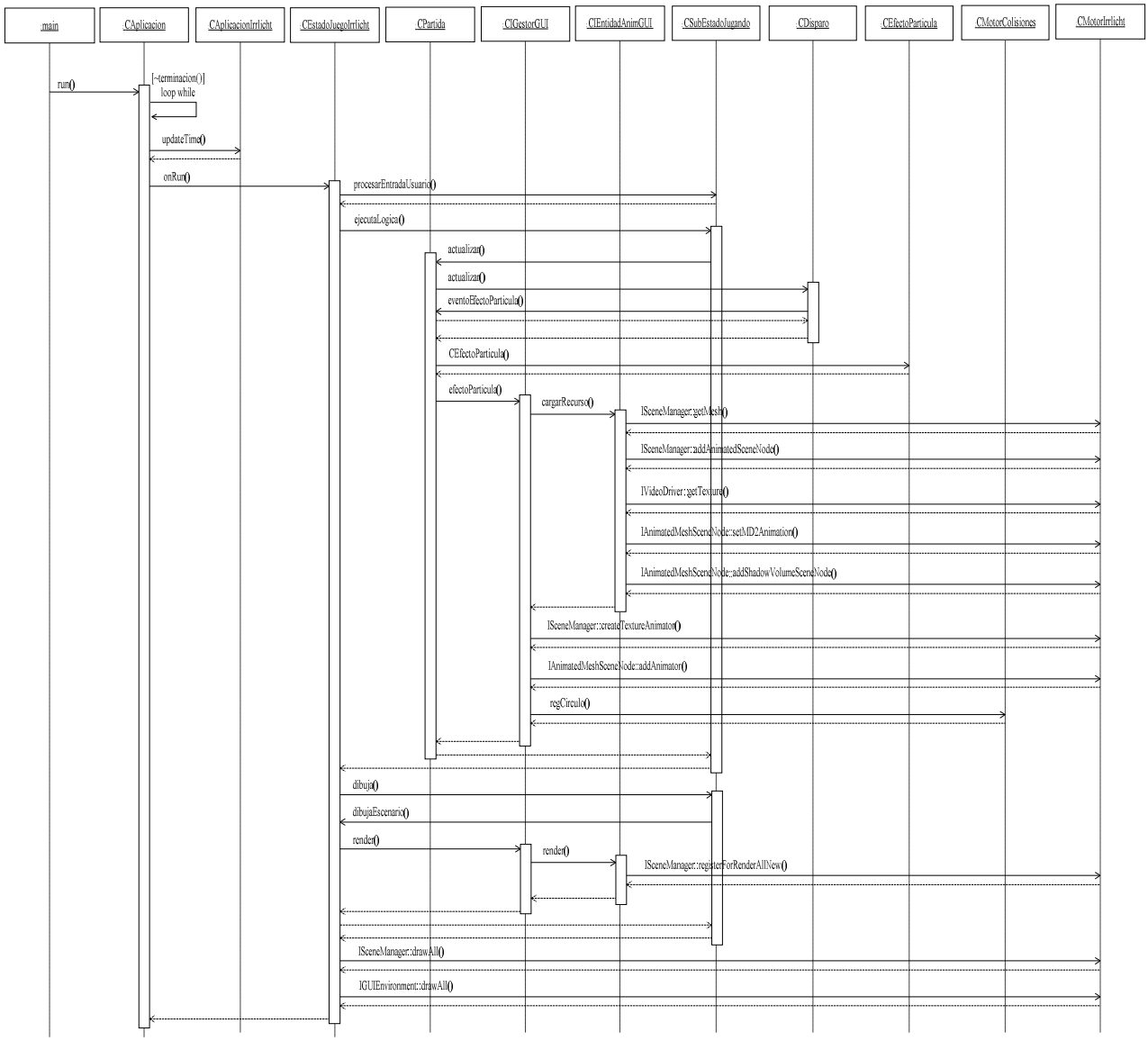
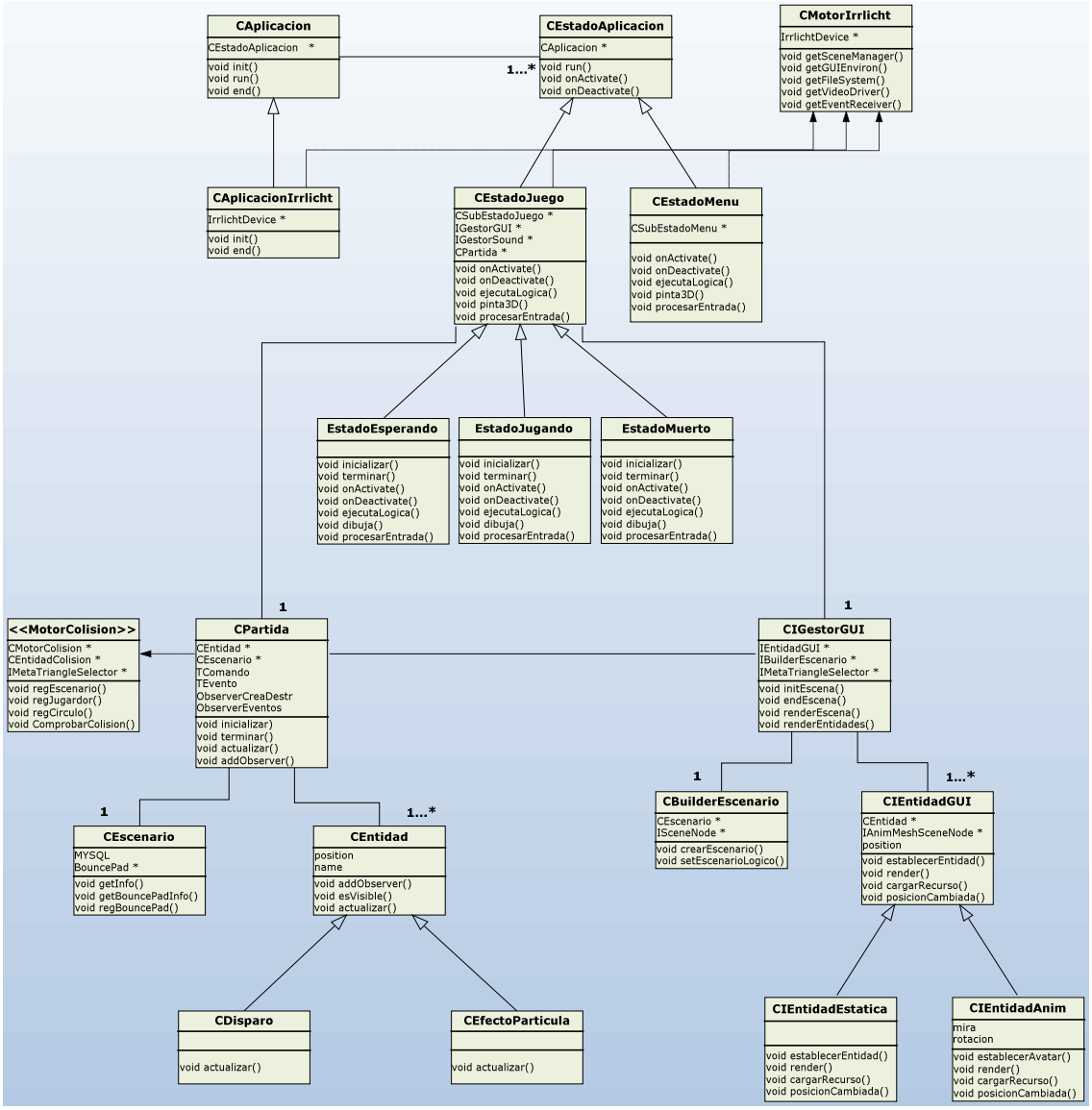


Figura 5.

Componente 2.4 - Vista de la lógica – Diagrama de clase

Esta sección describe las clases que se utilizan. Las operaciones definidas en las clases reflejan el comportamiento que se produce en este y otros componentes, pero aún no podrá representar la definición completa de la clase.



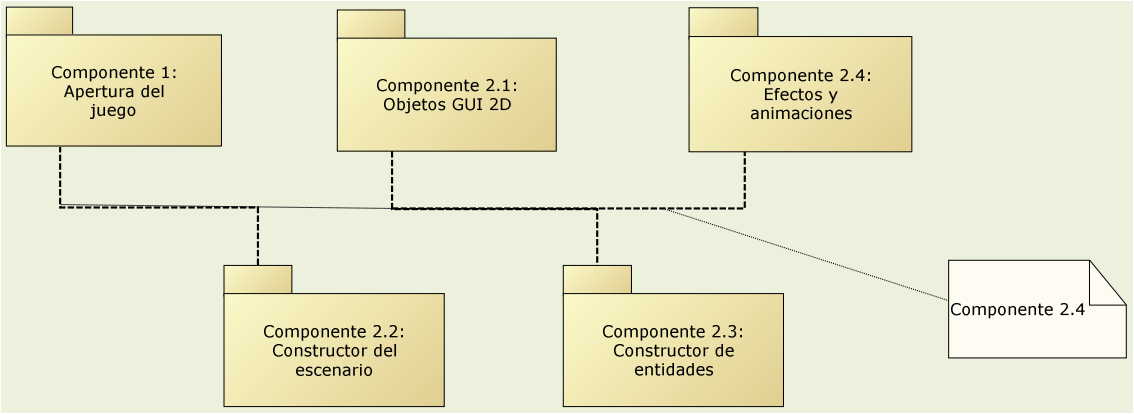
Componente 2.4 - Vista de componentes

Esta sección representa la vista de componentes.

Se representará un diagrama de componentes de alto nivel en el que se muestra junto con todos los demás componentes activos de la actualidad en el sistema. El propósito de la vistas de componentes es mostrar las dependencias entre componentes.

Componente 2.4 - Vista del sistema

Esta sección proporciona una vista de los componentes activos del sistema.



Componente 3 – Sonido

Componente 3 - Vista de requisitos

Esta sección identifica los requisitos y los casos de uso derivados del documento de requisitos del Software.

Componente 3 - Requisitos aplicables

La siguiente lista identifica los requisitos abordados:
Requisitos: 20, 21, 22, 23, 24, 25, 26

Componente 3 - Vista de casos de uso

Los siguientes casos de usos proporcionan una narración de una secuencia de acciones que pueden ser utilizados para situar el comportamiento del componente en su contexto.
Casos de uso: 11, 12

Componente 3 - Vista del comportamiento – Interacciones

Esta sección describe las interacciones en el comportamiento del componente.

Componente 3 - Vista de la secuencia

La secuencia muestra la vista de las interacciones básicas, junto con la identidad genérica de las clases que interactúan.

USUARIO

Diagrama Secuencia

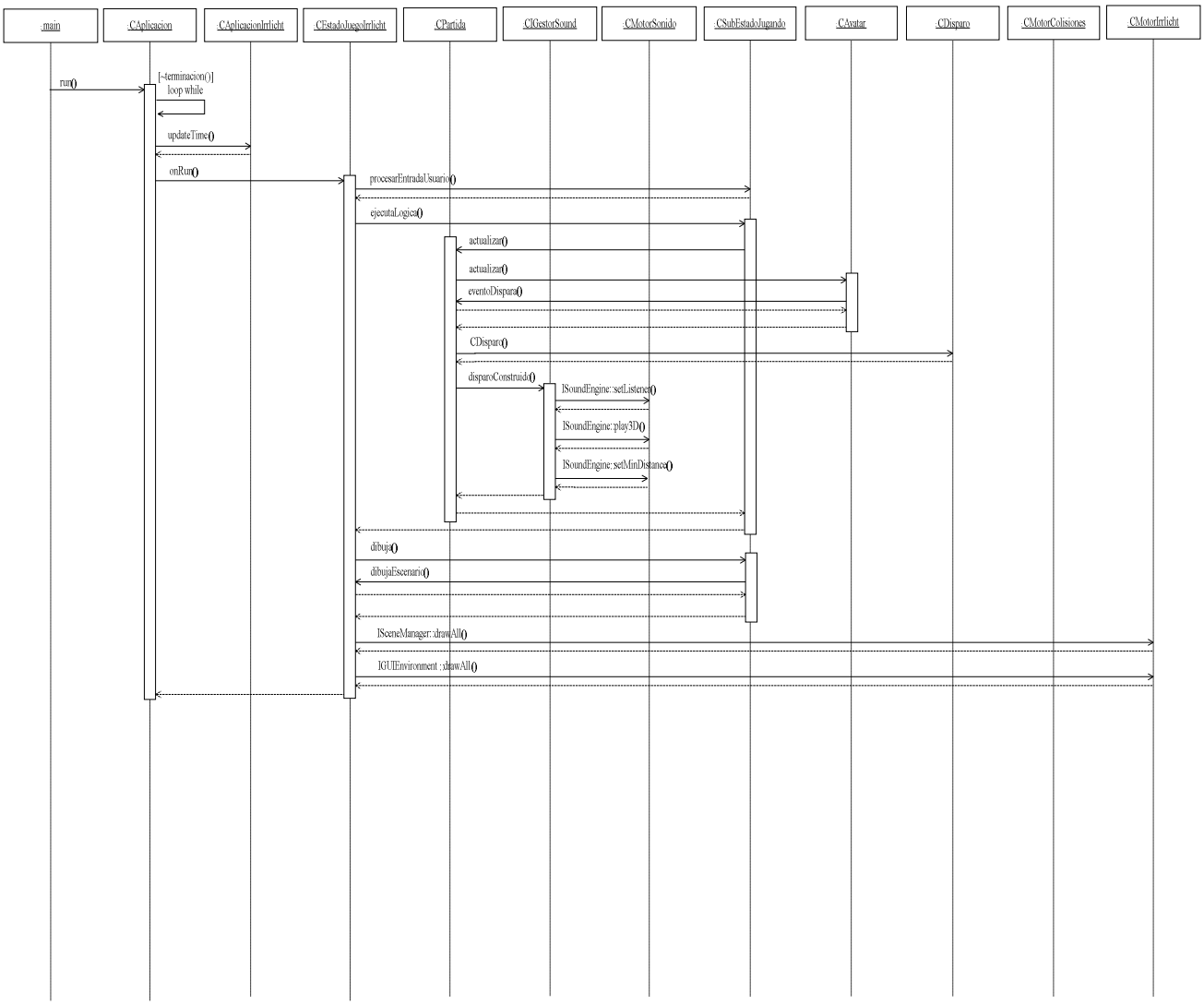
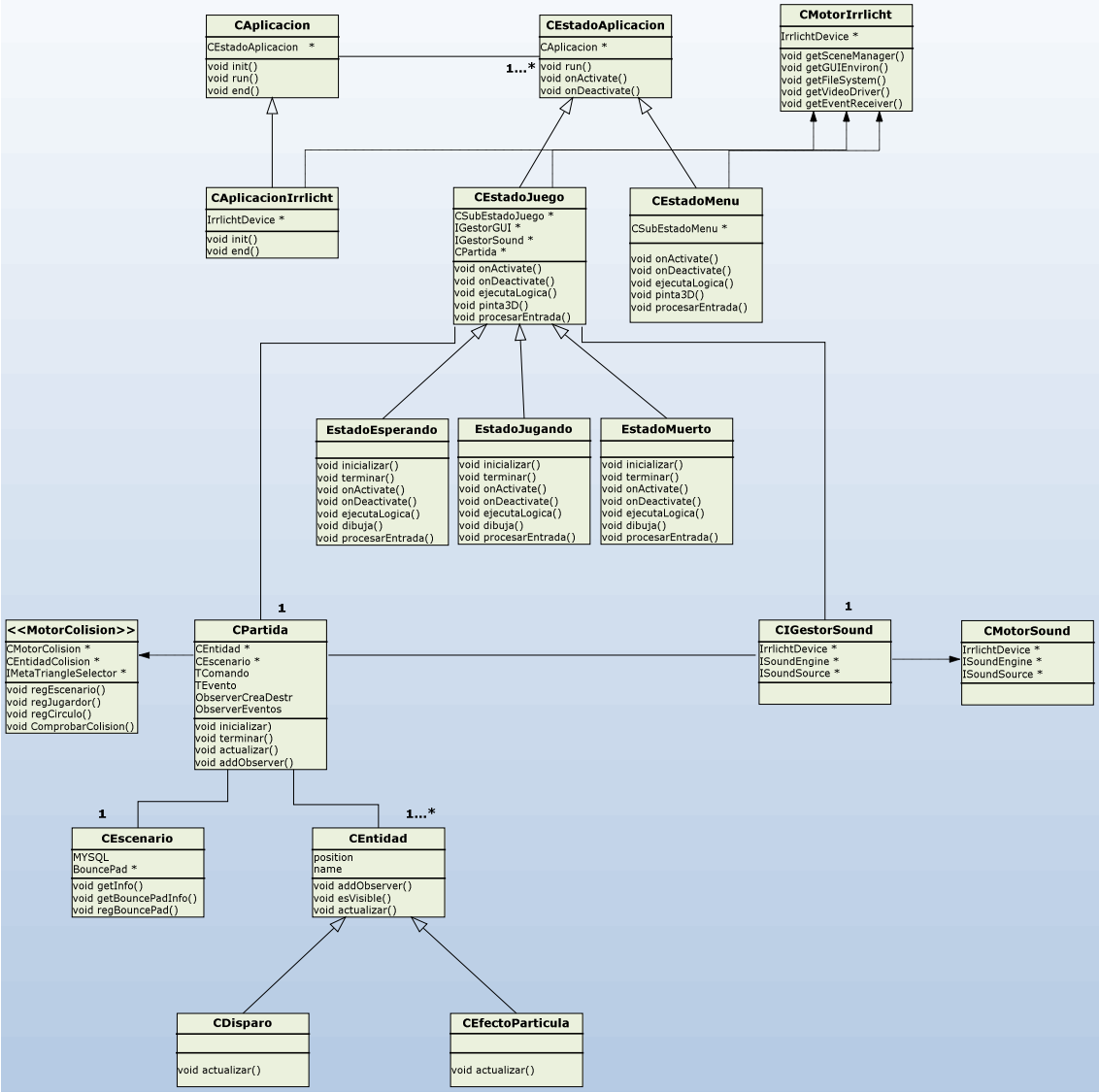


Figura 6.

Componente 3 - Vista de la lógica – Diagrama de clase

Esta sección describe las clases que se utilizan. Las operaciones definidas en las clases reflejan el comportamiento que se produce en este y otros componentes, pero aún no podrá representar la definición completa de la clase.



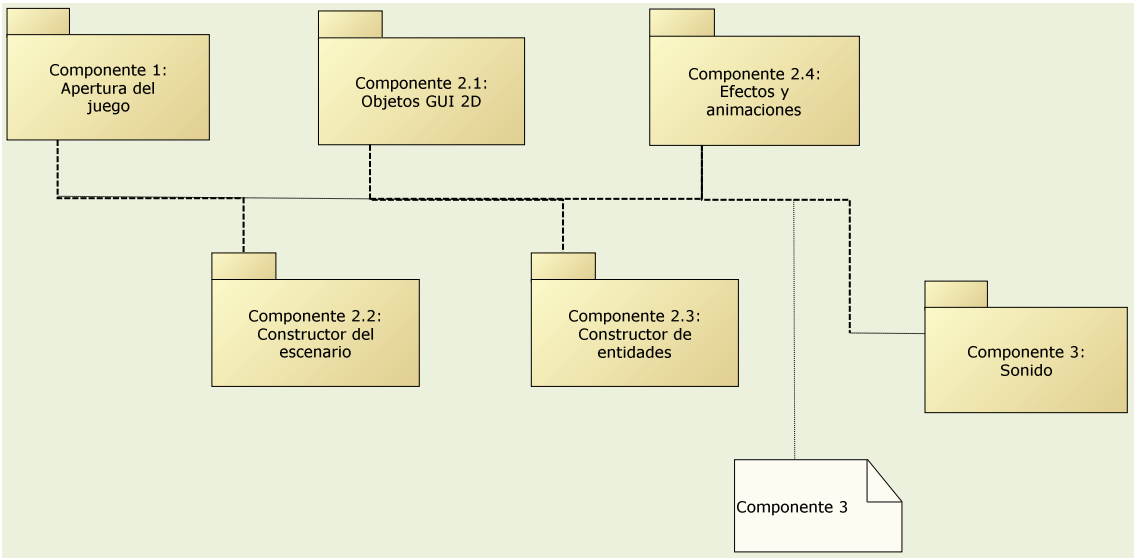
Componente 3 - Vista de componentes

Esta sección representa la vista de componentes.

Se representará un diagrama de componentes de alto nivel en el que se muestra junto con todos los demás componentes activos de la actualidad en el sistema. El propósito de la vistas de componentes es mostrar las dependencias entre componentes.

Componente 3 - Vista del sistema

Esta sección proporciona una vista de los componentes activos del sistema.



Componente 4 – Configuración del jugador

Componente 4 - Vista de requisitos

Esta sección identifica los requisitos y los casos de uso derivados del documento de requisitos del Software.

Componente 4 - Requisitos aplicables

La siguiente lista identifica los requisitos abordados:
Requisitos: 1, 2, 5, 7, 8

Componente 4 - Vista de casos de uso

Los siguientes casos de usos proporcionan una narración de una secuencia de acciones que pueden ser utilizados para situar el comportamiento del componente en su contexto.
Casos de uso: 14

Componente 4 - Vista del comportamiento – Interacciones

Esta sección describe las interacciones en el comportamiento del componente.

Componente 4 - Vista de la secuencia

La secuencia muestra la vista de las interacciones básicas, junto con la identidad genérica de las clases que interactúan.

USUARIO

Diagrama Secuencia

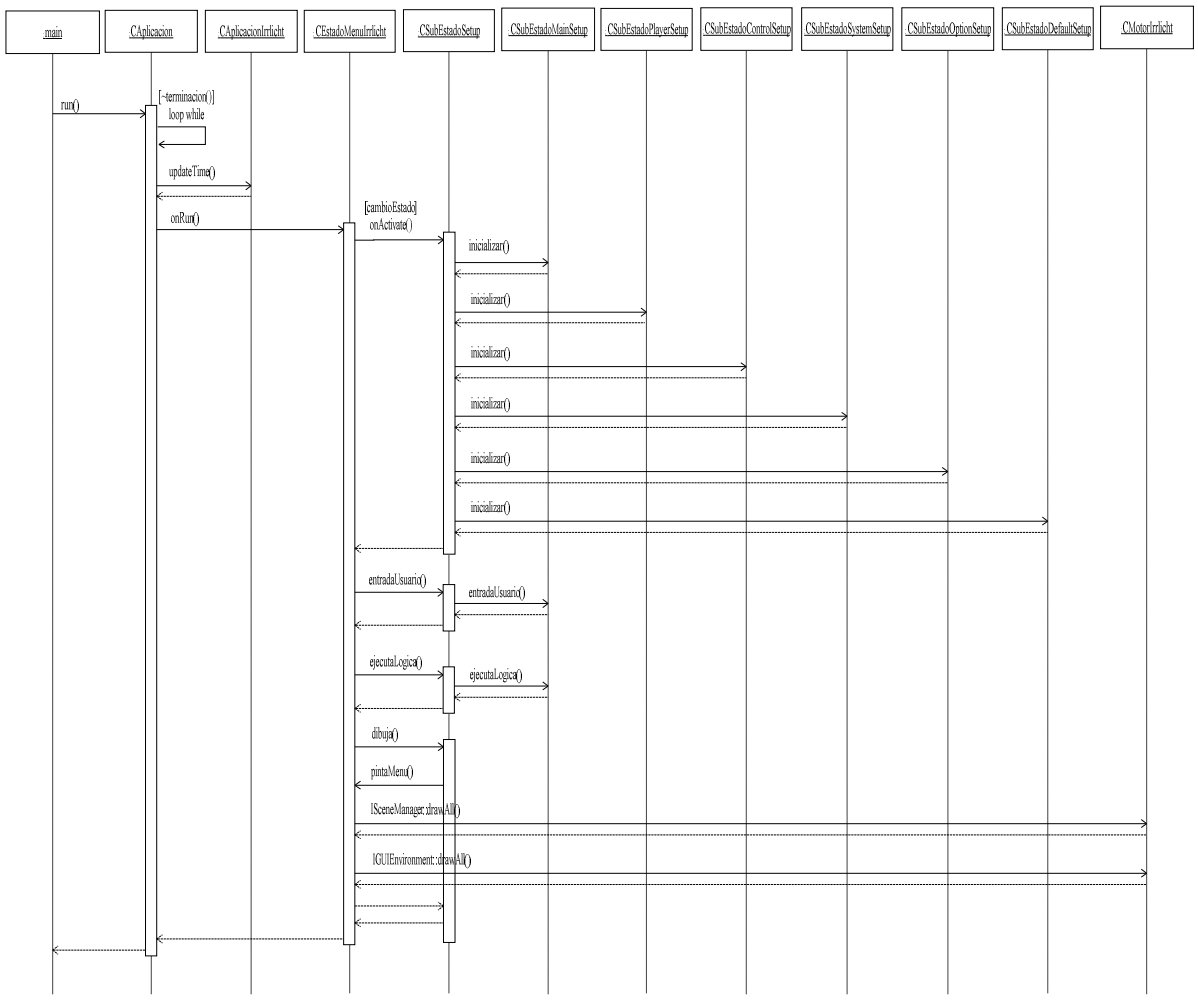
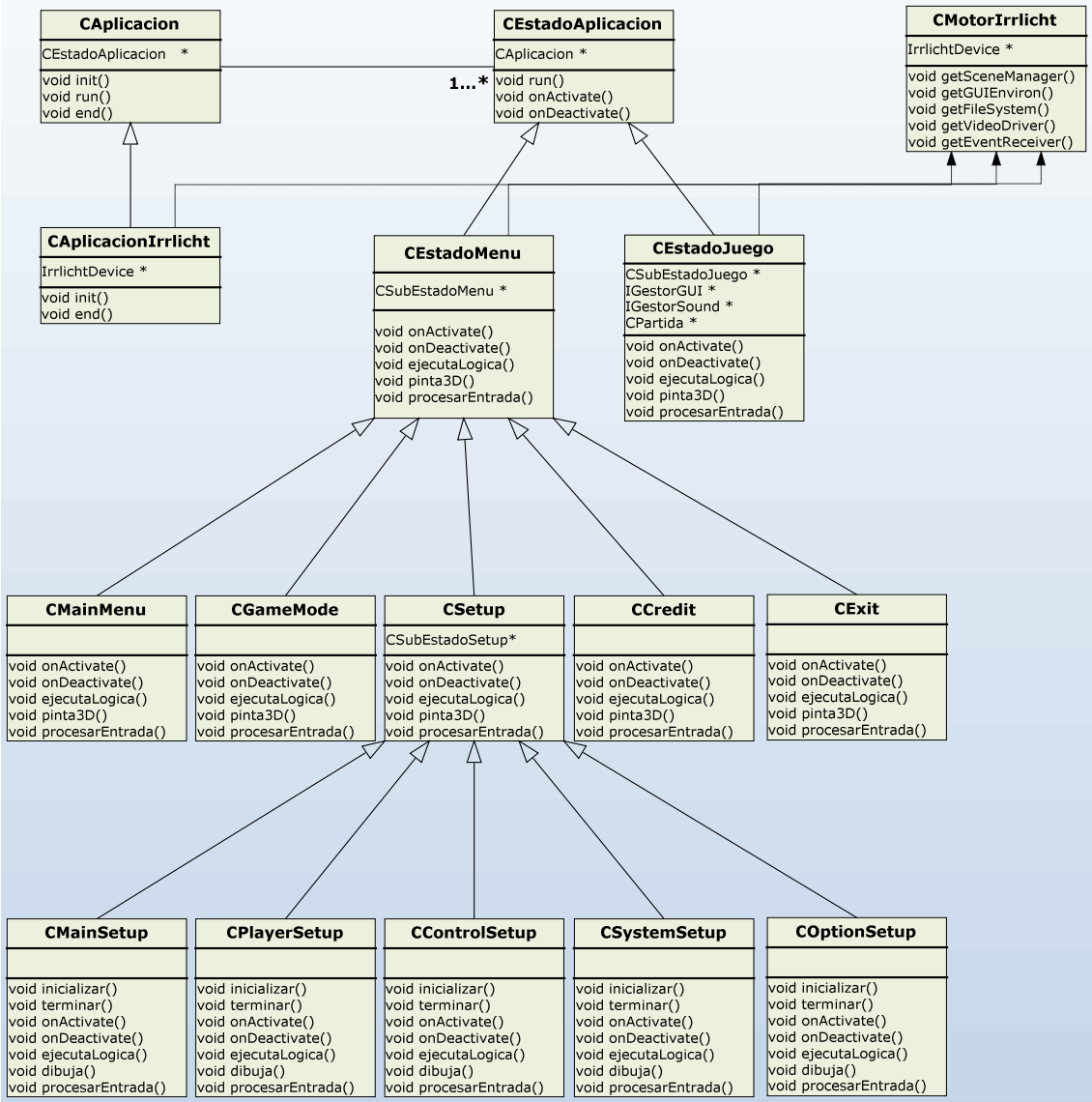


Figura 7.

Componente 4 - Vista de la lógica – Diagrama de clase

Esta sección describe las clases que se utilizan. Las operaciones definidas en las clases reflejan el comportamiento que se produce en este y otros componentes, pero aún no podrá representar la definición completa de la clase.



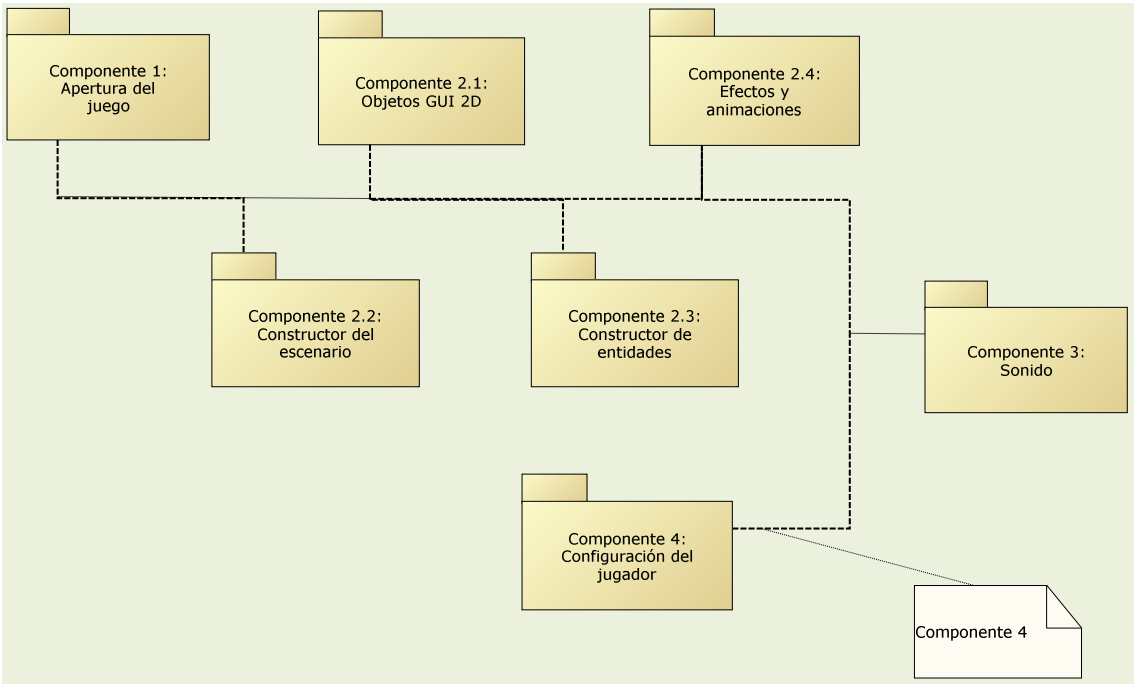
Componente 4 - Vista de componentes

Esta sección representa la vista de componentes.

Se representará un diagrama de componentes de alto nivel en el que se muestra junto con todos los demás componentes activos de la actualidad en el sistema. El propósito de la vistas de componentes es mostrar las dependencias entre componentes.

Componente 4 - Vista del sistema

Esta sección proporciona una vista de los componentes activos del sistema.



Componente 5 – Opciones del juego

Componente 5 - Vista de requisitos

Esta sección identifica los requisitos y los casos de uso derivados del documento de requisitos del Software.

Componente 5 - Requisitos aplicables

La siguiente lista identifica los requisitos abordados:
Requisitos: 1, 2, 5, 7, 19

Componente 5 - Vista de casos de uso

Los siguientes casos de usos proporcionan una narración de una secuencia de acciones que pueden ser utilizados para situar el comportamiento del componente en su contexto.
Casos de uso: 15

Componente 5 - Vista del comportamiento – Interacciones

Esta sección describe las interacciones en el comportamiento del componente.

Componente 5 - Vista de la secuencia

La secuencia muestra la vista de las interacciones básicas, junto con la identidad genérica de las clases que interactúan.

USUARIO

Diagrama Secuencia

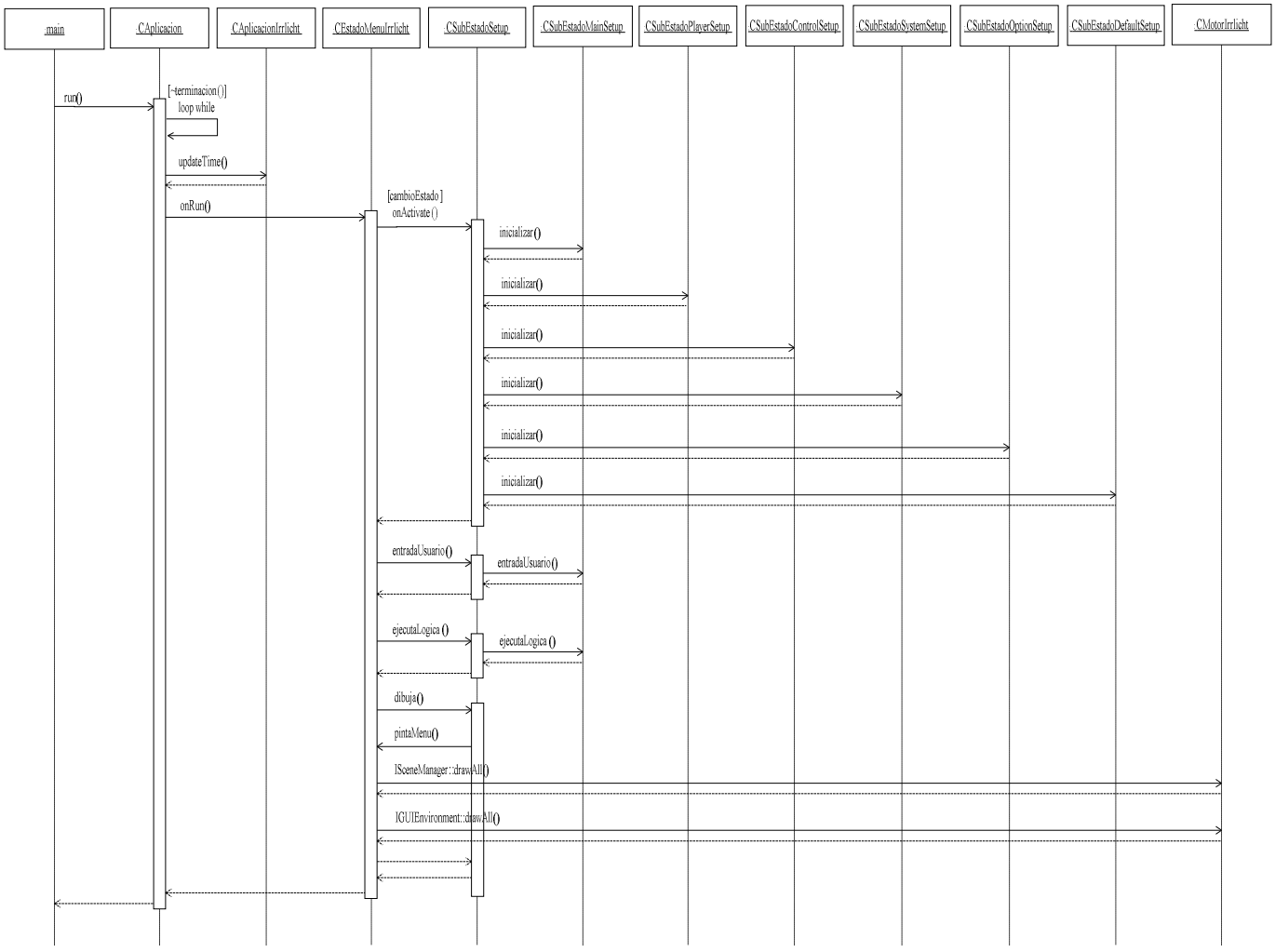
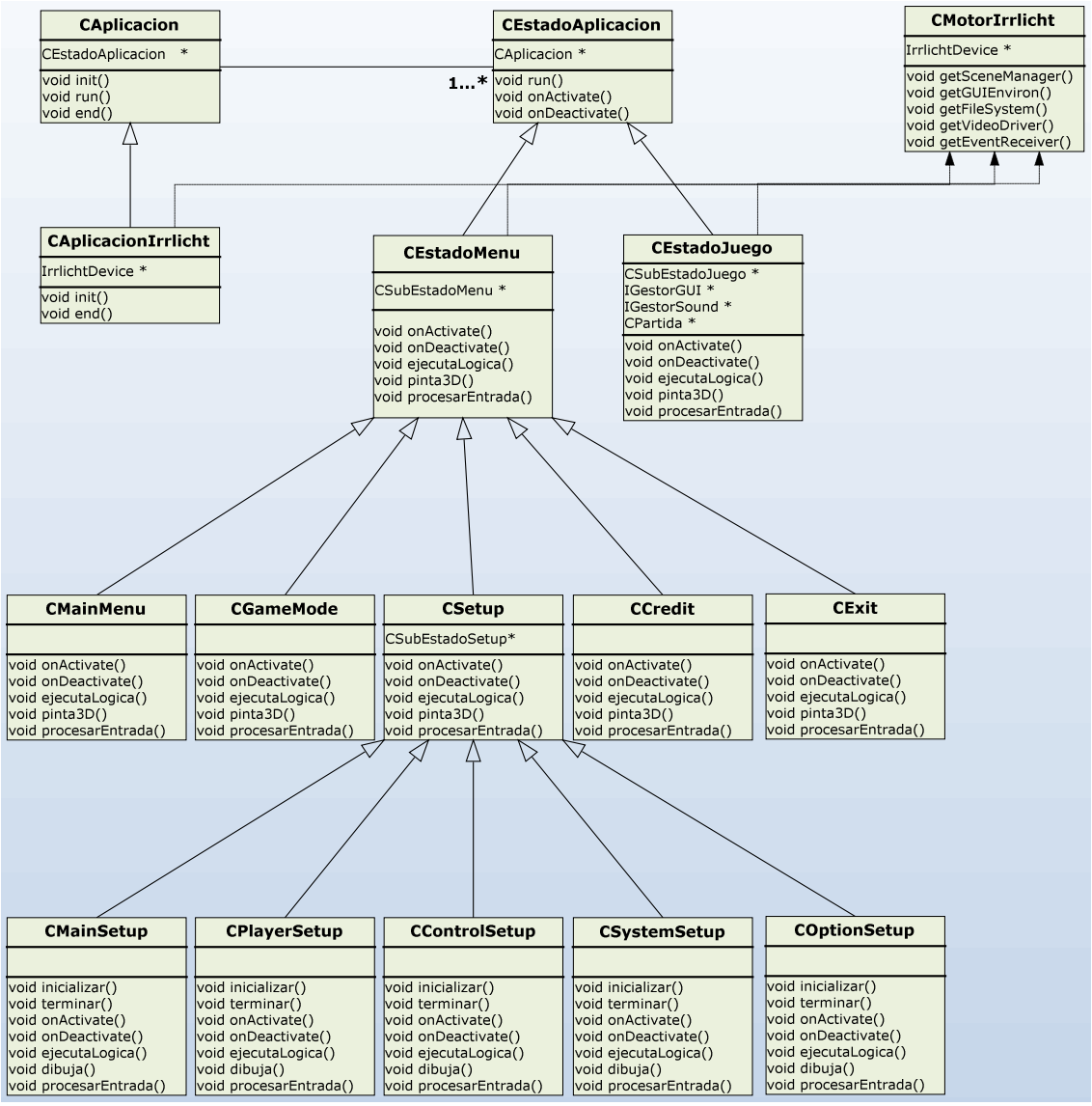


Figura 8.

Componente 5 - Vista de la lógica – Diagrama de clase

Esta sección describe las clases que se utilizan. Las operaciones definidas en las clases reflejan el comportamiento que se produce en este y otros componentes, pero aún no podrá representar la definición completa de la clase.



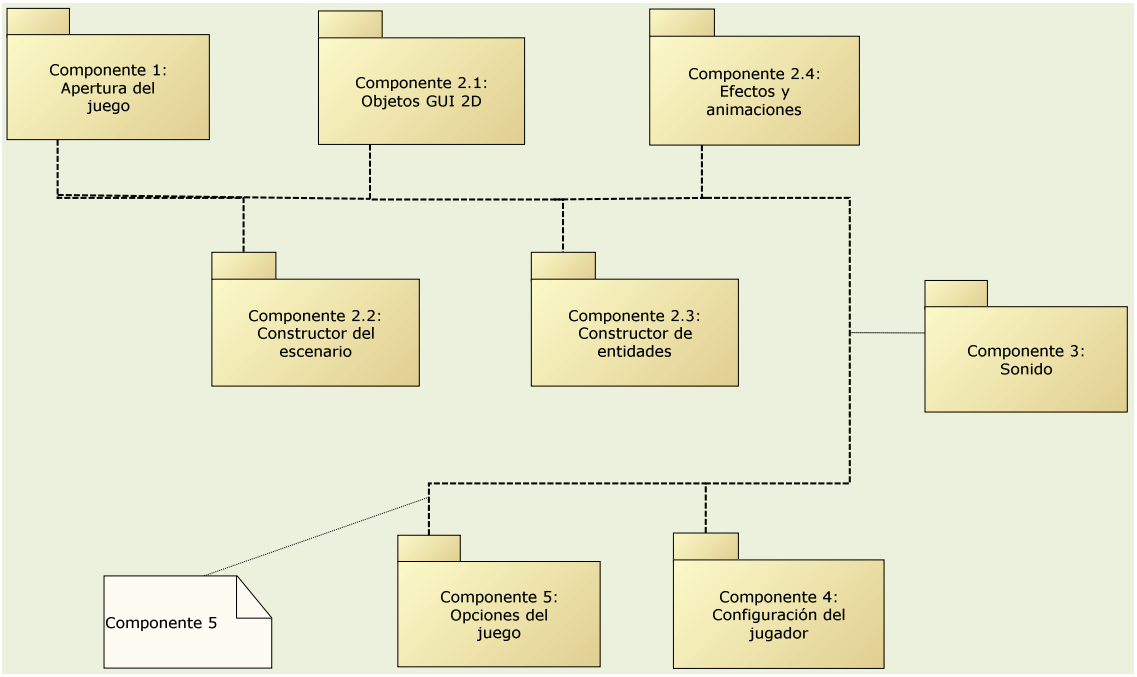
Componente 5 - Vista de componentes

Esta sección representa la vista de componentes.

Se representará un diagrama de componentes de alto nivel en el que se muestra junto con todos los demás componentes activos de la actualidad en el sistema. El propósito de la vistas de componentes es mostrar las dependencias entre componentes.

Componente 5 - Vista del sistema

Esta sección proporciona una vista de los componentes activos del sistema.



Componente 6 – Navegación en el nivel

Componente 6 - Vista de requisitos

Esta sección identifica los requisitos y los casos de uso derivados del documento de requisitos del Software.

Componente 6 - Requisitos aplicables

La siguiente lista identifica los requisitos abordados:
Requisitos: 1, 6, 7, 11, 25, 26

Componente 6 - Vista de casos de uso

Los siguientes casos de usos proporcionan una narración de una secuencia de acciones que pueden ser utilizados para situar el comportamiento del componente en su contexto.
Casos de uso: 16

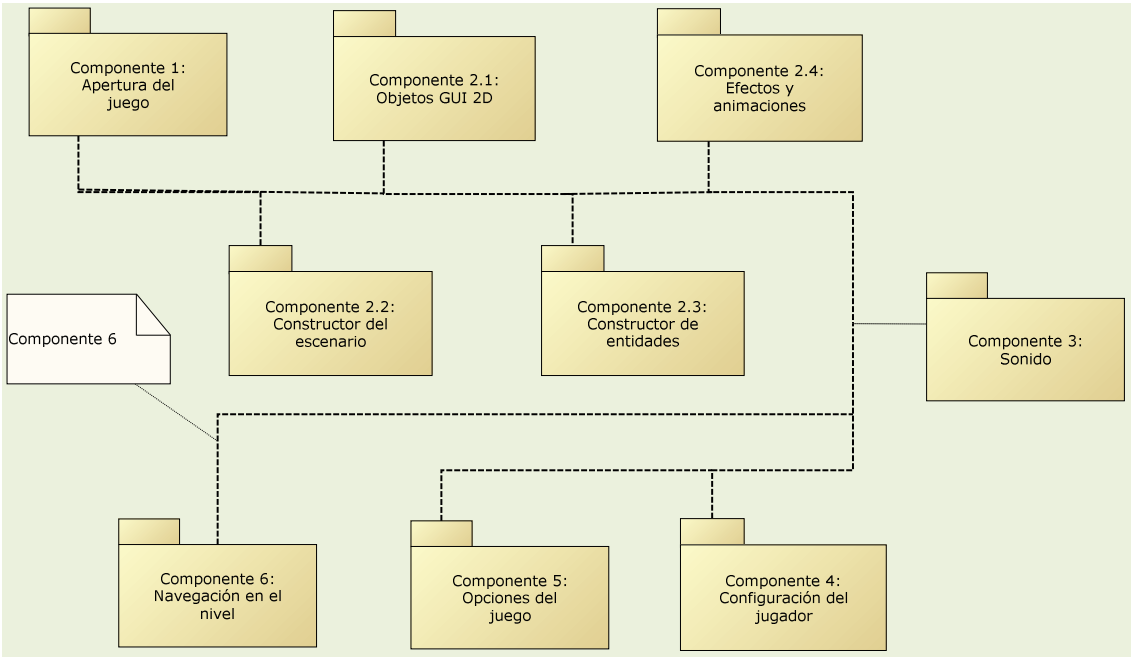
Componente 6 - Vista de componentes

Esta sección representa la vista de componentes.

Se representará un diagrama de componentes de alto nivel en el que se muestra junto con todos los demás componentes activos de la actualidad en el sistema. El propósito de la vistas de componentes es mostrar las dependencias entre componentes.

Componente 6 - Vista del sistema

Esta sección proporciona una vista de los componentes activos del sistema.



Componente 7 – Unidades físicas

Componente 7 - Vista de requisitos

Esta sección identifica los requisitos y los casos de uso derivados del documento de requisitos del Software.

Componente 7 - Requisitos aplicables

La siguiente lista identifica los requisitos abordados:
Requisitos: 1, 2, 7, 11, 29, 30, 31, 32, 33, 34, 38

Componente 7 - Vista de casos de uso

Los siguientes casos de usos proporcionan una narración de una secuencia de acciones que pueden ser utilizados para situar el comportamiento del componente en su contexto.
Casos de uso: 17, 18, 19, 20, 21, 26, 27

Componente 7 - Vista del comportamiento – Interacciones

Esta sección describe las interacciones en el comportamiento del componente.

Componente 7 - Vista de la secuencia

La secuencia muestra la vista de las interacciones básicas, junto con la identidad genérica de las clases que interactúan.

USUARIO

Diagrama Secuencia

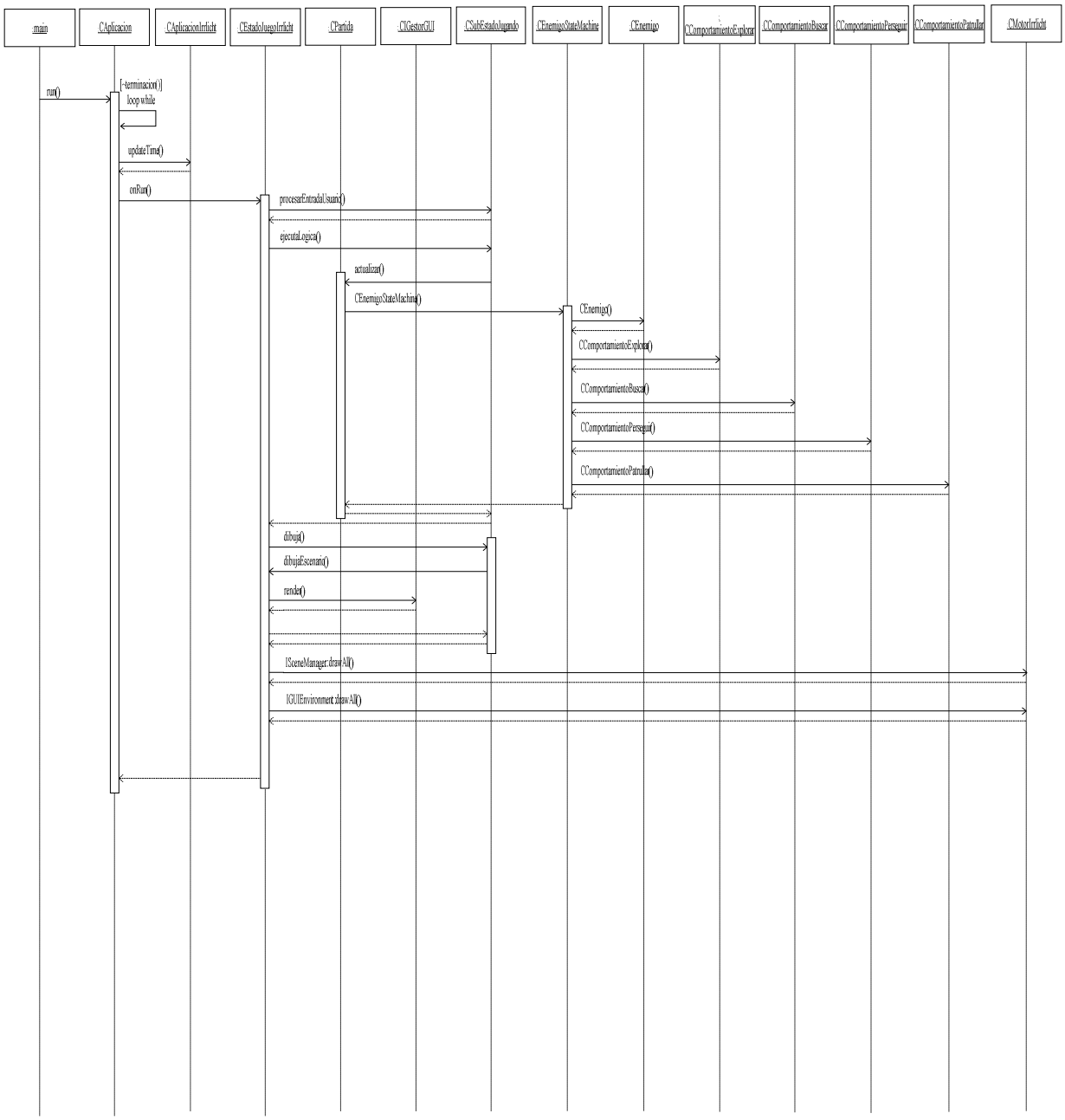
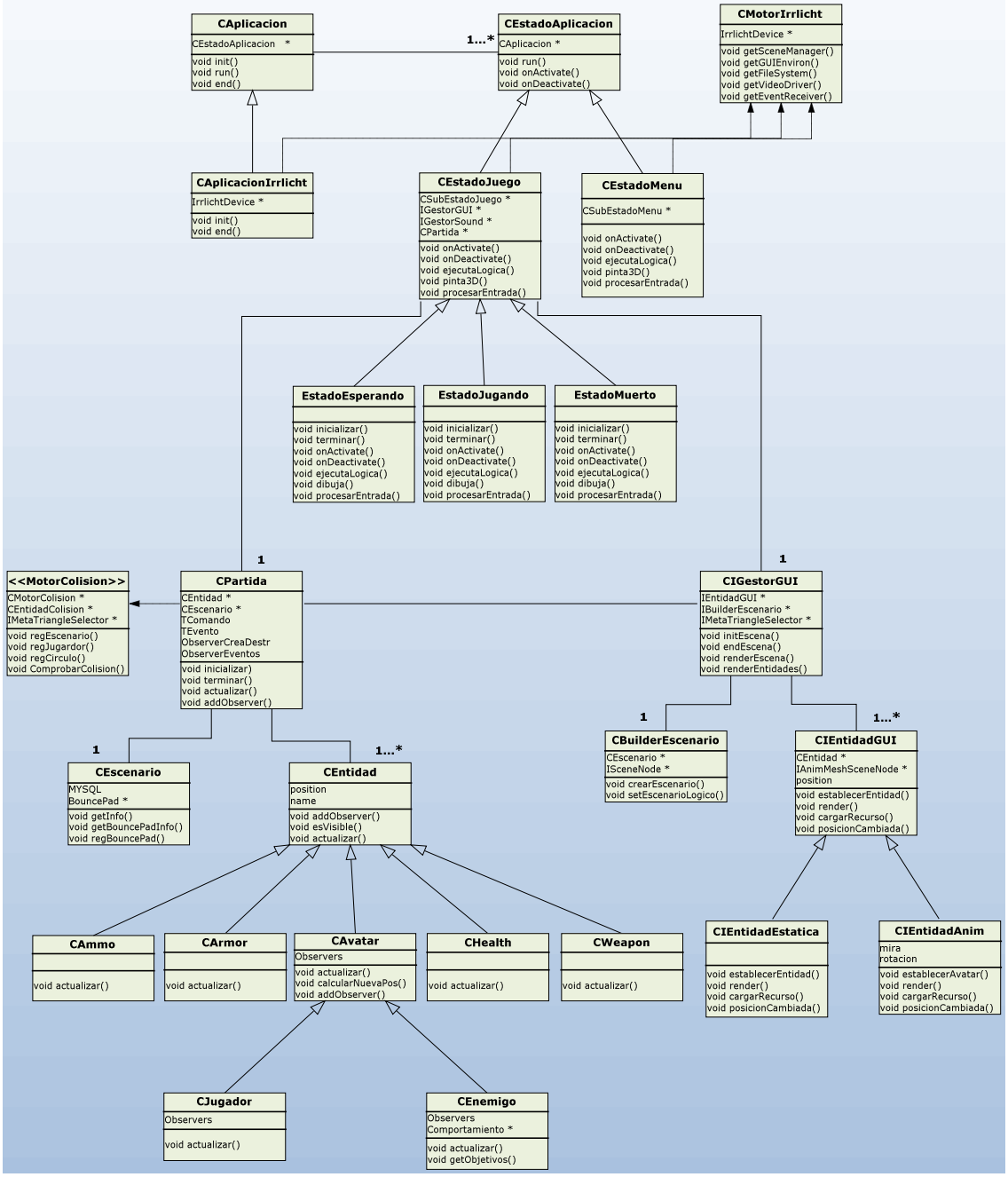


Figura 9.

Componente 7 - Vista de la lógica – Diagrama de clase

Esta sección describe las clases que se utilizan. Las operaciones definidas en las clases reflejan el comportamiento que se produce en este y otros componentes, pero aún no podrá representar la definición completa de la clase.



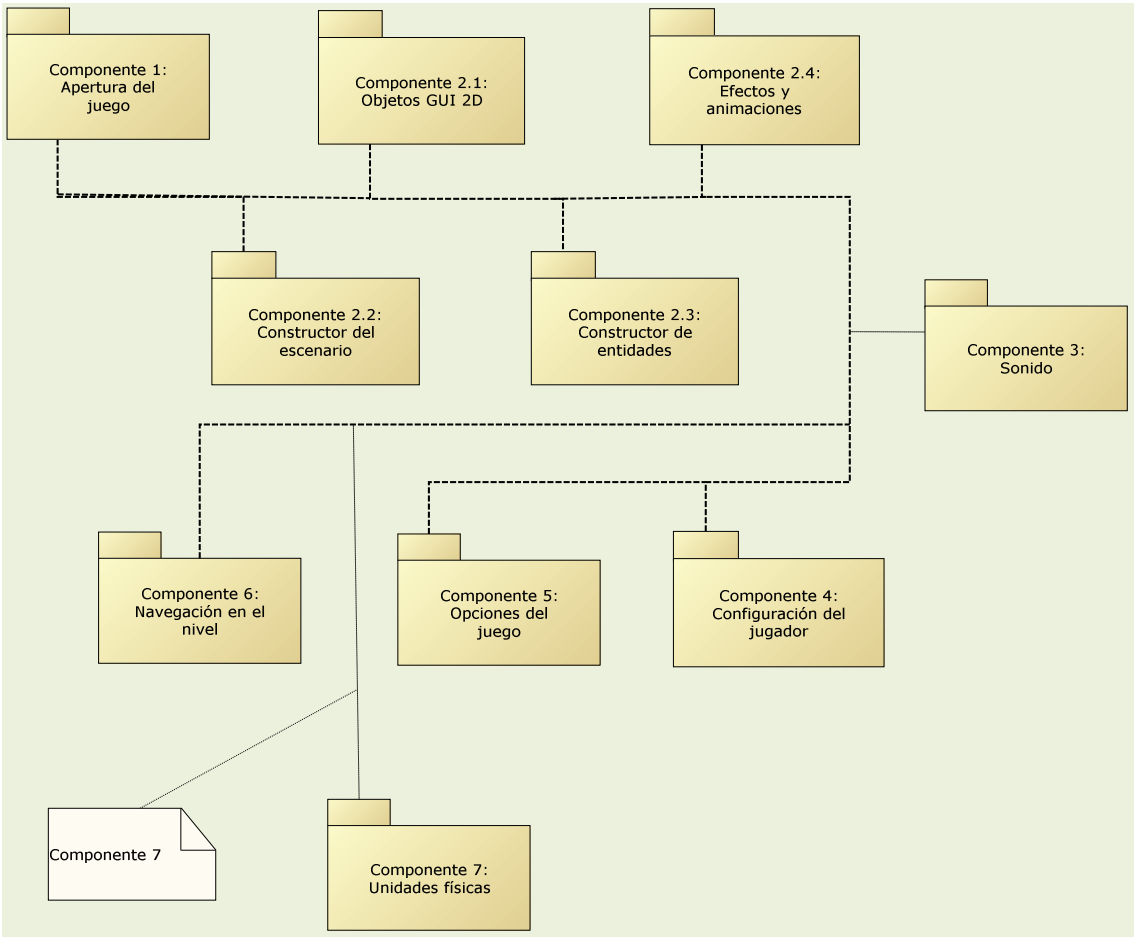
Componente 7 - Vista de componentes

Esta sección representa la vista de componentes.

Se representará un diagrama de componentes de alto nivel en el que se muestra junto con todos los demás componentes activos de la actualidad en el sistema. El propósito de la vistas de componentes es mostrar las dependencias entre componentes.

Componente 7 - Vista del sistema

Esta sección proporciona una vista de los componentes activos del sistema.



Componente 8 – Ítems

Componente 8 - Vista de requisitos

Esta sección identifica los requisitos y los casos de uso derivados del documento de requisitos del Software.

Componente 8 - Requisitos aplicables

La siguiente lista identifica los requisitos abordados:
Requisitos: 35, 36, 37

Componente 8 - Vista de casos de uso

Los siguientes casos de usos proporcionan una narración de una secuencia de acciones que pueden ser utilizados para situar el comportamiento del componente en su contexto.
Casos de uso: 25

Componente 8 - Vista del comportamiento – Interacciones

Esta sección describe las interacciones en el comportamiento del componente.

Componente 8 - Vista de la secuencia

La secuencia muestra la vista de las interacciones básicas, junto con la identidad genérica de las clases que interactúan.

USUARIO

Diagrama Secuencia

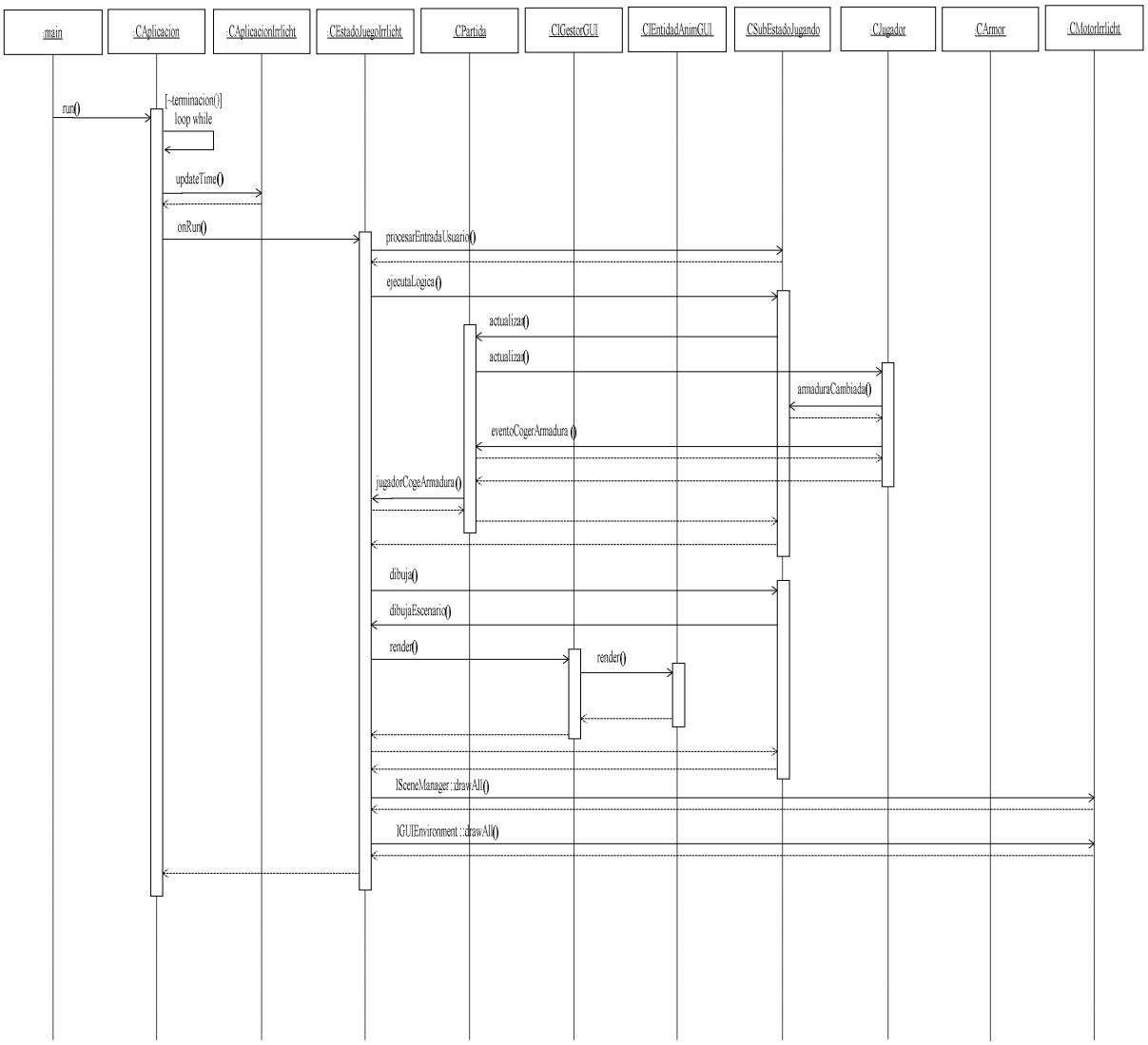
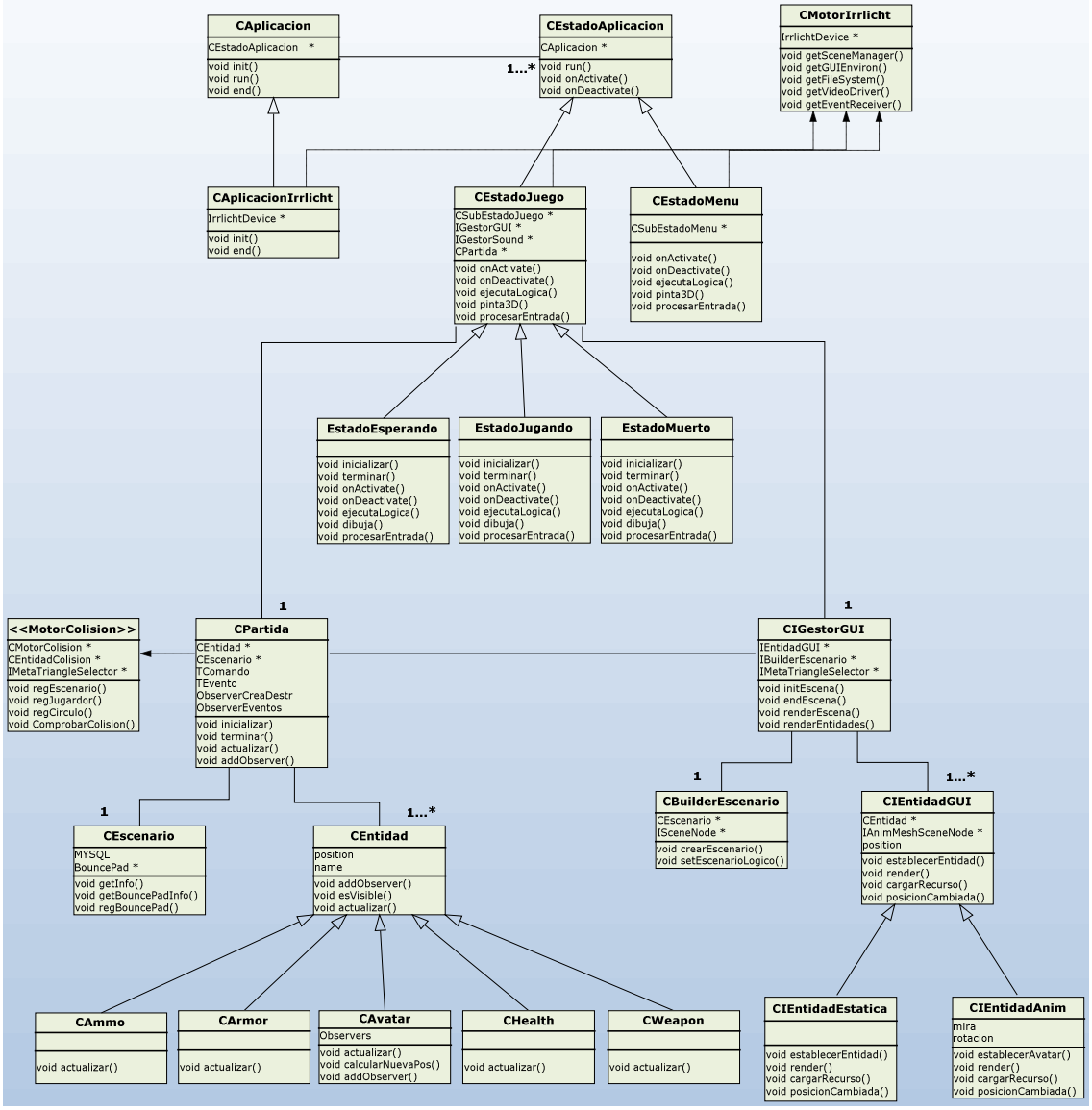


Figura 10.

Componente 8 - Vista de la lógica – Diagrama de clase

Esta sección describe las clases que se utilizan. Las operaciones definidas en las clases reflejan el comportamiento que se produce en este y otros componentes, pero aún no podrá representar la definición completa de la clase.



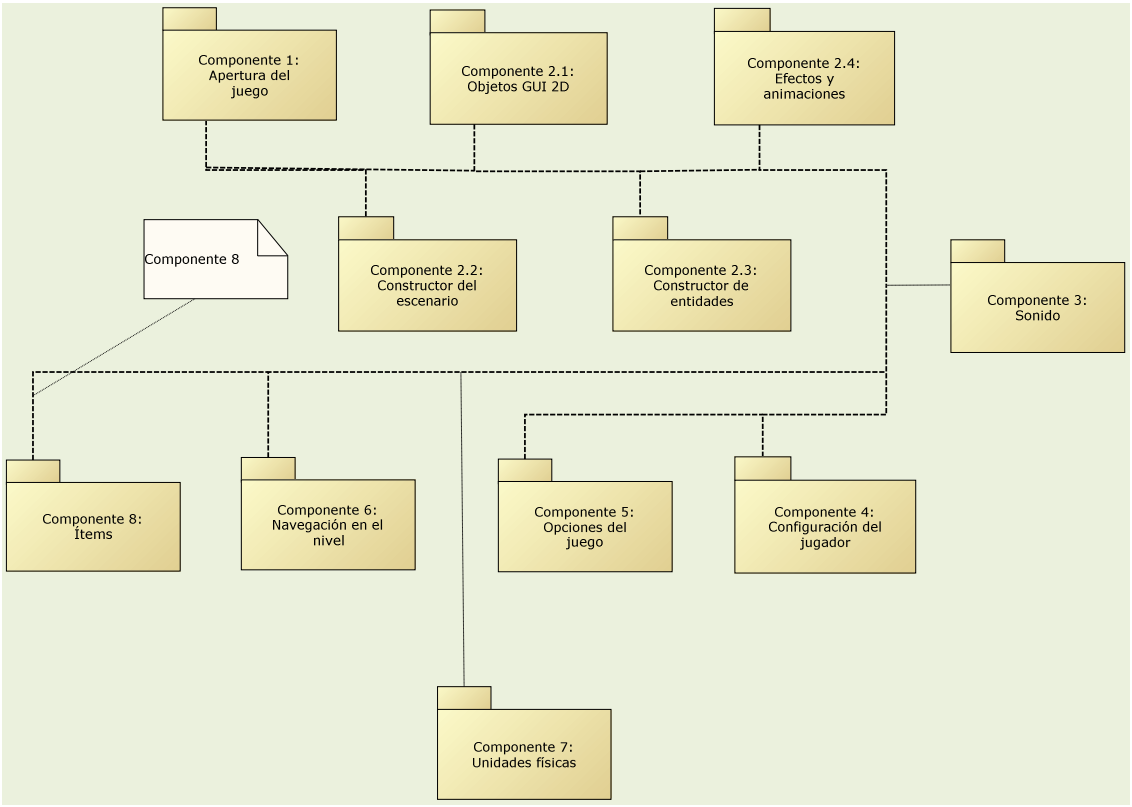
Componente 8 - Vista de componentes

Esta sección representa la vista de componentes.

Se representará un diagrama de componentes de alto nivel en el que se muestra junto con todos los demás componentes activos de la actualidad en el sistema. El propósito de la vistas de componentes es mostrar las dependencias entre componentes.

Componente 8 - Vista del sistema

Esta sección proporciona una vista de los componentes activos del sistema.



Componente 9 – Combate

Componente 9 - Vista de requisitos

Esta sección identifica los requisitos y los casos de uso derivados del documento de requisitos del Software.

Componente 9 - Requisitos aplicables

La siguiente lista identifica los requisitos abordados:
Requisitos: 39, 40

Componente 9 - Vista de casos de uso

Los siguientes casos de usos proporcionan una narración de una secuencia de acciones que pueden ser utilizados para situar el comportamiento del componente en su contexto.
Casos de uso: 28

Componente 9 - Vista del comportamiento – Interacciones

Esta sección describe las interacciones en el comportamiento del componente.

Componente 9 - Vista de la secuencia

La secuencia muestra la vista de las interacciones básicas, junto con la identidad genérica de las clases que interactúan.

USUARIO | Diagrama Secuencia

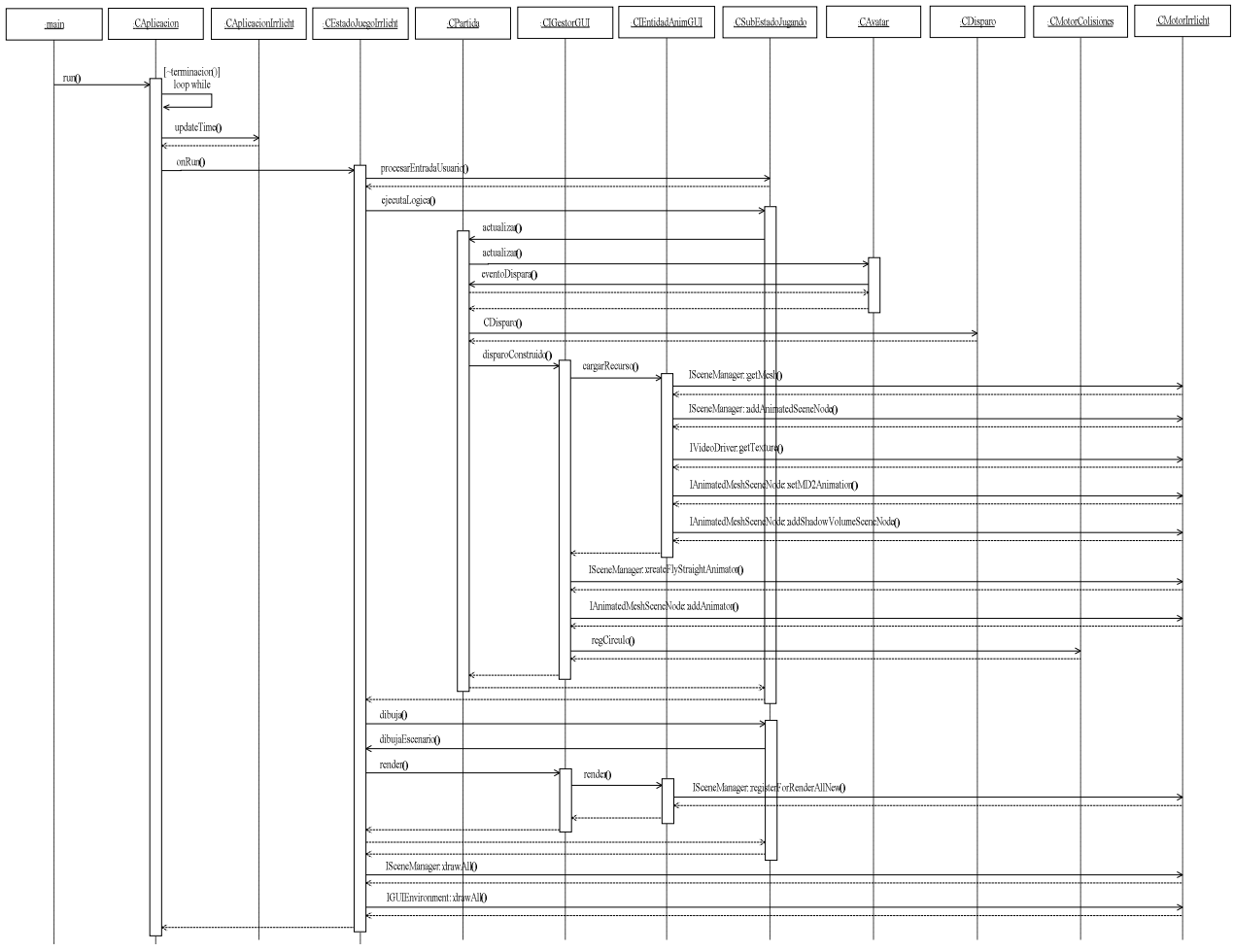
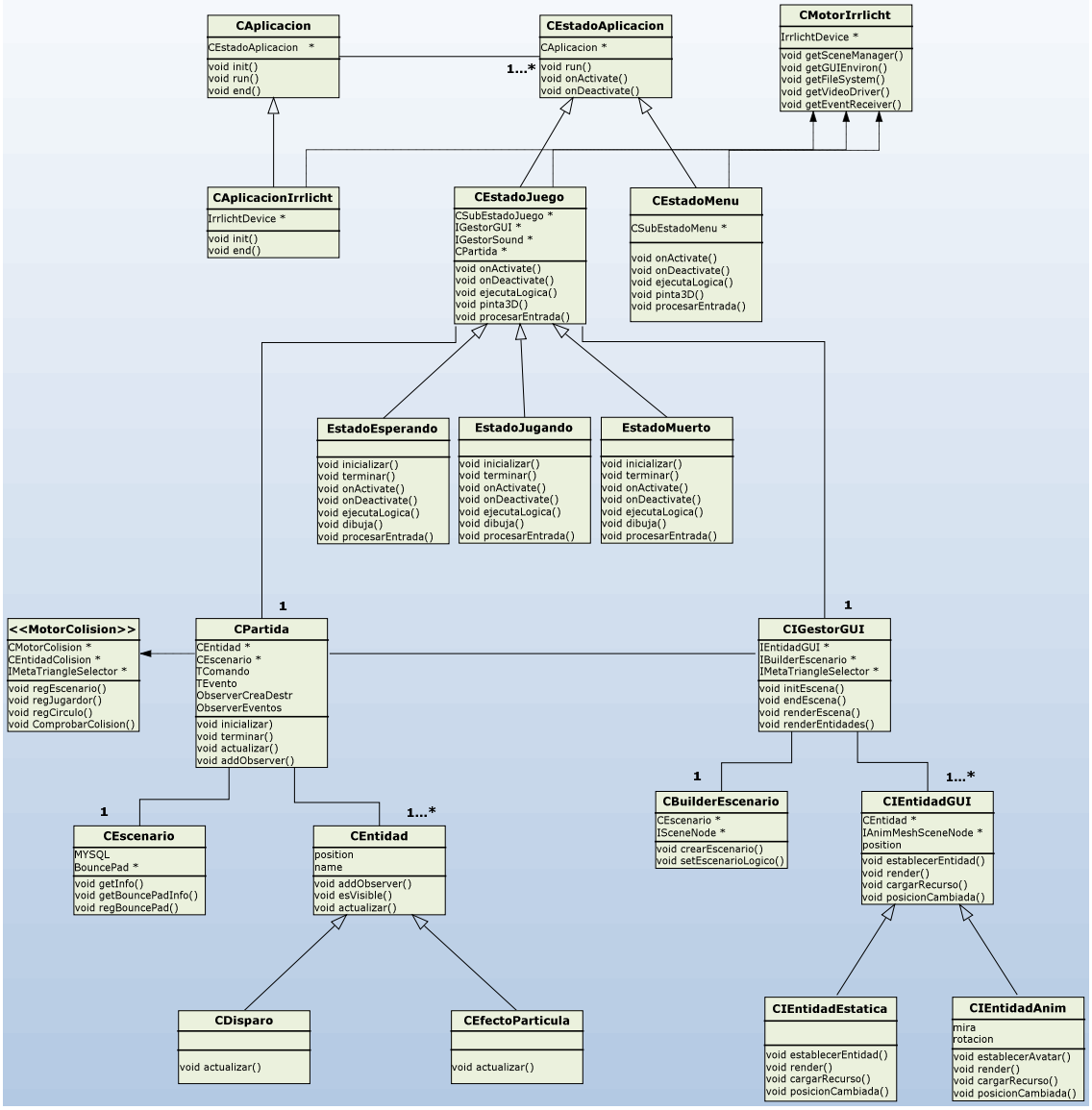


Figura 11.

Componente 9 - Vista de la lógica – Diagrama de clase

Esta sección describe las clases que se utilizan. Las operaciones definidas en las clases reflejan el comportamiento que se produce en este y otros componentes, pero aún no podrá representar la definición completa de la clase.



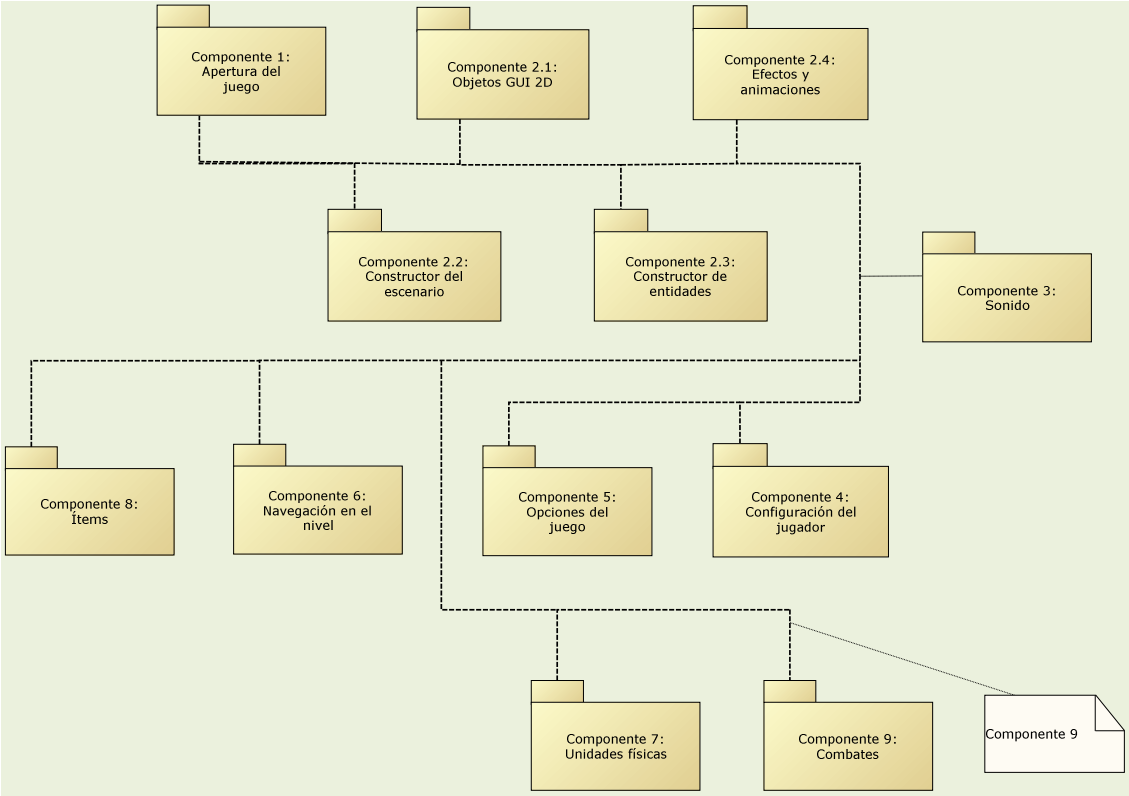
Componente 9 - Vista de componentes

Esta sección representa la vista de componentes.

Se representará un diagrama de componentes de alto nivel en el que se muestra junto con todos los demás componentes activos de la actualidad en el sistema. El propósito de la vistas de componentes es mostrar las dependencias entre componentes.

Componente 9 - Vista del sistema

Esta sección proporciona una vista de los componentes activos del sistema.



Componente 10 – Inteligencia Artificial

Componente 10 - Vista de requisitos

Esta sección identifica los requisitos y los casos de uso derivados del documento de requisitos del Software.

Componente 10 - Requisitos aplicables

La siguiente lista identifica los requisitos abordados:
Requisitos: 31, 41, 42

Componente 10 - Vista de casos de uso

Los siguientes casos de usos proporcionan una narración de una secuencia de acciones que pueden ser utilizados para situar el comportamiento del componente en su contexto.
Casos de uso: 28

Componente 10 - Vista del comportamiento – Interacciones

Esta sección describe las interacciones en el comportamiento del componente.

Componente 10 - Vista de la secuencia

La secuencia muestra la vista de las interacciones básicas, junto con la identidad genérica de las clases que interactúan.

USUARIO

Diagrama Secuencia

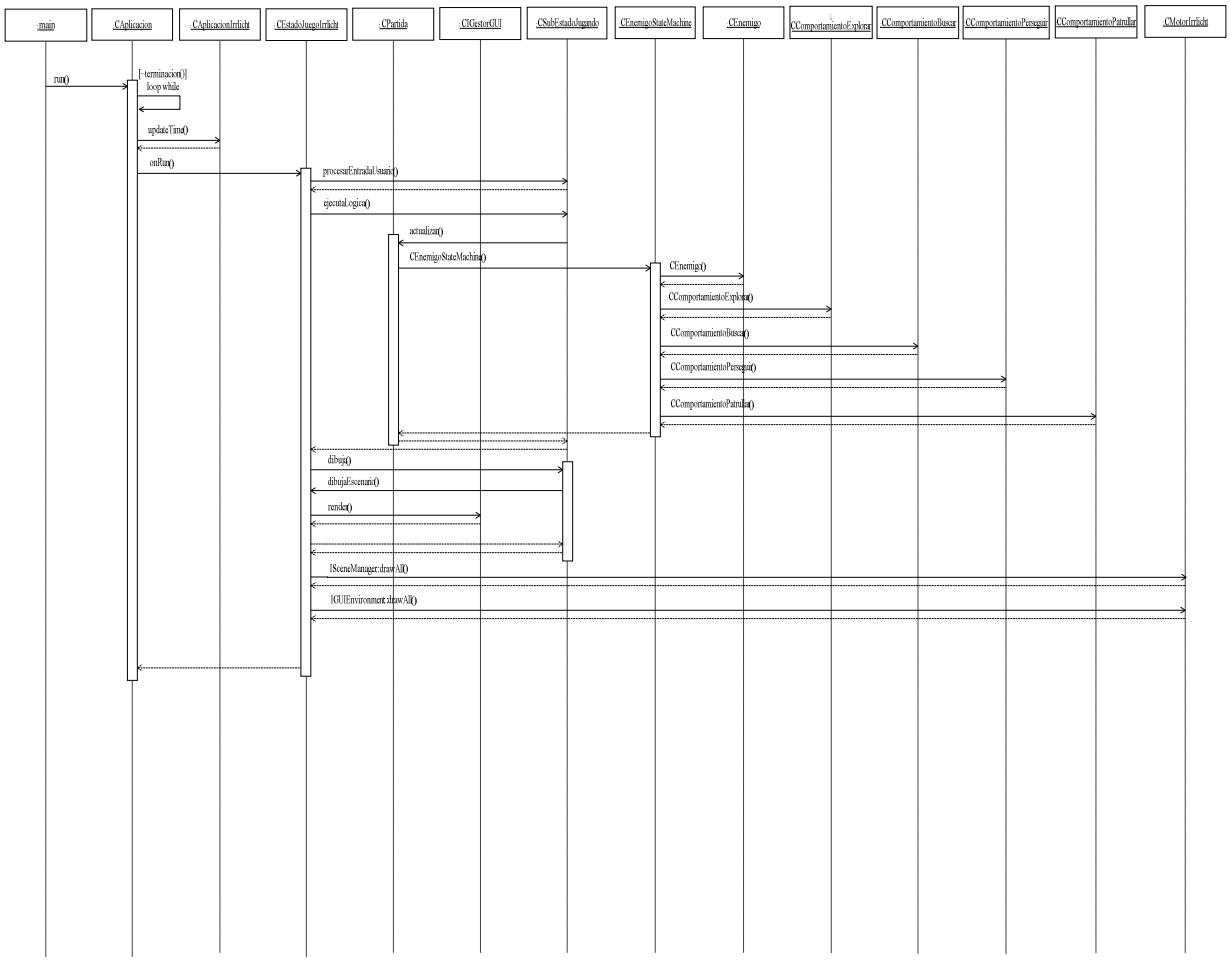
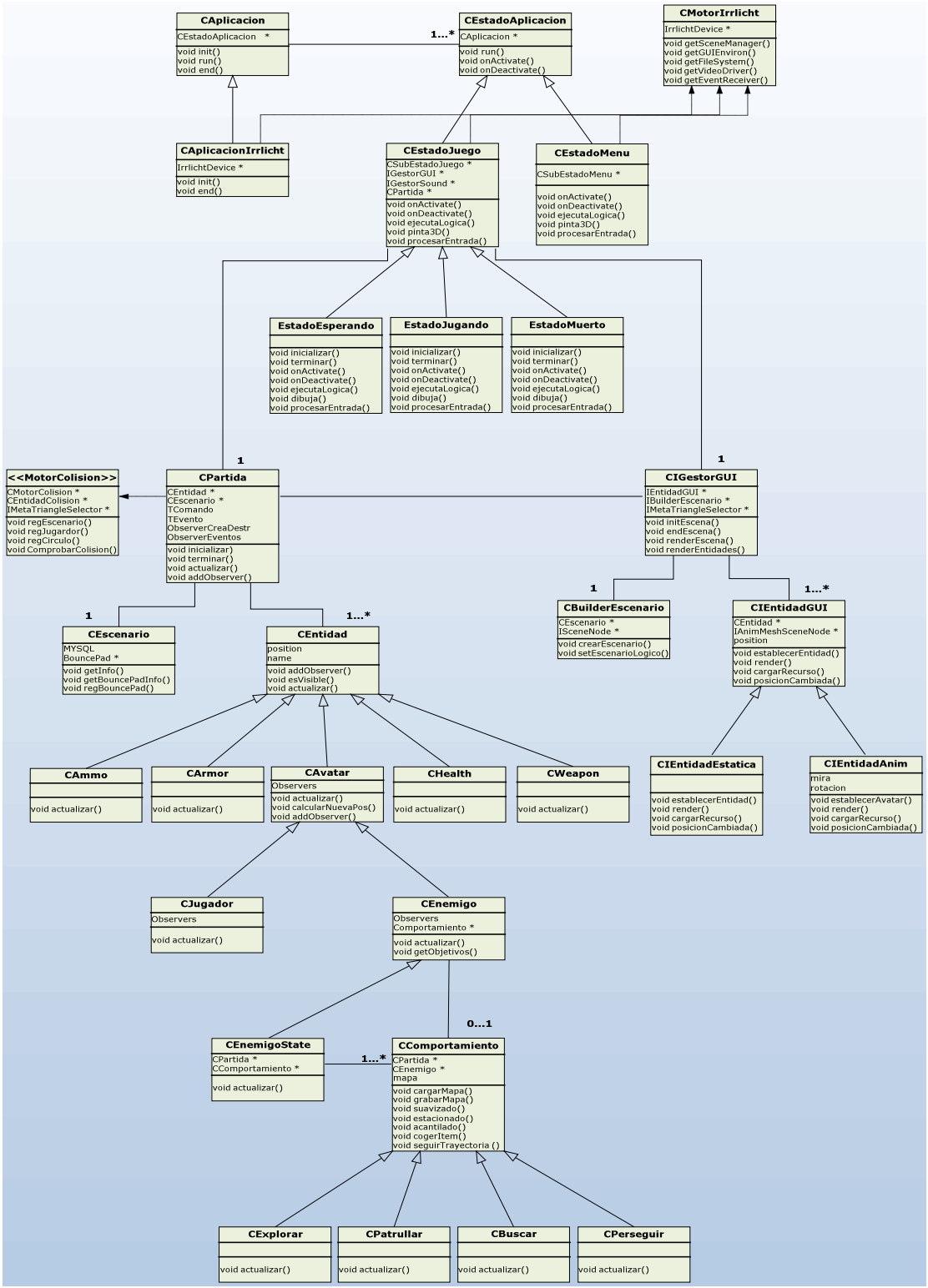


Figura 12.

Componente 10 - Vista de la lógica – Diagrama de clase

Esta sección describe las clases que se utilizan. Las operaciones definidas en las clases reflejan el comportamiento que se produce en este y otros componentes, pero aún no podrá representar la definición completa de la clase.



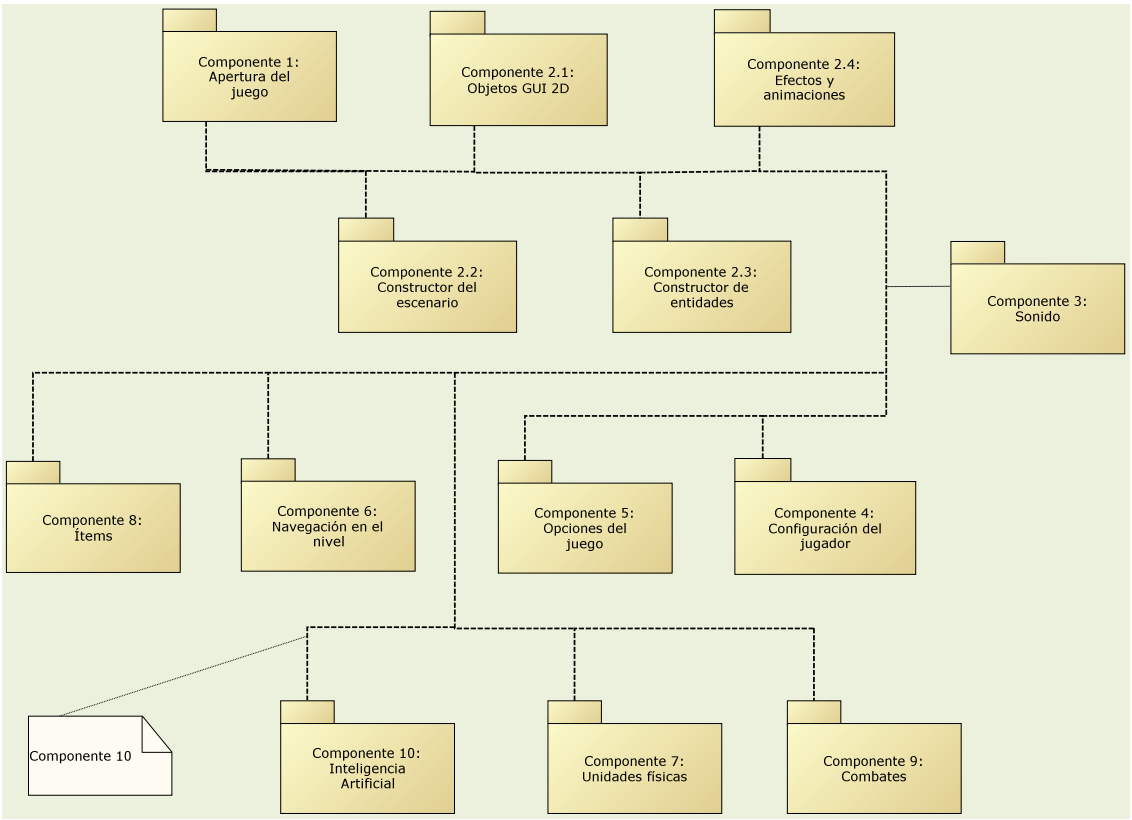
Componente 10 - Vista de componentes

Esta sección representa la vista de componentes.

Se representará un diagrama de componentes de alto nivel en el que se muestra junto con todos los demás componentes activos de la actualidad en el sistema. El propósito de la vistas de componentes es mostrar las dependencias entre componentes.

Componente 10 - Vista del sistema

Esta sección proporciona una vista de los componentes activos del sistema.



Componente 11 – Vista de las estadísticas

Componente 11 - Vista de requisitos

Esta sección identifica los requisitos y los casos de uso derivados del documento de requisitos del Software.

Componente 11 - Requisitos aplicables

La siguiente lista identifica los requisitos abordados:
Requisitos: 2, 6, 7, 11, 43

Componente 11 - Vista de casos de uso

Los siguientes casos de usos proporcionan una narración de una secuencia de acciones que pueden ser utilizados para situar el comportamiento del componente en su contexto.
Casos de uso: 23

Componente 11 - Vista del comportamiento – Interacciones

Esta sección describe las interacciones en el comportamiento del componente.

Componente 11 - Vista de la secuencia

La secuencia muestra la vista de las interacciones básicas, junto con la identidad genérica de las clases que interactúan.

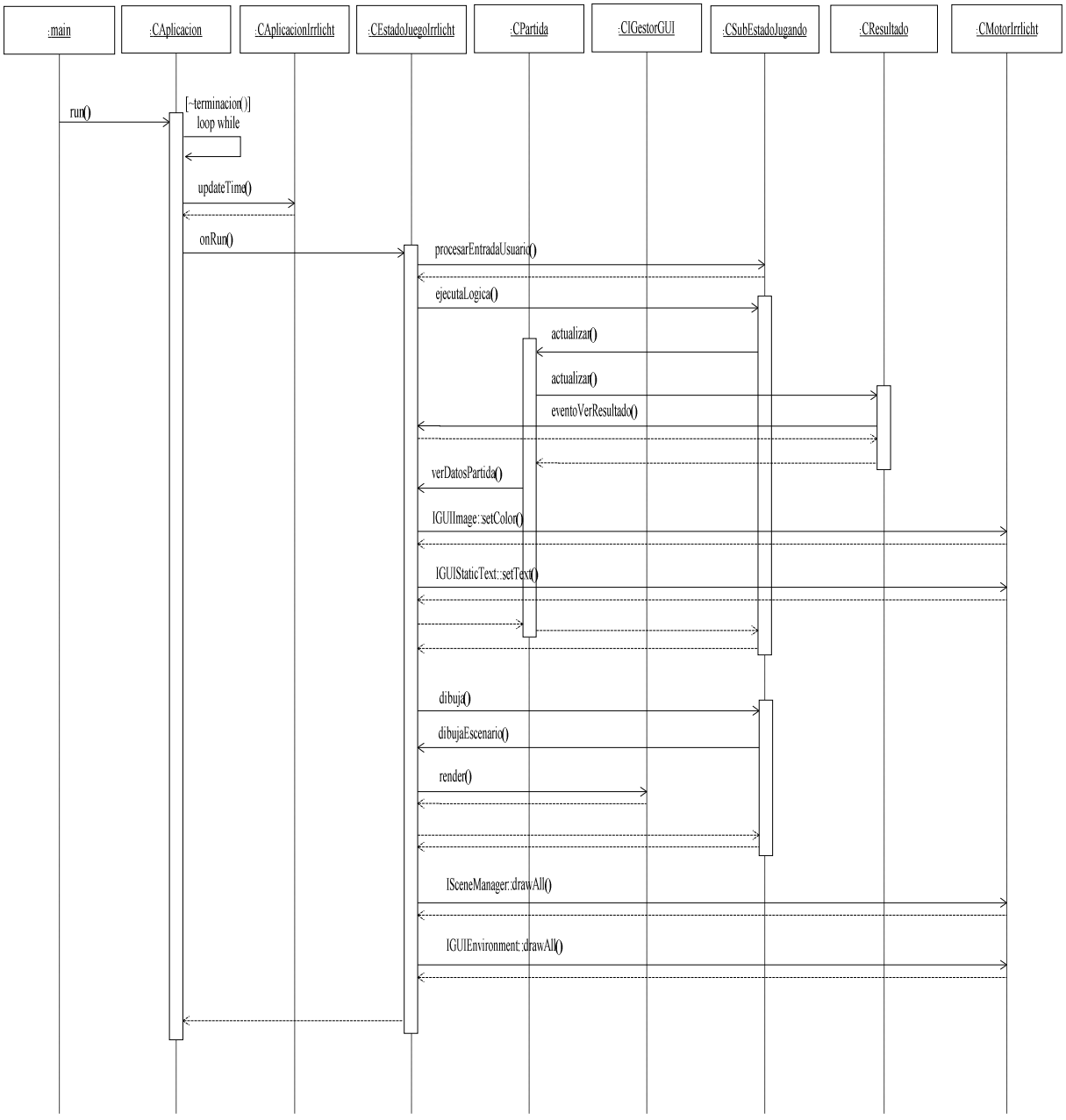
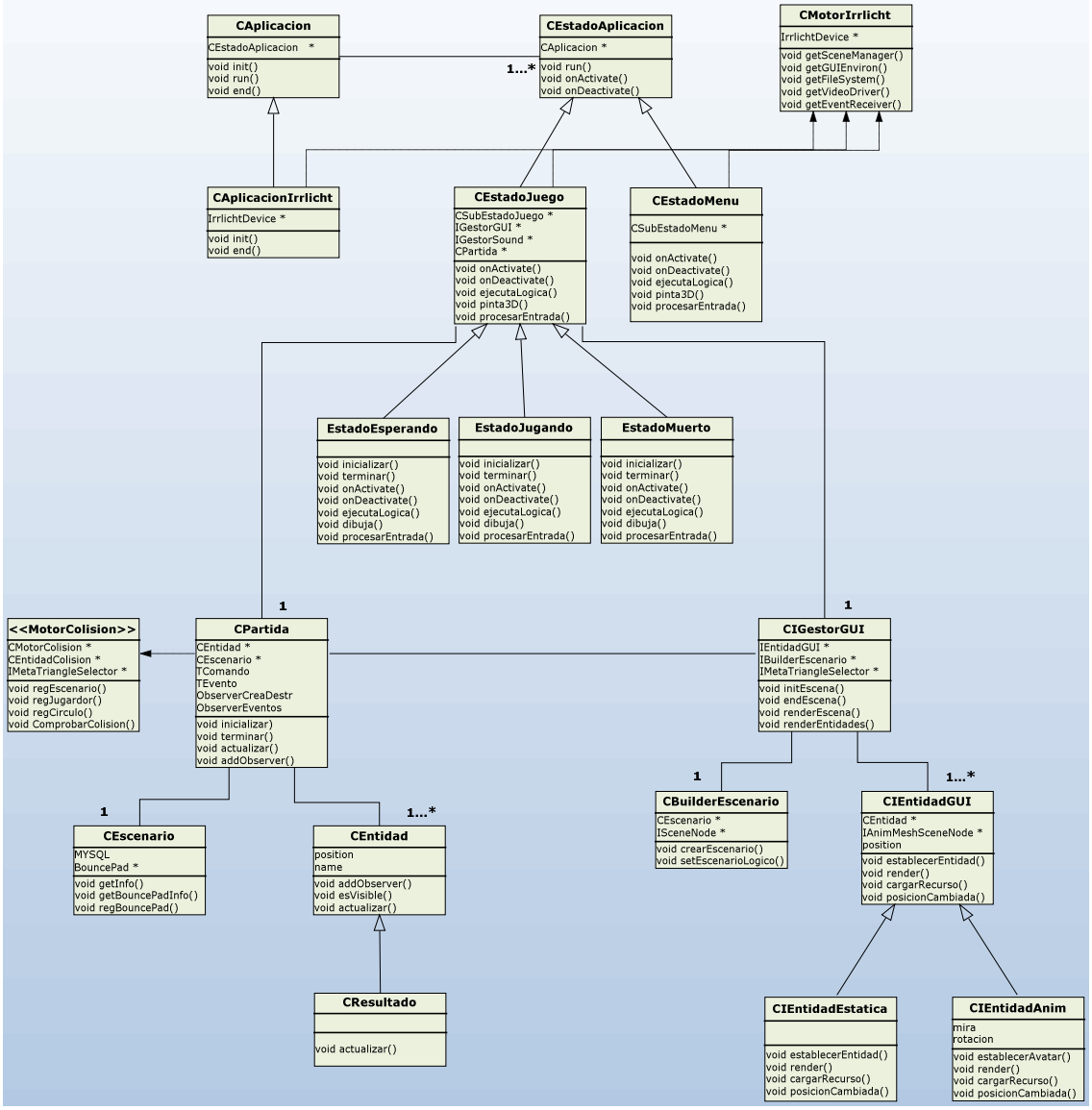

:main
:CAplicacion
:CAplicacionIrrlicht
:CEstadoJuegoIrrlicht
:CPartida
:CIGestorGUI
:CSubEstadoJugando
:CResultado
:CMotorIrrlicht
run()
[~terminacion()]
loop while
updateTime()
onRun()
procesarEntradaUsuario()
ejecutaLogica()
actualizar()
actualizar()
eventoVerResultado()
verDatosPartida()
IGUIImage::setColor()
IGUIStaticText::setText()
dibuja()
dibujaEscenario()
render()
ISceneManager::drawAll()
IGUIEnvironment::drawAll()

Figura 13.

Componente 11 - Vista de la lógica – Diagrama de clase

Esta sección describe las clases que se utilizan. Las operaciones definidas en las clases reflejan el comportamiento que se produce en este y otros componentes, pero aún no podrá representar la definición completa de la clase.



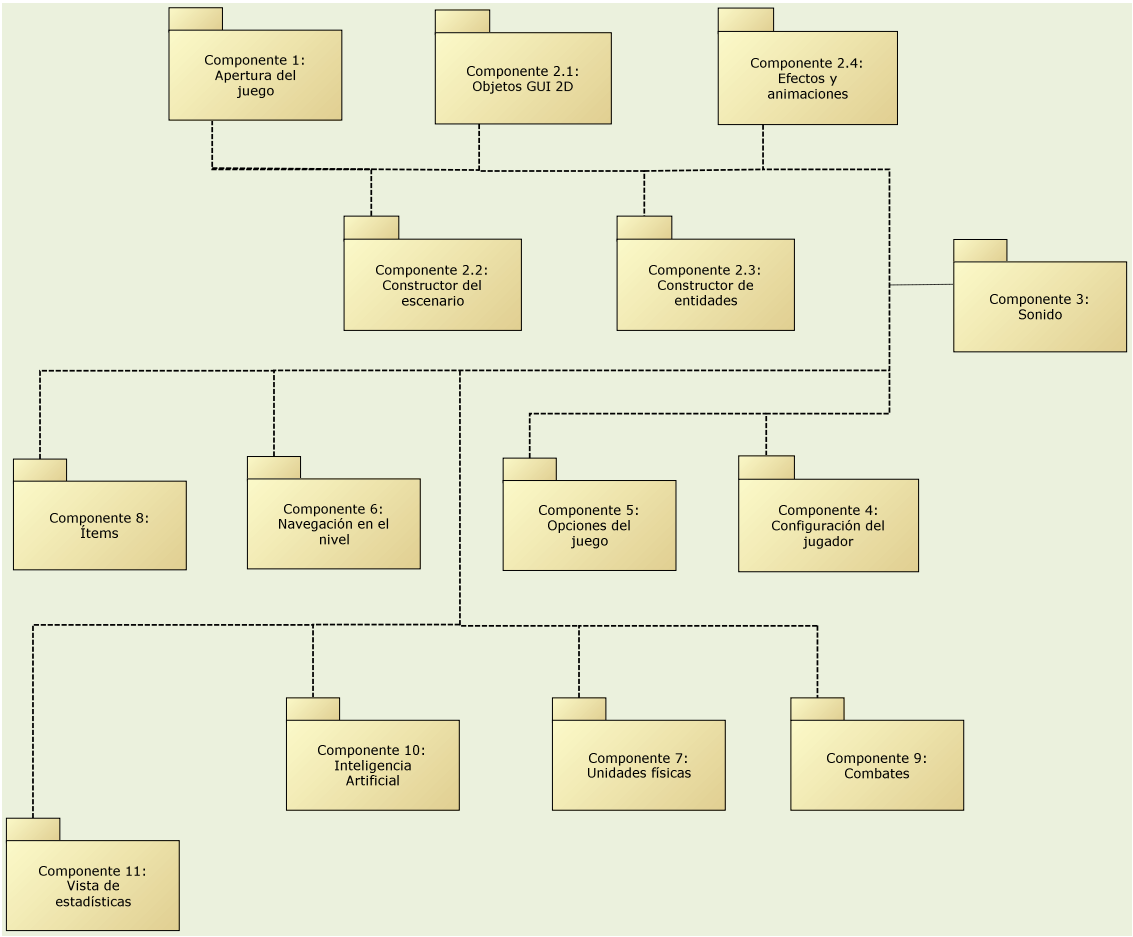
Componente 11 - Vista de componentes

Esta sección representa la vista de componentes.

Se representará un diagrama de componentes de alto nivel en el que se muestra junto con todos los demás componentes activos de la actualidad en el sistema. El propósito de la vistas de componentes es mostrar las dependencias entre componentes.

Componente 11 - Vista del sistema

Esta sección proporciona una vista de los componentes activos del sistema.



6. Programación

Programas, Librerías y Tecnologías Empleadas

Para el desarrollo del Proyecto Fin de Carrera, se han empleado una serie de herramientas mediante las cuales se podrá abordar los diferentes propósitos planteados. En éste apartado se comentan algunas de las características tanto de los entornos usados como las principales librerías empleadas.

Microsoft Visual Studio 2003 es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para sistemas Windows. Soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros.

Visual Studio permite a los desarrolladores crear aplicaciones, sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET. Así se pueden crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas web y dispositivos móviles.

Provee un conjunto de herramientas de desarrollo profesionales para programadores individuales o para aquellos trabajando en pequeños equipos y que están construyendo aplicaciones para Windows.

DirectX es una colección de APIs creadas y recreadas para facilitar las complejas tareas relacionadas con multimedia, especialmente programación de juegos en la plataforma Microsoft Windows.

Consta de los siguientes APIs:

- Direct3D: utilizado para el procesado y la programación de gráficos en tres dimensiones (una de las características más usadas de DirectX).
- Direct Graphics: para dibujar imágenes en dos dimensiones (planas), y para representación de imágenes en tres dimensiones.
- DirectInput: utilizado para procesar datos del teclado, mouse, joystick y otros controles para juegos.
- DirectPlay: para comunicaciones en red.
- DirectSound: para la reproducción y grabación de sonidos de ondas.
- DirectMusic: para la reproducción de pistas musicales compuestas con DirectMusic Producer.
- DirectShow: para reproducir audio y vídeo con transparencia de red.
- DirectSetup: para la instalación de componentes DirectX.

A pesar de ser desarrollado exclusivamente para la plataforma Windows, una implementación de su API se encuentra en progreso para sistemas Unix (en particular Linux) y X Window System conocida como Cedega, desarrollada por la empresa de software Transgaming. Está orientado a la ejecución de juegos desarrollados para Windows bajo sistemas Unix.

Irrlicht es un motor 3D (real-time engine) de alto rendimiento en tiempo real y escrito en **C+ +.** Es código abierto y multiplataforma, oficialmente se ejecuta sobre **Windows, Mac OS** y **Linux**.

Hace uso de **D3D, OpenGL** y su propio software de render, tiene todas las características que se pueden encontrar en motores 3D comerciales. Está muy bien documentado e integra muchas características de representación visual como sombras dinámicas, sistemas de partículas, animación y detección de colisiones. Todo esto es accesible a través de una interfaz bien diseñada y de fácil uso en **C++.**

Existe una gran comunidad activa, y hay muchos proyectos en desarrollo que utilizan dicho motor. Por último, indicar que soporta múltiples lenguajes “scripts”.

IrrKlang es una biblioteca de audio diseñado para ser usados en juegos, simulaciones científicas y similares.

Se trata de un motor de audio de alto nivel en 2D y 3D multiplataforma (**Windows**, **Mac OS**, **Linux**). La biblioteca de audio reproduce WAV, MP3, OGG, MOD, XM, IT, S3M y más formatos que pueden ser usados en C++ y todos los lenguajes .NET conocidos (C #, VisualBasic.NET, etc.) Tiene todas las características conocidas para bibliotecas de audio de bajo nivel, así como una gran cantidad de funciones útiles como un motor sofisticado de streaming, lector de audio extensible, modos multihilos, emulación 3D de audio para un hardware de baja gama, un sistema de plugin, etc... Todo esto se puede acceder a través de un API muy simple.

MySQL es un sistema de administración de bases de datos. Una base de datos es una colección estructurada de tablas que contienen datos. Esta puede ser desde una simple lista de compras a un vasto volumen de información en una red corporativa. Para agregar, acceder y procesar datos guardados en un computador, se necesita un administrador como MySQL Server. Dado que los computadores son muy buenos manejando grandes cantidades de información, los administradores de bases de datos juegan un papel central en computación, como aplicaciones independientes o como parte de otras aplicaciones.

MySQL es un sistema de administración relacional de bases de datos. Una base de datos relacional archiva datos en tablas separadas en vez de colocar todos los datos en un gran archivo. Esto permite velocidad y flexibilidad. Las tablas están conectadas por relaciones definidas que hacen posible combinar datos de diferentes tablas sobre pedido.

MySQL es software de fuente abierta. Fuente abierta significa que es posible para cualquier persona usarlo y modificarlo. Cualquier interesado puede estudiar el código fuente y ajustarlo a sus necesidades. MySQL usa el GPL (GNU General Public License) para definir que puede hacer y que no puede hacer con el software en diferentes situaciones.

Entre las características disponibles en las últimas versiones se puede destacar:

- Amplio subconjunto del lenguaje SQL.
- Disponibilidad en gran cantidad de plataformas y sistemas.
- Diferentes opciones de almacenamiento según si se desea velocidad en las operaciones o el mayor número de operaciones disponibles.
- Transacciones y claves foráneas.
- Conectividad segura.
- Replicación.
- Búsqueda e indexación de campos de texto.

C++ es un lenguaje que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos.

Actualmente existe un estándar, denominado ISO C++, al que se han adherido la mayoría de los fabricantes de compiladores más modernos. Las principales características del C++ son las facilidades que proporciona para la programación orientada a objetos y para el uso de plantillas o programación genérica (templates).

Además posee una serie de propiedades difíciles de encontrar en otros lenguajes de alto nivel:

- Posibilidad de redefinir los operadores (sobrecarga de operadores)
- Identificación de tipos en tiempo de ejecución (RTTI)

C++ está considerado por muchos como el lenguaje más potente, debido a que permite trabajar tanto a alto como a bajo nivel, sin embargo es, a su vez, uno de los que menos automatismos tiene (obliga a hacerlo casi todo manualmente al igual que C) lo que "dificulta" mucho su aprendizaje.

En C++, la expresión "C++" significa "incremento de C" y se refiere a que C++ es una extensión de C.

Comparativa de motores 3D

A continuación, se verá algunos motores de especial interés pero tan sólo haremos la comparativa con aquellos que tienen relevancia para nuestro proyecto.

OGRE 3D (acrónimo del inglés Object-Oriented Graphics Rendering Engine) es un motor de renderizado 3D orientado a escenas, escrito en el lenguaje de programación C++.

Sus librerías evitan la dificultad de la utilización de capas inferiores de librerías gráficas como OpenGL y Direct3D, y además, proveen una interfaz basada en objetos del mundo y otras clases de alto nivel.

El motor es software libre, licenciado bajo LGPL y con una comunidad muy activa. Este motor ha sido utilizado en algunos juegos comerciales.

Irrlicht es un motor 3D (real-time engine) de alto rendimiento en tiempo real y escrito en **C+ +.** Es código abierto y multiplataforma, oficialmente se ejecuta sobre **Windows, Mac OS** y **Linux**.

Hace uso de **D3D**, **OpenGL** y su propio software de render, tiene todas las características que se pueden encontrar en motores 3D comerciales. Está muy bien documentado e integra muchas características de representación visual como sombras dinámicas, sistemas de partículas, animación y detección de colisiones. Todo esto es accesible a través de una interfaz bien diseñada y de fácil uso en **C++.**

Existe una gran comunidad activa, y hay muchos proyectos en desarrollo que utilizan dicho motor. Por último, indicar que soporta múltiples lenguajes "scripts".

Crystal Space es un framework para el desarrollo de aplicaciones 3D. Está programado en C++ usando un diseño orientado a objetos. Se usa típicamente como motor de juego pero el framework es más general y puede ser usado para cualquier tipo de visualización 3D. Crystal Space es muy portable y se ejecuta en Microsoft Windows, Linux, UNIX, y Mac OS X. El código es abierto y licenciado bajo LGPL.

Puede usar opcionalmente OpenGL (en todas las plataformas), SDL (en todas las plataformas), X11 (Unix o GNU/Linux). También puede usar rutinas de ensamblador usando NASM y MMX.

Nebula Device es un motor de juego 3D en tiempo real. El código está abierto y escrito en C++. Incorpora un moderno motor de renderizado que hace uso pleno de shaders, programable, a través de lenguajes scripts como TCL, LUA, PYTHON, RUBY. En la actualidad soporta DirectX 9, mientras que OpenGL aún está en fase de construcción. Funciona en Windows, y se está integrando también para Linux y MAC OS.

Análisis de los motores de juegos

Partimos con Crystal Space y Nebula Device 2. Una buena opción para el desarrollo de un videojuego es utilizar un motor de juego ya que tendremos todos los aspectos necesarios integrados en el mismo motor. No obstante, no se trata de una razón determinante puesto que los motores gráficos 3D mencionados son flexibles, es decir, están construidos de tal modo que pueden incorporar código ajeno, para la física, el sonido, la red… Obviamente al ser código abierto se puede integrar código propio así como hacer las modificaciones que uno crea conveniente.

NEBULA DEVICE 2

Nebula2 destaca por su potente sistema jerárquico de nodos que permite organizar los objetos en escala. Cabe destacar, su particular sistema de punteros y, por supuesto, la facilidad de integración de los scripts en el código. Sería una buena opción ya que la forma en que está construida facilita el desarrollo de juegos con patrones de diseño. Sin embargo, es un SDK (software develop kit) complejo de entender que dispone de escasa ayuda y documentación.

CRYSTAL SPACE

Crystal Space, es la mejor opción entre los motores de juegos. Está muy enfocado a la creación de juegos y es la solución más rápida para generar uno, pero sin compromiso de calidad. Esto es debido a que su desarrollo se ha basado en optimizaciones para que el programador no tenga muchas preocupaciones en la etapa de implementación. Personalmente no lo encuentro como una opción atractiva aunque desde luego es la más fiable para obtener un juego.

CONCLUSIÓN DEL APARTADO

Se ha explicado dos motores de juego que cabrían destacar. Sin embargo, no se usarán en el proyecto ya que los motores más valorados y los que mayor importancia tienen son los que se citarán a continuación. A parte, ninguno de estos motores de juego podrían dar soporte al tipo de juego que se ha propuesto, puesto que Nebula2 no dispone de los lectores de mapas ni de modelos que se requieren y por parte de Crystal Space solo disponen de conversores de mapas y modelos a su propio formato.

Análisis de los motores gráficos

Los dos siguientes motores gráficos 3D, se trata, según los expertos, de los motores más competentes pudiendo ser incluso mejor que algunas comerciales.
Hay que destacar las siguientes propiedades que ambos motores tienen en común y que las hacen muy interesante para realizar cualquier proyecto.

- Son multiplataforma
- Hacen uso de las ventajas de C++ (single/multiple inheritance, STL, abstract interfaces, templates)
- Basado en el uso de diferentes patrones de diseños de los cuales se pueden destacar (Singleton, Estado, Observador-Observable…)
- Poseen las estructuras y los algoritmos más avanzados (Octree, BSP,…) en desarrollo de gráficos
- API, manuales, referencias, tutoriales y ejemplos que facilitan su compresión
- Son robustos, rápidos, flexibles y poseen bastantes propiedades de renderizado. Están muy bien diseñadas y documentadas.

OGRE3D

Sin duda alguna se trata del sistema más completo que cubre muchas características necesarias para el desarrollo de un motor gráfico 3D. El problema es que se trata de un entorno muy grande lo cual dificulta su manejo y su compresión

IRRLICHT

La fuerza de este motor gráfico 3D reside en su facilidad de uso y modificación. Sin embargo, no es tan potente como OGRE3D ya que carece de muchas características que OGRE3D integra. Cabe destacar su robustez, es muy estable y no tiene problemas de colapso “crash” debido a que las características que posee han sido testeadas bastante bien.

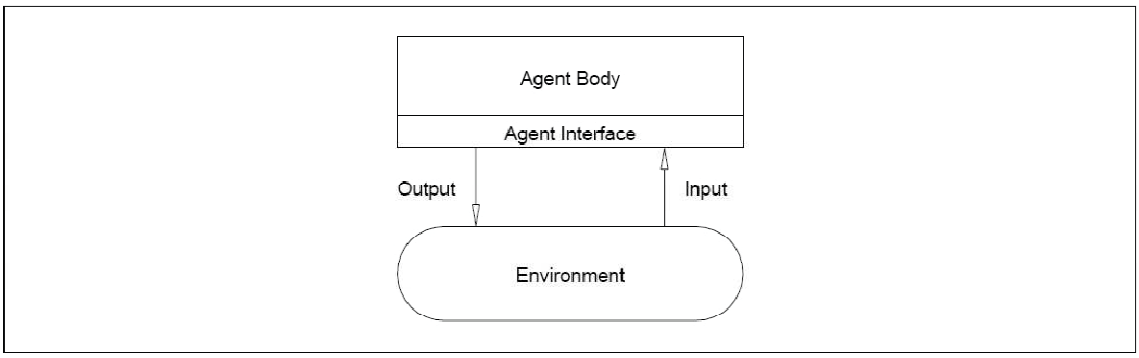
CONCLUSIÓN DEL APARTADO

En general, OGRE3D es el sistema más completo que cubre muchas características, está abierto a muchas posibilidades y es el entorno que presenta menos restricciones para desarrollar cualquier tipo de proyecto.
La razón principal para decantarse hacia IRRLICHT es que cubre todo lo necesario para poder desarrollar el proyecto gracias a su especialización en el tipo de juego propuesto. A parte de esto, es inevitable que para la consecución de un proyecto de esta envergadura será necesaria una compresión del código tanto como si yo lo hubiese desarrollado y en este aspecto Irrlicht ayudará más a la que quizá sea la labor más tediosa.

Inteligencia Artificial

En la última década la inteligencia artificial ha prestado mucha atención a los sistemas multiagentes, probablemente debido a las propiedades de los agentes, definidos como una entidad autónoma capaz de interaccionar con un entorno. La cuestión que se debe resolver es ¿cómo podemos diseñar e implementar una entidad capaz de actuar por sí mismo en un determinado entorno?

Esta pregunta envuelve a la investigación de tres componentes: un entorno, un número de agentes y las interfaces usadas por los agentes para interactuar en el mundo. Desde que existen entornos implementados para juegos de computadores, nos aproximaremos a ellos para usarlo como plataforma de simulación. En concreto, usaremos la interfaz que nos ofrece el juego que vamos a construir. De este modo, QuakeGarage nos facilitará la implementación de los agentes.



QuakeGarage

A continuación se describirán las tres componentes básicas explicadas anteriormente desde el punto de vista del QuakeGarage

El entorno: Está constituido por un mapa en tres dimensiones con entidades tangibles. Los agentes interactuando con el entorno son ejemplos de entidades que se mueven, otras están fijas en un determinado lugar del mapa. La simulación del estado del entorno puede ser leída y también modificada por el agente.
La interfaz: Se encarga de la conectividad, lectura y modificación del estado del mundo. Es un conjunto de funciones recogidas en una API que el agente lo utiliza para interactuar con un entorno simulado.
El agente: El objetivo del QuakeGarage es eliminar tantos agentes como sea posible. Para completar esta meta se requiere habilidad y astucia. Implementar un Bot (Agente) que sea capaz de exhibir este comportamiento es un reto desde la perspectiva de agente autónomo y la inteligencia artificial.

Objetivos

Los objetivos que se pretenden alcanzar en este apartado abarcan algunos de los conceptos de la inteligencia artificial, como puede ser el concepto de multiagentes y todo lo que implica. Sin embargo, también deben ser tratados ciertos temas como pueden ser los algoritmos de búsqueda o la interpretación de datos.

Con todo el conjunto de conceptos que deben ser tratados se requiere de la elaboración de herramientas que permitan construir agentes inteligentes. Para garantizar el correcto funcionamiento de estas herramientas se creará un BOT el cual posea un mínimo de inteligencia y sea capaz de moverse, detectar obstáculos e interpretar una serie de datos para poder abordar determinados objetivos. Este BOT nos servirá como tutorial o modelo de ejemplo para evaluar y conocer el alcance de la API. Recalcar nuevamente que, el juego implementado actuará como API (en C++) y nos será de utilidad para la consecución de este importante apartado.

Marco teórico

En este apartado se tratarán todos los aspectos referentes al marco teórico que envuelve el desarrollo de un BOT para que cumpla unas exigencias mínima, tales como, interpretación de un mapa, buscar entidades etc.

En un primer paso se describirá de forma resumida los problemas que se han abordado, dando consigo una pequeña explicación de aquellos detalles que vamos a tratar. Posteriormente se procederá a explicar las soluciones adoptadas para resolver dichos problemas enfatizando un poco en aquellos aspectos de interés.

Descripción de la API

Antes de entrar con el desarrollo de la Inteligencia Artificial debemos hacer hincapié en nuestra API, ¿Qué funciones están disponibles? ¿Qué permiten obtener?

Como sabemos el juego, QuakeGarage, hace de API, es decir, como medio de comunicación para que nuestro agente autónomo interaccione con el entorno. Se tratan de unas funciones interfaces que se usan para obtener información de nuestro entorno (para mayor información, toda la teoría del entorno se encuentra en el apéndice del documento). Además, el juego dispone de unas librerías exclusivas para el desarrollo del BOT que facilitan la detección de obstáculos y colisiones en general, algoritmos de búsqueda para alcanzar determinadas entidades y generación de mapas para los algoritmos de búsqueda. Un ejemplo similar de API lo podemos encontrar en el QASE para el Quake2.

Descripción del problema

El entorno de trabajo que se seleccionó para desarrollar la Inteligencia Artificial del BOT es una API en C++.

Lo primero que se hizo, antes de empezar a tomar decisiones de implementación, fue ver las posibilidades y limitaciones que tenia nuestro entorno de trabajo. Obviamente esto también nos ayudó para entrar en contacto y analizar con detalle cuáles eran nuestras posibilidades para lograr un BOT competitivo.

La situación no se planteaba fácil. Teniendo en cuenta las limitaciones que encontramos durante el desarrollo del juego QuakeGarage, debido al bajo rendimiento ofrecido por el motor, era y es de esperar que no se pudiera ofrecer un BOT muy complejo.

Después de esta breve introducción, discutiremos acerca de los temas que abordaremos en detalle:

- La manera en que nuestro BOT debía percibir el entorno y la forma más adecuada para moverse a través de él.
- A la misma vez que aprendemos de nuestro mundo, ¿debemos interactuar con nuestros enemigos?
- El tamaño que debía adoptar nuestro mapa es otro de los puntos clave.
- En cuanto a la forma de construir nuestro mapa, nos preocupaba, sobre todo, que este pudiera ser lo más simple posible y se cuestionó lo que es realmente

necesario (puntos por los que movemos, detección de muros, emplazamiento de entidades…)

- ¿Debíamos hacer caso a la lista de entidades que nos ofrecía el mundo, aún sabiendo que algunos de ellos serán inalcanzable?
- Otro aspecto esencial fue como lograr ir de un sitio a otro para lograr nuestros objetivos.
- En cuanto a nuestra forma de combatir, como elegir las acciones más deseables, es decir, que armas seleccionar y como apuntar a nuestro enemigo.

Como es de esperar estas preguntas fueron surgiendo conforme se avanzaba en el proyecto. Por ahora simplemente indicaremos, que conscientes de la dificultad que emprende todo este tema y las limitaciones del entorno, siempre se pensó en la manera más simple para resolver los problemas. No hay que entenderlo como buscar una solución cualquiera, sino más bien, como buscar una forma que resuelva el problema y que afecte lo menos posible en el rendimiento.

Discusión de la solución a nuestro problema

Tal y como se ha planteado la visión del trabajo, lo que se busca es eficiencia y simplificación para abordar punto a punto todos los problemas mencionados.

Para empezar, se hizo una rápida decisión acerca de cómo hay que interactuar con el mundo, se separo el aprendizaje y el modo de combate. Es imprescindible tener un mínimo conocimiento del entorno antes de entrar en un modo de lucha, esto implica que debe haber un agente que se encargue de facilitar esta tarea.

Por otro lado, ¿por qué no incluir en el modo de combate también al aprendizaje? En principio no hay ningún motivo para no hacerlo, de hecho hasta nos podría favorecer mejorando nuestro conocimiento del medio. No se ha incorporado pero es seguro que dicha ampliación mejoraría el comportamiento del agente.

Definiendo el mapa

Para obtener un conocimiento del mundo, como localizar una determinada entidad, moverse a un punto cualquiera, o que pasos debo seguir para hallar el enemigo, parece necesario tener una representación abstracta del mundo en forma de mapa.

Un mapa no puede ocupar mucho espacio en memoria y también debe estar construido de acuerdo con la capacidad de movimiento del Agente, es decir, cuantas posiciones del mundo físico puede avanzar en cada frame.

Se define un conjunto de puntos conectados por los que se puede pasar. No tendremos en cuenta las paredes ya que por esos lugares sabemos que no se puede pasar y, además se incorporarán una serie de tests para garantizar que la conectividad entre puntos no tienen obstáculos de por medio, con lo que conseguiremos que los caminos trazados sean fiables.

Otra posibilidad es hacer un tratamiento de las paredes (esto conlleva muchos problemas). Para tener información exacta de donde se ubican todas las paredes, se podría apuntar como pared, aquellos obstáculos que nos ha retenido, hay que hacer comprobaciones que garanticen que no se trata de alguien que se cruza con nosotros, un pequeño bloque u otras posibilidades que se contemplen. Esta solución es una tarea muy tediosa, que daría información incompleta porque sólo se dispondría de aquellas posiciones en el que Bot se chocó. Además tendríamos que fiarnos en todo momento de la información que nos indique el localizador de obstáculos (nada recomendable).

Información del entorno

La API nos proporciona un conjunto de funciones, mediante las cuales podemos detectar aquellas entidades próximas a nuestra localización actual. Por supuesto, y tal como se planteó en la pregunta, alguna de estas entidades, no será posible alcanzarlas por nuestro BOT.

Antes de seguir con la cuestión, debemos describir nuestra situación. Cualquier Agente explorador del mapa que este prácticamente encima de una entidad, considerará que es alcanzable e inmediatamente lo añadirá a la lista de objetos.

Esta información es útil para el Agente que actuará en modo combate, ya que se trata de información del entorno, que en algún momento dado podría ser necesario.

Movimiento del Agente

Se ha descrito como el agente percibirá la información que el mundo le proporciona, pero falta algo esencial, ¿cómo nos desplazaremos por nuestro entorno con el fin de cubrir el mayor porcentaje posible del mundo? Sólo para este propósito, al cual se le ha dado mucha importancia, se ha elaborado un conjunto de comportamientos que describen el movimiento del BOT:

- Estacionamiento
- Dirección hacia una entidad
- Suavizado
- Acantilado
- Movimientos pisando tierra:
  - Grados de libertad, sentido y velocidad de movimiento
  - Control de choque.
- Movimientos cuando no estamos en tierra

Por supuesto, esto es ampliable y es un aspecto de gran importancia. Otro movimiento que se podría añadir y es muy interesante es la posibilidad de esquivar los disparos del oponente. También se puede mejorar el comportamiento actual.

Algoritmos de búsqueda

Sabemos que hay muchas maneras de hacer algoritmos de búsqueda, pero lo que realmente es vital, es que el tiempo de ejecución sea el menor posible y los resultados buenos. La solución que se adoptó mantiene un compromiso en lo dicho gracias a una posible clasificación de las entidades que se han coleccionado en el aprendizaje y unos mecanismos de agilización del algoritmo de búsqueda (las podas).

Modo de combate

Tal y como se había decidido, el modo de aprendizaje y el de combate se separan, así que por ahora sólo nos centraremos en las estrategias de combates que mejoren la jugabilidad del Agente. Aquí no hubo muchas discusiones ya que no se podía complicar el trabajo y se optó por las soluciones más inmediatas. Estas ofrecen unos resultados aceptables pese a ser muy generales y no estar especializadas en ninguna situación en concreto.

Las decisiones acerca del objetivo que debemos seguir se realizo de una forma jerárquica que se establece según la importancia que consideramos oportuna. De este sistema poco hay que discutir ya que es lo más lógico a realizar. Posiblemente lo realmente determinante, sean los umbrales que se fijan para pasar de un nivel de la jerarquía a otra. Estas se aplican según la experiencia con el juego.

La otra clave del modo de combate es el ataque. Podemos resumir su importancia en tres actividades:

- Selección del arma: Se adoptó la forma más simple, que es seleccionar aquella que creemos que es la mejor arma. En realidad una buena base sería la de elegir el arma dependiendo de la distancia del objetivo y también del estado del enemigo, por ejemplo, si está agachado (escondido) o está a campo abierto.

- Selección del oponente: La estrategia es elegir al oponente más cercano. En términos generales es una opción muy válida, lo que pasa es que en juegos donde hay muchos rivales, quizá no interesa estar eligiendo al más cercano, sino fijar a uno hasta que nos maten o acabemos con este enemigo.
- Apuntar al oponente: Fijar el puntero en nuestro contrincante. Por supuesto esto es bueno para armas en las que el alcance es inmediato, sin embargo, para los que tardan más, lo mejor es hacer una predicción de movimiento del enemigo. Simplemente comprobando cuál fue su último movimiento realizado, ya tenemos un mecanismo de predicción.

Para finalizar con este modo, recordar una vez más, que aquí pueden ocurrir muchas situaciones lo cual hace complejo tratarlas todas.

Marco práctico

Una vez explicado todos los aspectos teóricos, se explicarán a continuación cada uno de ellos, entrando en algunas ocasiones en pequeños detalles de implementación.

Desarrollo de la práctica

En este apartado se desarrollará cada una de las partes que conforman el comportamiento del BOT. Disponemos de dos tipos distintos de BOTS, el explorador y el asesino, cada uno con una funcionalidad bien definida.
Como aspecto esencial de comunicación entre dichos Bots, tenemos una clase que permite cambiar la conducta del Bot, donde además, capturaremos toda la información recogida del explorador para que el asesino la pueda interpretar.

Exploración

Se desarrolló un BOT para capturar toda la información útil que va encontrando en el mundo. A medida que el BOT se desplaza por el “mundo” se construye un mapa propio. Se trata de un grafo de conexiones direccionales almacenado en forma de lista, no sólo se guardará en la base de datos, sino también en nuestra clase para poder comprobar que los nuevos datos no han sido insertados previamente.

Un aspecto muy importante para este BOT, es la capacidad de movimiento que disponga, ya que influirá directamente en la eficiencia con la que logrará sus objetivos y los caminos alternativos que será capaz de generar para poder llegar a una situación en particular.

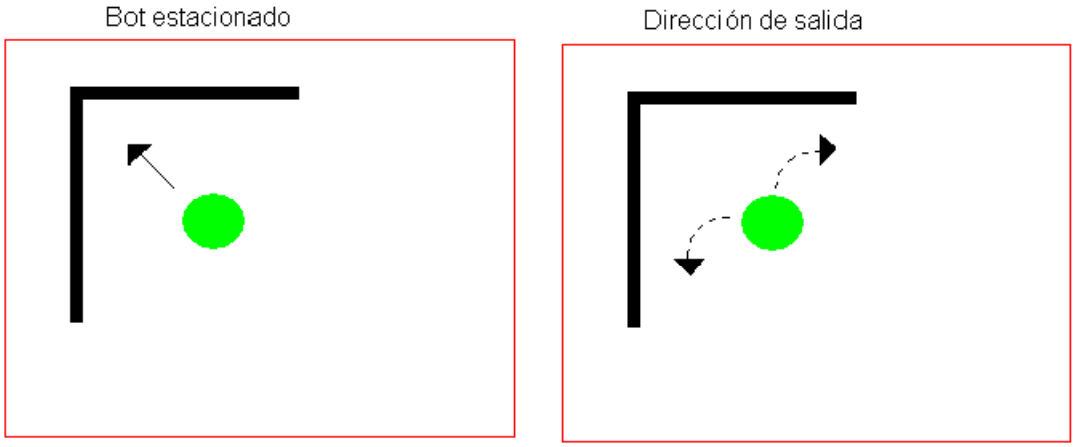
Las posibilidades de movimiento vienen dado por los grados de libertad que se les ha ofrecido, así como también por la aleatoriedad en los giros. Este paso del desarrollo del BOT ha llevado bastante tiempo hasta el punto de obtener un comportamiento que se consideró razonable.

Movimiento

Desde el punto de vista del código hay varios factores que determinan su comportamiento. Las funciones que se explican a continuación se basan en el uso de un mecanismo para saber en qué dirección nos podremos mover.

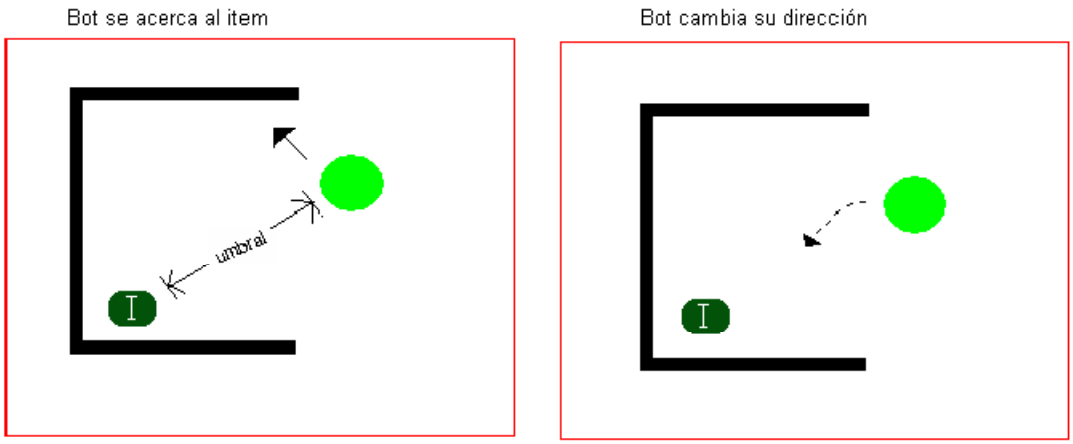
Estacionamiento

Si nuestra posición no varía durante unos instantes, giramos bastante en un sentido dado.



Ítem cercano

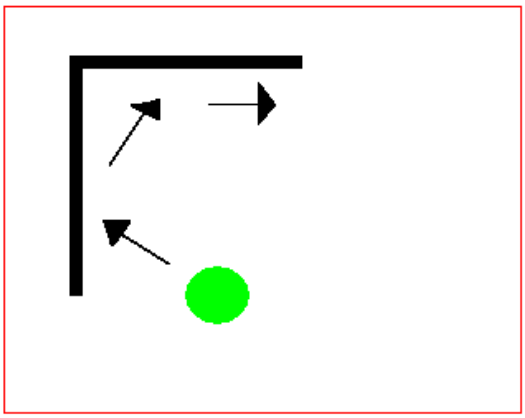
Si dentro de un radio determinado a nuestra posición nos topamos con un ítem que es visible desde nuestro punto de vista nos dirigiremos hacia él.



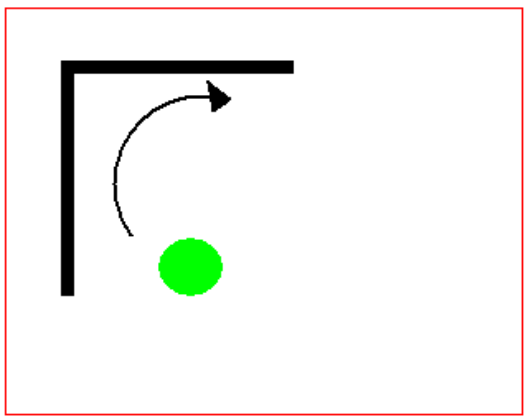
Suavizado

Para simular un comportamiento realista en el movimiento, se intentó hacer giros suaves para evitar los obstáculos. La idea introducida es bastante sencilla, lo que no se puede decir lo mismo para el código desarrollado. Cuando nos encontramos a una distancia bastante próxima a un obstáculo miramos hacia nuestros lados para saber por donde debemos avanzar.

Bot sin movimiento suavizado

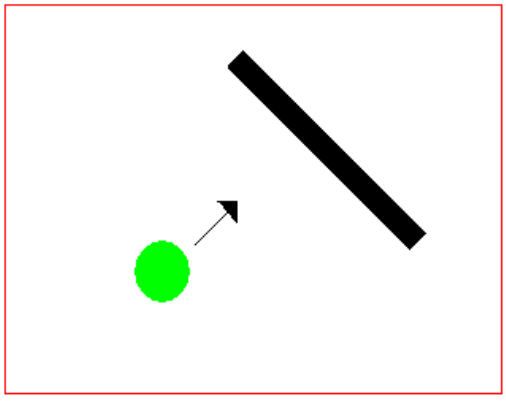


Bot con movimiento suavizado

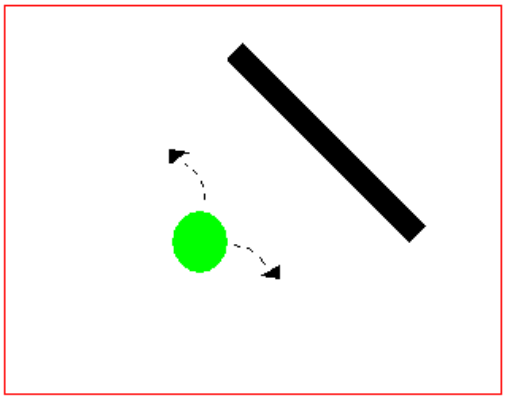


En el código lo primero que se fija un lado aleatoriamente para chequearlo. A continuación, se gira hasta un máximo de noventa grados, se trata de un bucle donde vamos calculando para cada ángulo la distancia al obstáculo, si existe algún ángulo que supere el umbral de aproximación a un obstáculo, entonces marcaremos esta dirección como nuevo frente de avance. En caso contrario, procederemos de la misma manera pero con el sentido contrario. Finalmente, si ningún lado supera el umbral giraremos con el máximo grado permitido.

Bot se acerca a un muro

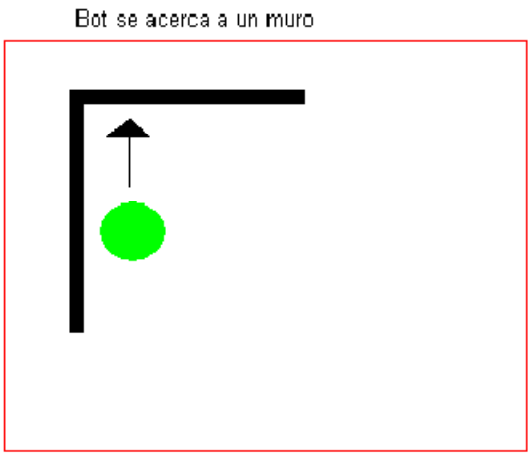


Bot elige aleatoriamente una salida



Este simple mecanismo consigue que a medida que nos vamos aproximando a un obstáculo, el BOT vaya girando levemente hasta dar con una salida. Un detalle de este método es que el BOT tiende a mantener el sentido de giro. Esto se debe a que a medida que vas girando en un

sentido, para esquivar el obstáculo, el otro sentido pierde posibilidades de realizarse, ya que cada vez será más difícil que algún ángulo en el sentido contrario supere el umbral.



Acantilado

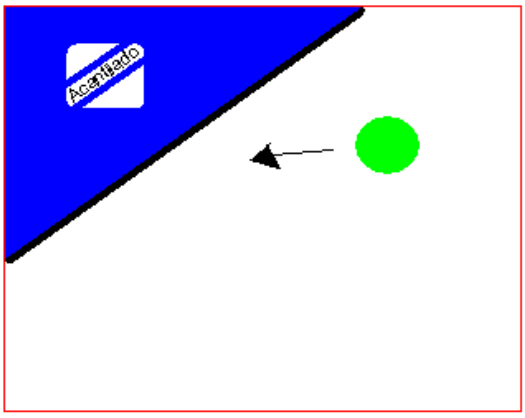
¿Cómo podemos estar seguros que nos encontramos ante un acantilado? Para poder resolver esta pregunta se propuso una estrategia, se trata de trazar un rayo hacia el suelo, obteniendo a qué distancia del suelo se encuentra nuestro BOT. De esta forma, nos podemos hacer una idea de si estamos ante un acantilado o en el suelo.

La situación no es tan sencilla, porque hay que considerar que podemos estar ante una rampa, para la cual, si no tenemos cuidado, el BOT se podría pensar que se trata de un acantilado. Es por ello que debemos ajustar el ángulo para lanzar un rayo al suelo, a mayor ángulo, mayor visión de lo que tenemos adelante y viceversa.

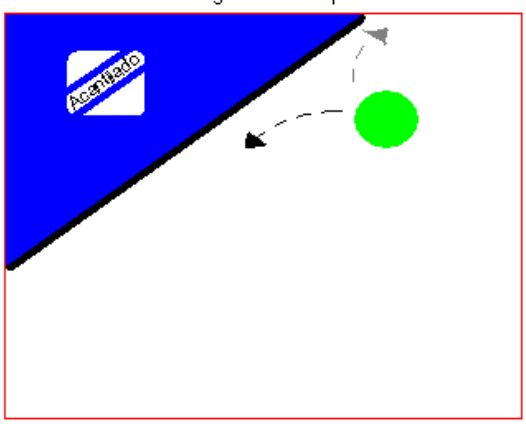
La parte anteriormente comentada es obviamente la más importante. Una vez detectado que estamos ante un acantilado, debemos saber cuánto hemos de girar para poder salir del paso. Como queda reflejado, estamos ante la misma situación que el “Suavizado” así que aprovecharemos el método y lo aplicaremos.

A nivel de código lo que se propuso fue un bucle donde vamos calculando para cada ángulo un rayo hacia el suelo, y con esto, se espera saber si supera el umbral de detección de acantilado. El efecto obtenido es, como cabe esperar, el mismo que para el “Suavizado”. Se va girando levemente hasta salir del acantilado a menos que haya que girar bruscamente debido a que estemos cerca de la esquina de un precipicio.

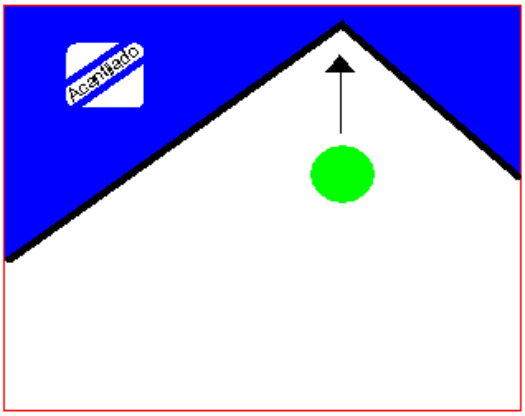
Bot se acerca a un acantilado



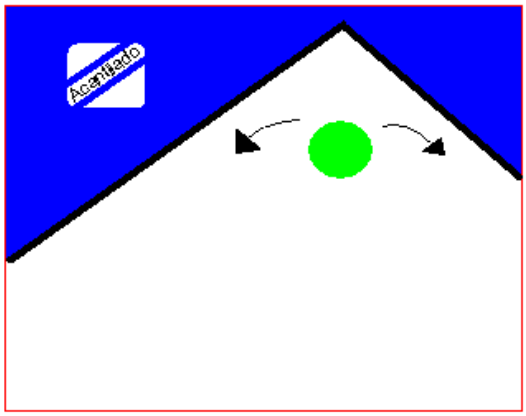
Bot tenderá a girar a su izquierda



Bot se acerca a la esquina de un acantilado



Si no encuentra salida hace un gran giro



Un problema que se observó es que cuando avanzamos hacia un acantilado y además avanzamos hacia una pared, el Bot preferirá evitar la pared en lugar del acantilado. Este problema lo aprovechamos como una solución para que de vez en cuando el BOT se lance por algún sitio. Para este mismo propósito, se aprovechó que en cada ciclo, no siempre se hace la comprobación del acantilado.

Movimientos pisando tierra

Dentro de este apartado encontramos una serie de comportamientos, cada unos especializado en un aspecto determinado. En la realidad se tratan de pequeños comportamientos que son emergentes en determinadas ocasiones. Su tratamiento no siempre será apreciable, sin embargo su control mejora notablemente el movimiento del BOT.

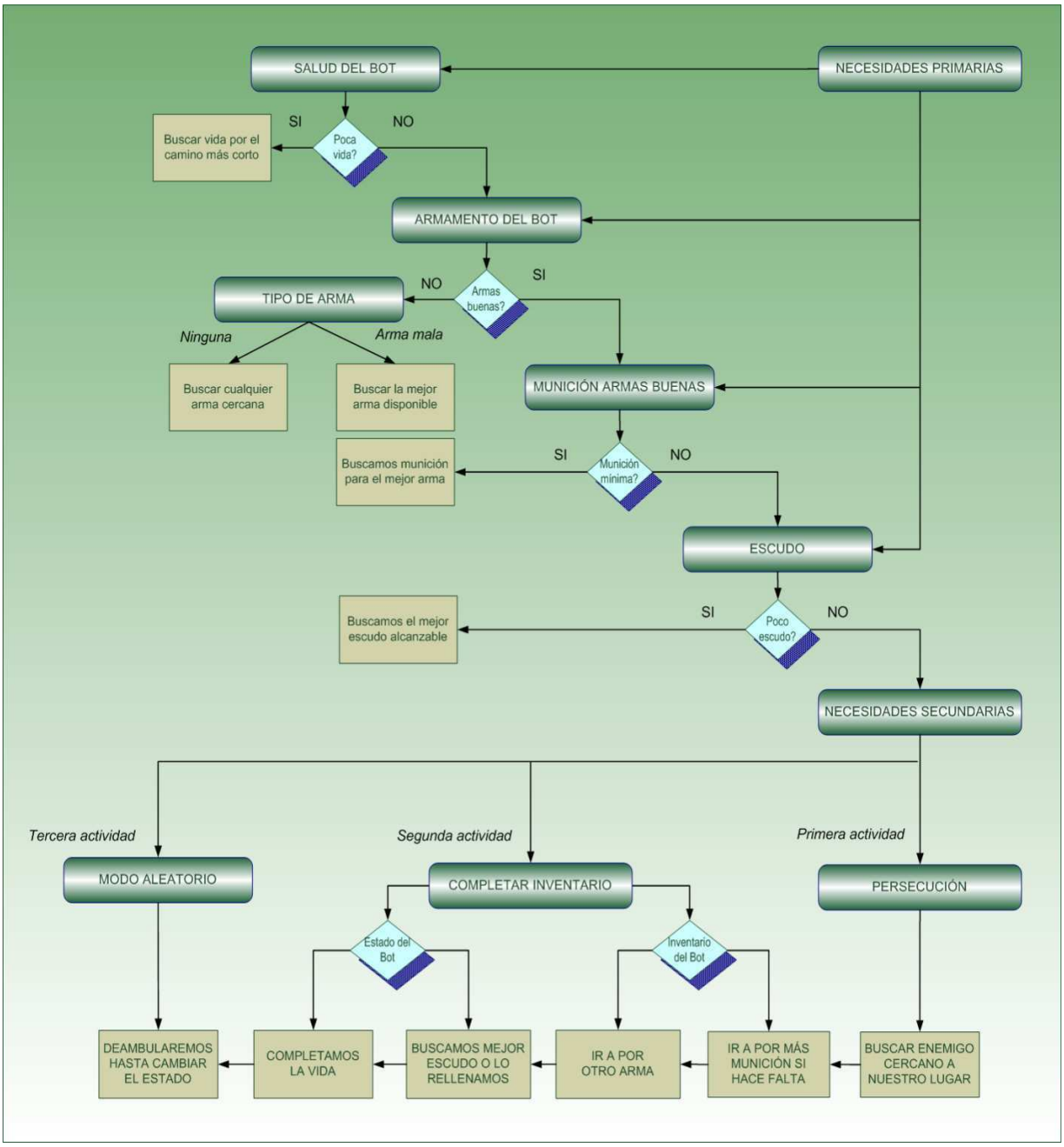
- Grados de libertad, sentido y velocidad de movimiento:

Para determinar los grados de libertad se usa una función aleatoria y un factor máximo de grado de libertad “g”. El sentido siempre vendrá determinado por la dirección que se obtenga al salir de una curva o un acantilado. Por último, la velocidad siempre estará fijada a un único valor para que en todo momento el BOT esté corriendo.

Estrategias

Se definió de forma jerárquica y abarca desde las necesidades más básicas hasta las necesidades complementarias, como pueden ser obtener un buen inventario de armas, munición e incluso mejoras del estado del BOT (la vida y el escudo).

Lo que se describirá a continuación, es una serie de contextos, en cada uno de ellos se fijaron un determinado umbral de entrada basándonos en la experiencia con el juego. La intención era poner límites que el BOT debía superar para ser medianamente competitivo. Dentro de la tabla de necesidades se tuvo en cuenta la presencia del adversario, si la notamos, empleamos la búsqueda que tarde menos y si no se aplicará lo que más nos convenga.



Algoritmos de búsqueda

Todo el mecanismo para agilizar la búsqueda que se implementó consiste en una búsqueda en profundidad con dos podas.

La primera poda es totalmente lógica. Hay un hecho en este tipo de búsqueda que se tuvo en cuenta y es que si un punto cualquiera en nuestra búsqueda es alcanzado más de una vez durante nuestra exploración, no vamos a permitir que se siga explorando por ese punto, ya que estaríamos repitiendo el mismo proceso de exploración que ese hizo en ese mismo punto anteriormente, a menos que ese punto se haya alcanzado con una menor distancia que el que lo encontró antes. La segunda poda es más determinante en cuestión de tiempo de realización de la búsqueda. Se trata de ponerle un límite a la distancia de alcance de objetivos.

Empíricamente se probó un valor razonable pero por supuesto aquí influye mucho las características del ordenador donde se ejecuta.

Gráficos por Computador

Un punto muy importante en el desarrollo de un juego es el aspecto visual. En este sentido son diversos los motores 3D que permiten la visualización gráfica de modelos 3D. Un gráfico 3D difiere de uno 2D principalmente por la forma en que ha sido generado. Este tipo de gráficos se origina mediante un proceso de cálculos matemáticos sobre entidades geométricas tridimensionales producidas en un ordenador, y cuyo propósito es conseguir una proyección visual en dos dimensiones para ser mostrada en una pantalla. En general, el arte de los gráficos 3D es similar a la escultura o la fotografía, mientras que el arte de los gráficos 2D es análogo a la pintura.

Fases para la creación de elementos/gráficos 3D:

- MODELADO. Consiste en dar forma a los objetos para integrarlos en la escena
- SHADING/TEXTURIZADO. Se aplican algoritmos que producen efectos sobre los materiales y se combinan con la textura del objeto.
- ILUMINACIÓN. Creación de luces puntuales, direccionales, por áreas, etc.
- ANIMACIÓN. Se hace uso de diversas técnicas para producir movimientos sobre ciertas entidades.
- RENDERIZADO. Esta es la fase donde nos centraremos en el proyecto, todas las anteriores fases en la creación de gráficos 3D se nos proporcionarán debido a que se trata de un trabajo artístico y no informático.

Se llama render o reproducción al proceso final de generar la imagen 2D o animación a partir de la escena creada. Esto puede ser comparado a tomar una foto o en el caso de la animación, a filmar una escena de la vida real. Generalmente se buscan imágenes de calidad fotorrealista, y para este fin se han desarrollado muchos métodos especiales. Las técnicas van desde las más sencillas, como el esquema de alambre (wireframe rendering), pasando por la reproducción basada en polígonos, hasta las técnicas más modernas como el Scanline Rendering, el Raytracing, la radiosidad o el Mapeado de fotones.

Marco teórico

Se procederá a la visualización de un entorno 3D aplicando algunas técnicas de rendering. Se modificará un determinado motor gráfico, a la que aplicaremos técnicas de gráficos por computador, con el objetivo de visualizar un escenario en 3D.

- Familiarización con las DirectX. Librerías con las que el motor gráfico trabaja.
- Modificación del motor gráfico para permitir su acoplamiento con la arquitectura del juego.
- Se realizarán mejoras o ajustes gráficos del motor y si es posible se intentará realizar una propia versión del motor gráfico.

Implementación

Es la parte fundamental y más importante de todo el proyecto donde se definirá el juego, por lo tanto se le dedicará la mayor parte del tiempo.

Se implementará una arquitectura, para la parte específica del juego, mediante el uso de patrones de diseño de ingeniería del software. Un patrón de diseño es una jerarquía de clases que dan solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias. En nuestra aplicación se han utilizado los siguientes patrones de diseño:

- Singleton (Instancia única): Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.
- Command (Orden): Encapsula una operación en un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma.
- Observer (Observador): Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambia de estado, se notifica a todos los objetos que dependen de él, para que se actualicen automáticamente.
- State (Estado): Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno.
- Strategy (Estrategia): Permite disponer de varios métodos para resolver un problema y elegir cuál utilizar en tiempo de ejecución.
- Template Method (Método plantilla): Define una estructura algorítmica como súper clase, y delega su implementación a las subclases. Es decir, define una serie de funciones que serán redefinidas en las subclases.

Las partes que se integrarán en la aplicación son el sonido, la GUI (guide user interface), los gráficos, física y todos los aspectos que participen en el juego. Se destaca cuatro partes desarrolladas independientemente en la arquitectura del videojuego:

- El estado de la aplicación (menú, juego)
- La lógica del juego (jugador, enemigos, reglas del juego, subestados del juego, entidades, colisiones...)
- La GUI es un gestor que se encarga de la pantalla (el dibujado)
- El sonido.

Parte no específica del juego <DLL>

Los módulos que dan soporte al motor del juego y que ya hemos descrito son:

- Cargador de mapas
- Gestión de entrada
- Motor de física
- Motor GUI
- Motor de Red
- Motor de Sonido
- Motor de Base de Datos
- Motor Gráfico

Parte específica del juego: Código <Exe>

Es el código en reproducción, el motor de nuestro juego. Contiene un bucle principal que hace la distribución del juego. La construcción de este bucle ahora tiene mucha más importancia debido a que disponemos de varias CPUs que nos permite distribuir hebras e hilos concurrentemente.

En un juego es imprescindible que la CPU esté al 100% disponible para él en todo momento.

Main

En la clase mainIrrlicht distinguimos dos formas de ejecución:

- La primera consiste en la ejecución de la aplicación con la consola (para el modo debug), de este modo podremos imprimir por pantalla datos que nos interese.
- La segunda consiste en la ejecución de la aplicación sin la consola (para el modo release), esta es la versión que se lanzará al público.

Aplicación

La clase aplicación Irrlicht dispone de un puntero hacia el dispositivo Irrlicht, de aquí podremos acceder a todos los servicios que el motor gráfico nos puede ofrecer. Además, esta clase actualiza el tiempo usando el del motor Irrlicht en lugar de usar el de Windows.

Es una clase que se encarga de ejecutar el bucle principal. Mientras el usuario no haya solicitado terminar la aplicación se llamará al método run() del estado actual de la aplicación (clase EstadoAplicación).

La aplicación de Irrlicht se construye usando la herencia, a partir de la clase Aplicación y la clase EstadoAplicación. Se hace uso del patrón State ya que nos permite definir distintos tipos de aplicaciones con sus correspondientes estados. En la clase mainIrrlicht se puede ver que creamos una instancia de la clase AplicaciónIrrlicht, la ejecutamos y finalmente la destruimos.

Dentro de la AplicaciónIrrlicht hay que destacar lo siguiente:

- Contiene la referencia al dispositivo principal de Irrlicht, y por tanto, todos los servicios de Irrlicht
- Es la responsable de la actualización del tiempo usando el servidor de Irrlicht
- Se encarga de realizar la inicialización y la terminación del motor.
- La aplicación sigue delegando en el estado actual la ejecución del método OnRun.

EstadoIrrlicht

Es una especialización de la clase EstadoAplicacion, de la que debemos destacar lo siguiente:

- Contiene el bucle principal de Irrlicht. El bucle principal está construido usando el patrón de diseño Template (plantilla), que nos permite no tener que copiar y pegar el bucle principal en cada uno de los estados que construyamos, sino que basta con redefinir varios métodos. Estos métodos serán los que nos producirán diferentes comportamientos en los estados. Son tres métodos virtuales que se redefinen en cada clase:

  1. Lectura de la entrada del usuario: procesarEntradaUsuario()
  2. Actualización de la lógica del juego: ejecutaLogica (msecs)
  3. Dibujar la pantalla (GUI): pinta3D (msecs)

A partir de esta clase se definen dos subclases: EstadoMenuIrrlicht y EstadoJuegoIrrlicht. Dentro de la clase EstadoIrrlicht se redefinen tres métodos, pero dos de ellos se dejan vacíos, ejecutaLógica y pintar3D (no hay escena 3D, ni partida, por lo que no es necesario gestionar la cámara). En cambio, se redefine procesarEntradaUsuario para que, dependiendo de la tecla que pulse, se cambie de estado.

EstadoMenuIrrlicht

Es una subclase de EstadoIrrlicht y tiene la responsabilidad de las siguientes tareas:

- Inicializa el gestor del escenario de Irrlicht.
- Ofrece todas las opciones que la aplicación posee.

EstadoMenuIrrlicht se descompone en los siguientes subestados: subestadogamemode, subestadobotmode, subestadosetup, subestadocredit, subestadoexit.

Este estado delega a unos subestados la ejecución de la aplicación. Para poder cambiar de un subestado a otro, hemos creado un patrón Observer de eventos de partidas como mecanismo de comunicación. De este modo, la partida nos comunicará cuando se produce algún cambio de estado, manteniendo el desacoplamiento de la aplicación con la lógica de la partida. La especialización de cada subestado es muy importante, ya que los diferentes eventos que transcurren durante un juego, a veces exigen que el juego pase por un nuevo estado, para cubrir diferentes necesidades. Los subestados son:

- Modalidad Juego. Esta es la modalidad por defecto en el que el usuario podrá jugar contra la CPU. Tendrá la posibilidad de elegir algún mapa que le ofrece el juego.
- Modalidad Bot. Esta modalidad se ha creado para la simulación de combates usando únicamente la CPU (el usuario no participa). La idea es crear y mejorar comportamientos de los BOTs.
- Setup. Agrupa diferente opciones de configuración del juego, el sistema y el jugador. Al igual que la clase EstadoMenuIrrlicht este subestado posee una especialización para cada situación. Se delega a los siguientes subestados la ejecución de la aplicación: subestadomainsetup, subestadoplayersetup, subestadocontrolsetup, subestadosystemsetup, subestadooptionsetup, subestadodefaultsetp.
- Credit. Se exponen los aspectos más importantes que participaron en la creación del juego.
- Exit. Última opción del juego que sirve para finalizar la aplicación.

Los subestados son subclases de SubestadoJuego. Al igual que el resto de los estados, tienen métodos para la inicialización y terminación, activación y desactivación, (inicializar(), terminar() y dibuja()). Estos subestados tienen referencias a la aplicación y el estado. Para activarlos y desactivarlos la clase EstadoMenuIrrlicht llama a los métodos onActivate() y onDeactivacte() de los subestados.

EstadoJuegoIrrlicht

Es una subclase de EstadoIrrlicht y tiene la responsabilidad de las siguientes tareas:

- Es responsable de la creación de la partida, de su inicialización y de su destrucción.
- Es responsable de la creación del GestorGUI y del GestorSound, de su inicialización y de su destrucción
- Es responsable de la gestión de una cámara.

EstadoJuegoIrrlicht se descompone en 3 subestados: subestadojugando, subestadoesperando, subestadomuerto.

Este estado delega a unos subestados la ejecución de la aplicación. Para poder cambiar de un subestado a otro, hemos creado un patrón Observer de eventos de partidas como mecanismo de comunicación. De este modo, la partida nos comunicará cuando se produce algún cambio de estado, manteniendo el desacoplamiento de la aplicación con la lógica de la partida. La especialización de cada subestado es muy importante, ya que los diferentes eventos que transcurren durante un juego, a veces exigen que el juego pase por un nuevo estado, para cubrir diferentes necesidades. Los subestados son:

- Esperando empiece. Es el estado Inicial, su objetivo es dar una única iteración para que el tiempo se estabilice, una vez se haya cargado la partida.
- Jugando. Es el estado en el que se encuentra generalmente la partida, por lo tanto, es el estado más importante.
- Muerto. Ocurre cuando el jugador se ha quedado sin vida. Se espera un evento para hacer reaparecer al jugador en una nueva zona del mapa.

Los subestados mencionados son subclases de SubestadoJuego. Al igual que el resto de los estados, tienen métodos para la inicialización y terminación, activación y desactivación. Así como también los métodos de ejecución de la lógica, pintado3D y de procesamiento de la entrada del usuario, aunque con otros nombres (inicializar(), terminar() y dibuja()). Estos subestados tienen referencias a la aplicación, el estado y a la partida. Para activarlos y desactivarlos la clase EstadoJuegoIrrlicht llama a los métodos onActivate() y onDeactivacte() de los subestados.

Inicialización, Bucle Principal y Finalización Irrlicht

Como se explica en la sección anterior, la inicialización y la destrucción de Irrlicht se realiza en la clase AplicaciónIrrlicht. Antes que nada, hay que indicar la necesidad de inicializar el dispositivo Irrlicht, mediante inicializaIrrlicht(). Dentro de este método cabe destacar lo siguiente:

- Se realiza la inicialización de la ventana de Irrlicht.
- Se inicializa el tiempo con el motor Irrlicht.
- Incluimos al gestor de ficheros Irrlicht la carpeta “media”, donde se localiza todos los archivos del juego.

El BUCLE PRINCIPAL de Irrlicht se encuentra en el método onRun(msecs) de la clase EstadoIrrlicht. El bucle principal tiene que realizar las siguientes tareas:

- Actualización de la variable de tiempo (time), que será usado por Irrlicht para su actualización.

- Llamada al método run() de Irrlicht. Es responsable de la limpieza del dispositivo Irrlicht en cada iteración, para que este pueda reiniciar sus funciones programadas (actualizar el tiempo, almacenar eventos de teclado, ratón, dibujar por pantalla).
- Se llama al método procesarEntradaUsuario().
- Se llama al método ejecutaLógica().
  a. Procesar los comandos pendientes
  b. Atender los eventos (de la partida) pendientes
  c. Actualizar las entidades
- Dibujamos usando los método drawAll() de Irrlicht, tanto para la escena como para el HUD. Antes de hacer el render de la escena se llamará a nuestro método renderEntidades(), que será el responsable de indicar que objetos debemos pintar

La finalización del motor Irrlicht se realiza en el método terminaIrrlicht(). Básicamente se encarga de realizar la destrucción de todos los servicios del dispositivo usando el método drop().

Entrada del usuario

La entrada de usuario se gestiona usando el método procesarEntradaUsuario(), ya sea en las subclases de EstadoIrrlicht o en las subclases de SubEstado. En la gestión de la entrada, se va procesando de manera secuencial los eventos que van llegando. Haciendo uso del patrón de diseño Command, permitiremos el encapsulamiento de la información, que posteriormente será procesado dentro de la lógica

Gestión del HUD

El HUD (Head Up Display) está formado por los elementos en 2D que aparecen sobreimpresionados en la pantalla. Las clases usadas para la construcción de un HUD se encuentran dentro del gestor GUIEnvironment de Irrlicht. Es responsable del renderizado de todos los objetos del HUD y de invocar los métodos asociados a los eventos que se pueden producir.

Cada subestado de la aplicación (Menú y Juego) va a tener su propio HUD. Este HUD se va a almacenar dentro del gestor GUIEnvironment de Irrlicht. El gestor nos proporciona las funciones básicas para su modificación y su visualización (específicamente se trata de cambiar el grado de transparencia de los objetos, haciendo uso del canal alfa).

Lógica

La lógica del juego se encarga de controlar toda la información de la partida que está en transcurso. Son muchas las clases que representa la lógica, aquí explicaremos las principales y como se integran dentro de la arquitectura.

Partida

Es la clase que se encarga de controlar la situación actual de nuestra partida, y centraliza todas las reglas del juego. Su atributo actualizar (msec) es la encargada de procesar la lógica del juego transcurrido cierto tiempo.

Su primera función es procesar los comandos pendientes, para ello dispone de una lista de teclas pulsadas. A continuación, la clase EstadoJuegoIrrlicht leerá las interrupciones del teclado, haciendo uso del patrón de diseño Command. Este patrón encapsula la información que posteriormente será procesado dentro de la clase Partida.

A continuación se procede a actualizar las entidades. En vez de actualizarlas en una simple pasada se ha hecho una subdivisión. Primero, recorremos toda la lista de entidades para actualizarlas, y luego atendemos los eventos pendientes de la partida. Durante dicha actualización es posible que se produzcan nuevos eventos, estas se anotarán en una lista, y serán atendidos en la siguiente iteración del bucle. La razón por la que se ha separado esta tarea en dos pasos, se debe a que muchos eventos modifican la misma cola de entidades que estamos actualizando. Hay que esperar a procesar la cola entera para después poder modificarla. La segunda razón es que facilita la implementación en red, pues desacopla el instante en el que se realiza la invocación y en el que recibe el resultado, sin congelar la aplicación en ningún caso.

2. Actualizar la lógica del juego

    a. Procesar los comandos pendientes
    b. Actualizar las entidades
    c. Atender los eventos (de la partida) pendientes

Muchos de los eventos que son atendidos durante la partida, representan la creación y destrucción de nuevas entidades (clase ObserverCreaciónDestrucción), que deben ser informados al gestor de pantalla GUI. Para ello haremos uso del patrón Observer. Es un patrón, que informa a cualquier objeto, de cualquier evento producido por un objeto determinado. La entidad que produce dicho evento, llama a todas las entidades interesadas en conocerlo. Para ello, disponemos de una lista de observadores asociados al objeto que produce los eventos. Cada vez que se produzca un evento, se emitirá a todos los observadores registrados.

Un segundo tipo de observador (clase ObserverEventosPartida) sirve para que la partida notifique sucesos importantes, como por ejemplo que el jugador ha muerto en el interior del escenario o que la partida ha terminado porque el jugador pulsó "escape".

Entidad

Es la clase que representa cualquier objeto interactivo en el juego. En este juego se actualizan todas las entidades en cada instante. Sin embargo, en juegos más complejos las entidades se actualizan según la situación del jugador.

La clase dispone de un método actualizar(mseg) que se ejecuta en cada iteración del BUCLE PRINCIPAL y refresca la información de la entidad que esté representando. Esta clase permite que se registren "observers" para ser notificados cuando se produce algún cambio en el estado de la entidad.

Como ya sabemos, de esta clase derivan todas las entidades del juego que son armaduras, municiones, armas, ítems, vidas, avatares (jugadores y enemigos). También se incluyen entidades dinámicas como los disparos y los efectos que producen.

Avatar

Representa cualquier tipo de jugador (usuario o NPC). Contiene toda la información que es común a cualquier personaje en el juego. Concretamente, se especializa en todos los posibles estados comunes a los personajes. En cada instante del juego esta clase se encargará de regular al Avatar según el estado en el que se encuentra, algunos de los estados son: quieto, corriendo, cayendo, saltando, disparando, etc…

A partir de esta clase, se generan subclases que especializan el comportamiento del avatar (subclases Jugador y Enemigo). Por un lado, existirá una clase que representa al avatar del usuario (Jugador), este recibirá la entrada del usuario. Por otro lado, existirá una clase que represente al enemigo (Enemigo), donde se desarrollará la Inteligencia Artificial. Además cada una de las subclases debe implementar las reglas del juego en lo que se refiere a acciones permitidas. Es decir, en el método de actualizar() han de realizar dos cosas:

- Actualizar el avatar en función de su estado.
- Asegurarse de que la actualización anterior cumple las reglas del juego, es decir no se atraviesan paredes, si el jugador es eliminado debe morir, etc…

Habitualmente cada avatar tendrá una representación visual. Cuando el estado del avatar cambia, en un entorno en tres dimensiones, es importante que la representación gráfica se entere de los cambios en el movimiento. Es decir, si el avatar empieza a saltar, andar o cualquier nuevo estado, será necesario un cambio de animación. Cada tipo de animación, por comodidad, se notificará de forma independiente.

Estas operaciones son importantes, en lo que se refiere a la lógica general del juego, y tendrá que notificarlo de algún modo al resto del mundo. Para ello, haremos uso de los Observers de los avatares. Normalmente a cada avatar le corresponde una entidad gráfica (clase EntidadAnim) que implementa esta interfaz. Cuando el avatar cambie de animación, por ejemplo, su movimiento empieza o finaliza, se generará un aviso a través del observer.

¿Por qué separamos las clases Enemigo y Jugador? Es obvio que ambos son avatares que comparten las mismas características, sin embargo cada uno se actualiza de diferente manera. La clase Partida delega a ambas clases, al igual que el resto de las entidades, la responsabilidad de actualización.

Jugador

Entidad lógica que representa al usuario. Para mover al jugador hay que utilizar los métodos declarados. El jugador es el responsable de controlar su propio estado, dentro del método actualizar(). Los elementos externos al jugador pueden ser informados de los cambios de su estado, como la cantidad de vida, escudo o munición mediante un observer. Se trata de una clase abstracta (interfaz) de la que deben heredar todas aquellas clases que quieran ser notificadas.

Enemigo

Entidad lógica que representa al enemigo. La IA de los enemigos está implementada utilizando el patrón strategy, que puede derivar en el patrón State, si la clase Enemigo implementa una máquina de estados. Al igual que en la clase Jugador, esta clase es el responsable de controlar su propio estado. Los elementos externos al enemigo pueden ser informados del cambio de su estado, en la cantidad de vida, escudo o munición, mediante un observer.

Existe un puntero a una clase ComportamientoEnemigo, que es llamado en cada tick lógico, a través del método actualizar(msecs). La clase ComportamientoEnemigo dispone de todas las funciones necesarias para que sus clases hijas puedan dotar al Bot cualquier tipo de comportamiento. De esta clase derivan los siguientes comportamientos:

- Comportamiento Explorar, tiene como misión mover al BOT por todo el escenario para construir un mapa lógico.
- Comportamiento Perseguir, tiene como misión perseguir al adversario si está cerca de su posición.
- Comportamiento Buscar, tiene como misión buscar entidades por todo el escenario.
- Comportamiento Patrullar, se mueve aleatoriamente por el escenario.

Algunas de estas clases hacen uso del servicio de búsqueda que se ha implementado para la Inteligencia Artificial. Se trata de la clase Busqueda, que como su propio nombre indica, está especializada en algoritmos de búsqueda para la I.A. Se implementó con el patrón de diseño singleton.

Escenario

Es la clase que nos define el escenario en el que estamos jugando, indica donde se ubica cada entidad y localiza determinadas infraestructuras.

Colisiones

La decisión de emplazar el sistema de colisiones, en nuestra arquitectura, está muy abierta. Por lo general, se suele aplicar en la parte lógica.

La clase MotorColisiones se encarga de calcular las colisiones que se producen entre las entidades físicas del juego y el escenario. Se creó con el patrón singleton, con lo cual sólo puede existir un único objeto de esta clase, dentro de la aplicación.

Se dispone de un motor de colisiones que es capaz de decidir que entidades registradas están colisionando y con qué. Además si dentro de la arquitectura hubiese un motor físico, este indicaría como debe reaccionar ante la colisión.

Debido a la independencia entre la GUI y el motor de colisiones puede suceder que a veces no se corresponda lo que uno indica a lo que se representa por pantalla.

GUI

Gestión de la Escena 3D

Disponemos de un gestor de la interfaz gráfica que será responsable de la creación y dibujado de los objetos 3D (clase GestorGUI). El gestor se encuentra en la clase EstadoJuegoIrrlicht y será usado por el método pintar() de los distintos subestados del juego. Este gestor diferencia entre el dibujado de la escena (escenario, suelo, cielo) del dibujado de las entidades dependientes de la lógica. De esta forma se pueden dibujar ambos independientemente. Es decir, separamos la lógica de la GUI para evitar poner un proceso pintar() para cada clase dentro de la lógica. De este modo, hay mayor modularidad en nuestro programa. Cualquier cambio realizado en nuestro espacio GUI no afectará en la lógica y viceversa.

La traducción del escenario lógico al escenario en 3D la realiza la clase GestorGUI haciendo uso de la clase BuilderEscenario. Esta clase construye la malla del escenario, y la almacena en las estructuras de datos de Irrlicht. De esta manera, ya disponemos de la información necesaria para el renderizado y detección de colisiones del escenario.

GestorGUI

La clase guarda todas las representaciones gráficas de las entidades lógicas en la clase EntidadesGUI. Esta clase dispone de un contenedor de Entidades que pueden ser de dos subclases, de acuerdo con la naturaleza de la entidad Irrlicht usada para renderizarla:

- EntidadEstaticaGUI se encarga de renderizar entidades no animadas.
- EntidadAnimGUI se encarga de renderizar entidades animadas. Incluye métodos para establecer una determinada animación.

EntidadGUI

La clase se encarga de pintar las entidades representadas. Disponemos de un conjunto de objetos pertenecientes a la clase EntidadGUI y que se van a dibujar en pantalla, entonces ¿Cómo notificamos los cambios de estas entidades? Disponemos de varias alternativas:

1. Disponemos de un puntero de la entidadGUI representada en la lógica y sondeamos en cada Frame su posición. Es una solución ineficiente, ya que es absurdo estar preguntando siempre cuando es posible que ni siquiera haya ningún cambio.

2. La lógica informa al GUI cuando se haya producido alguna variación de la entidad. Sin embargo, en esta solución requiere de duplicación de información en la lógica. Además, cada vez que informemos de un cambio a una entidadGUI estaremos usando en la lógica un puntero que pertenece a otro a la gui y, por tanto, acoplando los módulos.

3. La mejor solución es usar el patrón Observer. Como ya sabemos, el patrón informa a cualquier objeto de cualquier evento producido por un determinado objeto. La entidad que produce dicho evento llama a todas las entidades interesadas en conocerlo. Para ello disponemos de una lista observadores asociado al objeto que los produce los eventos. Cada vez que se produzca un evento se emitirá a todos los observadores registrados dicho evento.

Observers

Las clases GUI que hacen uso del servidor gráfico Irrlicht implementan distintos observadores. Los más importantes son los siguientes:

- El GestorGUI implementa la interfaz del observer de creación y destrucción de la partida para construir la escena y añadir las correspondientes entidades a la clase EntidadesGUI.

- EntidadAnimGUI implementa la interfaz del observer del avatar. Los métodos de esta interfaz se emplean fundamentalmente para realizar cambios de animaciones.
- EntidadGUI implementa la interfaz del observador de la clase Entidad. Se usa para cambiar la posición y la visibilidad de las entidades (tanto estáticas como animadas).

Gestión de la cámara

Como ya se indicó, la clase EstadoJuegoIrrlicht contiene la cámara. Una cámara no es más que una matriz con información sobre su posición, hacia dónde está mirando y cuál es su orientación. Esta matriz es usada por el motor Irrlicht para renderizar la escena. Si se desea modificar dicha matriz hacemos usos de funciones proporcionadas por Irrlicht. Estas se encargan de cambiar la posición de la cámara de acuerdo a los movimientos del ratón y teniendo en cuenta cuál de los botones está pulsado. Con ello, el usuario puede hacer zoom y cambiar hacia donde está mirando. La actualización de este gestor se realiza en dos etapas:

- Primero se procesa la entrada de usuario en busca de los eventos del ratón luego proporciona esta información al gestor de cámara.
- En ejecutaLogica se realiza la actualización de la cámara en base a la información obtenida en el paso anterior. Esto se hace en la invocación de su método Update.

Precisamente el método ejecutaLógica ha de ser el responsable de que la cámara de Irrlicht sea actualizada convenientemente de acuerdo al gestor de cámara usado. Este gestor de cámara llamado ICamara nos permite pinchar distintas cámaras y hacer transaciones entre ellas. Para pinchar una cámara tenemos que usar el método SetStyle. Las cámaras que vamos a usar son las siguientes:

- Estacionaria (ICamera::Total_Estationary): establece una cámara fija en una determinada posición.
- De seguimiento (ICamera::Chase). Dada una entidad de la lógica, esta cámara se encargará de seguirla. Para ello, la cámara de seguimiento se establece como observador de esta entidad. Para fijar la entidad a la que se va a seguir se usa el método SetTarget().

Además de las anteriores cámaras, también existe una cámara de transición que permite hacer traslaciones entre los cambios de cámara. Esta cámara se pincha automáticamente cuando se cambia entre una de las dos anteriores.

Este gestor de cámara se usará sólo durante la ejecución del juego. Por tanto será la clase EstadoJuegoIrrlicht el responsable de la creación, inicialización y destrucción del gestor. Además se encarga de pinchar las cámaras que convengan en cada momento y de la redefinición del método ejecutaLógica. El gestor de cámara tendrá que actualizar la matriz de

cámara que usamos para renderizar. Los subestados también podrán pinchar distintas cámaras.

Sonido

Disponemos de un gestor de la interfaz sonido que será responsable de la creación y reproducción de sonidos en un entorno 3D, dentro de la clase GestorSound. El gestor se encuentra en la clase CEstadoJuegoIrrlicht y será usado gracias al patrón Observer. Se mantiene a la escucha de los eventos de la partida y de los avatares.

GestorSound

La clase se encarga de inicializar el motor de sonido 3D de IrrKlang e incorporar los sonidos que, presumiblemente, se van a reproducir. El gestor de sonido implementa la interfaz del observer de creación y destrucción de la partida, para los eventos de la partida y avatares. Cada vez que se produzca algún evento, el gestor procederá a emitir algún tipo de sonido. Por último, también se encarga de la liberación de los recursos utilizados por IrrKlang.

7. Conclusiones

A lo largo de este documento hemos visto la aplicación de la ingeniería del software dentro de los videojuegos. Es importante tener presente esta labor ya que, en gran medida, el éxito en la elaboración de nuestro producto depende de este trabajo. Un modelo de análisis y diseño lo suficientemente robusto garantiza el éxito en la implementación y el mantenimiento.

En la fase de programación, es posible que el resultado final en la ejecución difiera en algunos de los aspectos explicados teóricamente. El problema, es que las tecnologías empleadas son demasiado exigente con los recursos tecnológicos. Sin embargo, todo aspecto de implementación funciona correctamente, además es transportable, se podría utilizar en diferentes contextos para ver su auténtico potencial.

La fase de implementación, es la parte fundamental y más importante de todo el proyecto, donde se definió el juego. Gracias al trabajo realizado durante la fase de análisis y diseño se pudo implementar una arquitectura, para la parte específica del juego, mediante el uso de patrones de diseño de ingeniería del software. Los problemas que fueron surgieron se resolvieron sin demoras de tiempo ni tampoco la necesidad de recomponer o rehacer el código de nuevo, ya que en todo momento estábamos respaldado bajo un sólido trabajo de Ingeniería del Software.

La inteligencia artificial es otro de los pilares importantes en este proyecto, la construcción de robots viene siendo un campo de investigación muy activo en el ámbito de la inteligencia artificial y de la robótica móvil durante las dos últimas décadas. El problema de la construcción de mapas se considera uno de los hitos más importantes en el camino hacia robots que sean auténticamente autónomos. Hoy en día se dispone de métodos robustos para la construcción de mapas en entornos estáticos, estructurados y de un tamaño limitado. Consecuencia de ello es que hemos sido capaz de generar un BOT (limitado por los recursos que disponemos) capaz de desenvolverse en un entorno conocido. Sin embargo, construir mapas de entornos no estructurados, dinámicos y de grandes dimensiones continúa siendo un problema abierto a la investigación.

El último punto que queda por comentar es la aportación de los gráficos por computador a nuestro trabajo. Como bien sabemos, su evolución ha crecido en la medida que el hardware disponible avanzaba, ofreciendo como resultado la visualización de un entorno 3D cada vez más realistas. Estas exigencias provocan una gran consumición de los recursos del equipo informático, con una consecuente disminución del rendimiento del juego y del desarrollo de otras aéreas de importancia como la Inteligencia Artificial. También cabe destacar la complejidad que emprende estudiar un motor gráfico para modificarlo, a fin de permitir su acoplamiento con la arquitectura del juego.

Para finalizar, comentar que el principal objetivo de este trabajo ha sido crear una aplicación la cual abarcara el mayor número de ramas estudiadas durante los cinco años de carrera. Por

ello, se ha desarrollado una aplicación basada en múltiples y diversas tecnologías, que ha derivado al aprendizaje de diferentes técnicas. Como mérito personal, destaco el logro de incorporar importantes ramas de la informática dentro del proyecto.

Consideramos que se han alcanzado las expectativas planteadas dado que el proyecto se aproxima al trabajo que desarrolla un ingeniero en informática en el ámbito laboral, poniendo de manifiesto la necesidad de sus capacidades.

8. Apéndice

Documento de diseño del QuakeGarage

Género

El género del juego es de acción en primera persona, más conocido por FPS (first-person shooter). Un FPS es un videojuego que renderiza el mundo del juego desde la perspectiva visual del carácter y prueba la habilidad del jugador en la puntería con las armas y su capacidad de movimiento ya sea para atacar o defenderse. El realismo producido por los gráficos 3D combinado con una violencia extrema ha hecho de este género un asunto de controversia dentro de los videojuegos.

Descripción

Al igual que en muchos juegos del género, el objetivo en QuakeGarage es moverse a través de todo el campo de batalla eliminando a los adversarios. Cuando los puntos de vida de un jugador llegan a cero, el avatar del jugador es eliminado. La partida termina tan pronto como el jugador quiera abandonar la partida. El modo jugador te permite combatir contra un oponente controlado por una computadora (BOT). Este modo de juego, que viene por defecto en QuakeGarage, ponen a prueba las habilidades de los jugadores en batallas uno contra uno. Se pondrá en manifiesto las destrezas en el movimiento, los combates tácticos y estratégicos.

Introducción

QuakeGarage sigue la misma temática que Quake3: Arena. Ambos juegos no tienen una campaña basada en misiones. En su lugar hay una serie de luchas que simulan la experiencia multijugador, usando oponentes controlados por computadora, más conocidos como BOTS.

El juego carece de historia; Se trata de una propuesta de continuidad de los famosos juegos del Quake. Esto se ve reflejado en la inclusión de las entidades específicas del Quake2 y la adopción de la arquitectura de mapas del Quake3 (mezcla de arquitectura gótica con tecnología).

El juego

Menú principal

Esta es la raíz del juego, se obtiene cuando se ejecuta la aplicación QuakeGarage. Desde aquí se dispone de varias opciones.

- **Game mode:** Se establece la modalidad de juego. Desde esta opción se podrá elegir el escenario de QuakeGarage donde tendrá lugar el enfrentamiento.
- **Bot mode:** Esta modalidad está pensada sólo para la creación y desarrollo de los BOTS. Al igual que en el modo juego, se podrá elegir un escenario de QuakeGarage
- **Bot Setup:** La opción de configuración nos lleva a la pantalla de configuración, donde se puede personalizar los controles, las opciones del juego y del sistema.
- **Credit:** Se muestran los nombres de los elementos más significativos en la construcción del juego.
- **Exit:** La opción de salida. Después de confirmar la selección, se cerrará QuakeGarage y se regresará al escritorio.

Menú de configuración

El menú de configuración, accesible desde "Setup" en el menú principal, le permite cambiar ajustes importantes, como las características del jugador, los controles, los parámetros del sistema, los gráficos, y las opciones de juego. Cada una de las opciones se describe a continuación.

Player

La configuración del jugador permite especificar varias características acerca de su perfil. Las opciones son las siguientes:

- **Name**: Nombre del jugador
- **Model**: El modelo que vamos a usar para el jugador
- **Skin**: La apariencia que tendrá el modelo elegido.

Controls

El menú de los controles permite personalizar los controles del jugador y la interfaz de juego. Probablemente haya que pasar un poco de tiempo en esta opción, para acomodar la configuración del teclado adecuadamente. Contiene los siguientes submenús:

- **Move:** Submenú de movimiento que permite configurar las teclas de dirección de movimiento, así como las acciones de salto y agacharse.
- **Shot:** Submenú de disparo que permite asignar teclas y botones del ratón para atacar y cambiar las armas.
- **Misc:** Submenú con varias opciones, tales como ver los resultados o salir del juego.

System

El menú del sistema permite ajustar algunas de sus configuraciones del sistema para mejorar el rendimiento del juego y experiencia. Contiene varios submenús:

- **Graphics:** Submenú de gráficos que permite ajustar la configuración gráfica, como los drivers, la resolución y otros aspectos.
- **Sound:** Submenú de sonido que permite ajustar la configuración de sonido.
- **Network:** Submenú de la red que permite ajustar las tasas de transferencia de datos.

Game Options

El menú de opciones de juego permite ajustar aspectos internos al juego, son parámetros que afectan a la jugabilidad.

Modo Juego

QuakeGarage es un juego para un único jugador, que se caracteriza por los torneos de “1vs1”. Ofrecen un juego más estratégico que el caos que caracteriza los “free for all” (todos contra todos). Las acciones que garantizan éxito, dependen del dominio de los siguientes aspectos: movimiento, combates tácticos y estratégicos.

Eligiendo un escenario

Antes de cargar el juego, debemos elegir un mapa. A través de la interfaz del juego se mostrarán los diversos escenarios. Cuando queramos comenzar la partida tan sólo debemos pulsar en “fight” y cambiaremos al modo juego.

Niveles dinámicos

QuakeGarage dispone de una serie de mapas de batallas, en los que combatiremos contra un único enemigo, dotado de las mismas capacidades que nuestro avatar. Para superar una batalla, el jugador deberá alcanzar un máximo de puntuación sin límite de tiempo. El juego es simple: Cada jugador comenzará desde una posición aleatoria del mapa en las mismas condiciones, y tendrá que intentar eliminar a su adversario, utilizando los recursos que se le proporciona a lo largo del escenario. Cada vez que un jugador muere, este comenzará nuevamente en una posición aleatoria del mapa, con toda la puntuación inicializada. Existen ciertos riesgos que debemos tomar en cuenta; se sabe que ciertas zonas de comienzo son mejores que otras, y eso provoca una situación de desequilibrio. También, cuando un jugador muere es posible que este vuelva a aparecer cerca de su adversario, y se encontraría en una situación muy desfavorable.

The Camping Grounds

Descripción

Es uno de los mapas más populares del Quake3. The Camping Grounds es un gran nivel, con un amplio surtido de “bounce pads” y de una arquitectura extremadamente alta. Es un escenario suficientemente complejo para hacer batallas muy interesante, pero no tan grande como para que los jugadores no se puedan encontrar. El mapa está lo suficientemente conectado con las diversas áreas que las componen, y las entidades están bien dispersadas por todo el nivel. La verdadera clave de este nivel está en aprender a desenvolverte. Esto podría permitir a un jugador experto, por ejemplo, cuando esta herido, perder a un atacante el tiempo suficiente para sanar.

Diseño

- Armadura: Armor Shard, Combat Armor
- Armas: Super Shotgun, Hyper Blaster, Chaingun, Grenade Launcher, Rocket Launcher, Railgun
- Munición: Slugs, Bullets, Shells, Rockets, Cells.
- Vida: First Aid, Medkit

Dentro del juego

La Interfaz gráfica de usuario (GUI), consiste en la ventana principal del campo de batalla (visto desde la perspectiva de nuestro "héroe"). Se incluye un puntero, que indica a donde está señalando y una serie de iconos, que indican el estado de nuestro jugador. Además cuando el usuario solicita ver los resultados, se mostrará por pantalla diversos datos, tales como el resultado, el ping, los frames por segundo y el nombre.

HUD (Head Up Display)

El HUD es una herramienta valiosa para el que el usuario pueda seguir el estado de su jugador. El HUD de QuakeGarage tiene un número de componentes; la entidad que acabamos de coger, la vida, el arma y la munición usada, el escudo y la última arma que hemos cogido, estos estados se pueden observar en la parte inferior de la pantalla.

Entidades y Jugadores

Para probar la habilidad y la capacidad de maniobrar del usuario se han dispuesto a lo largo del campo de batalla (mapa) una serie de entidades. Las entidades pueden referirse a cualquier cosa que tiene un papel dinámico en el juego. Estas son las armaduras, las armas, las municiones, los jugadores, los disparos, algunos ítems especiales, etc…

Los jugadores poseen elementos, armas, escudos, habilidades, y atributos que son comunes para todos. Del mismo modo, siguiendo esta igualdad, cada uno de los jugadores pueden realizar las mismas acciones, (moverse, atacar, utilizar entidades…) sin que haya ningún tipo de beneficio durante el combate, siendo controlados por un humano o una computadora.

Todas las entidades tienen unos atributos en común, y otros propios dependiendo de la clase a la que pertenezca, en los siguientes apartados desglosaremos una por una cada característica de las entidades. Aquí mencionaremos las comunes:

- Nombre
- Clase
- Ubicación
- Visibilidad
- Malla

Clases

Cada entidad de QuakeGarage pertenece a una determinada clase. Entendemos a clase como un conjunto de características que definen a la entidad. Las clases son las siguientes:

- Munición
- Armadura
- Avatar
- Disparo
- Efectos de disparo
- Vida
- Items especiales
- Arma

Las clases actúan como guía de cómo se desarrollan las entidades con el tiempo. Debido a que teóricamente todos los personajes del juego tienen un mismo comportamiento, se ha desarrollado una clase común para ellos (Avatar) en el que queda claramente definido sus propiedades.

Todas las entidades miembros de una clase se han clasificado según su uso en el juego y se pueden identificar visualmente por su forma y retrato.

Avatares

Entidad que controlaremos en todo el juego, son el centro de atención del juego y los modelos que utilizaremos para lograr nuestros objetivos.

Atributos y habilidades

Cada jugador tiene unos atributos individuales, que van cambiando a lo largo del juego, influenciados por la capacidad de ataque del adversario, cuánto daño se le puede infligir al jugador y demás. Los atributos recogidos son los siguientes:
Velocidad, salto, movimiento, rotación, agachado, vida, escudo, arma, munición, ítem especial.

- El arma y la munición determinan la cantidad de daño que un jugador puede deliberar en combate. Hay que añadir a esto, que las posibilidades de daño serán mayores mientras más cercano del oponente se esté y también mientras menos movimientos se produzca.
- La velocidad indica la distancia que puedes recorrer por el tiempo, su variación depende del control del usuario con el teclado o también de la máquina. Determinados eventos durante la partida pueden producir variación en la velocidad.
- El escudo conforma la resistencia del jugador, influye en cuanto daño puede recibir un jugador. Dependiendo del tipo de escudo que poseas tendrás mayor resistencia a los

disparos de ciertas armas. En principio el escudo, sólo protege al jugador y no varía la velocidad del mismo.

- La condición describe el estado actual del jugador. Hay diversas posibilidades en la que se puede encontrar; saltando, moviéndose, quieto, agachado o rotando. También puede estar en un estado especial, tras haber utilizado algún ítem especial, que le ha podido aumentar la vida, el escudo, la fuerza de disparo o la velocidad.

Nombre	*Clase*	*Información personal*
Jugador	Avatar	Jugador que puede ser manejado por I.A o usuario

Vida

Stim Pack

Stim Pack proporciona siempre dos punto de mejora a tu puntuación de salud total. A menudo, Stim Packs se encuentran en grupos de tres o más y no deberían ser ignorados en las batallas. La recopilación de sólo un Stim Pack, por ejemplo, marca la diferencia entre la muerte y la supervivencia después de ser alcanzado con el arma más potente (la Railgun).

First Aid

First Aid puede ser recogido sólo para recuperar la perdida de salud, y no se pueden colectar cuando se ha alcanzado el máximo de salud.

Medkit

Medkit está en el mismo caso que el First Aid salvo que recupera mayor cantidad de salud.

Nombre	Carga por unidad	Carga máxima	Tiempo reaparición
Stim Pack	2	sin limite	30 seg
First Aid	10	100	30 seg
Medkit	25	100	30 seg

Armadura

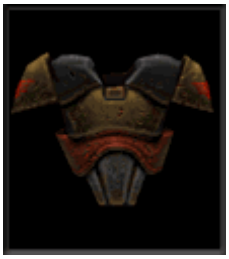
Armor Shard

El Armor Shard añade siempre 2 puntos de armadura a tu protección actual. Si no tienes ningún tipo de protección, el efecto protector será equivalente al escudo más débil.

Jacket Armor

Jacket Armor es la armadura física más débil. Proporciona un 30% de protección contra las armas normales y ninguna protección contra las armas de energía (Blaster, Hyperblaster). Además ofrece la menor cantidad de protección física de todas las armaduras. Si queremos pasar a un nivel mayor de protección se harán cálculos de equivalencia.

Combat Armor



Combat Armor es el traje de protección intermedio. Proporciona un fuerte 60% de protección contra armas normal y 30% de protección contra las armas de energía. Combat Armor es la armadura más equilibrada del juego, sus prestaciones te defienden de todas las armas y los efectos disminuyen con una relativa lentitud. Si tenemos una armadura de mayor grado, nuevamente se harán cálculos de equivalencias.

Body Armor

Body Armor es la más fuerte de todas las armaduras físicas. Proporciona hasta un 80% de protección contra los ataques normales y un 60% contra las armas de energía. Body Armor aumenta considerablemente la cantidad de protección. Sin embargo, tiene el inconveniente de que sus efectos disminuyen más rápidamente.

Nombre	Carga por unidad	Carga máxima	Protección normal	Protección energía	Cambio armadura	Tiempo reaparición
Armor Shard	2	sin limite	-	-	-	20 seg
Jacket Armor	25	50	30%	0%	30%	20 seg
Combat Armor	50	100	60%	30%	60%	20 seg
Body Armor	100	200	80%	60%	-	20 seg

Munición

Shells **Bullets** **Grenades** **Rockets** **Cells** **Slugs**






Nombre	Carga por unidad	Carga máxima	Armas	Tiempo reaparición
Shells	2	50	Super Shotgun	30 seg
Bullets	50	200	Chaingun	30 seg
Grenades	5	50	Grenade Launcher	30 seg
Rockets	5	50	Rocket Launcher	30 seg
Cells	50	200	Hyper Blaster	30 seg
Slugs	10	50	Railgun	30 seg

Items especiales (Power-Ups)

Se disponen de power-ups ofensivas y defensivas en Quake Garage, que pueden tener un impacto significativo en la lucha.

Quad Damage

El Quad Damage te permite hacer cuatro veces más de daño que con un disparo ordinario. Funciona de igual modo para cada arma. Quad Damage dura 30 segundos desde el momento en que es activada.

Mega Health

La recolección de Mega Health, al menos dobla tus posibilidades de vida a lo largo del enfrentamiento. Cada Mega Health te añadirá una friolera cantidad de 100 puntos a su total de la salud, hasta un máximo de 250 puntos de salud. En cada segundo trascurrido, una vez que se haya captado el Mega Health, disminuirá un punto de tu salud hasta alcanzar 100 puntos.

Armas

Blaster

- Clase de arma: Arma de energía
- Tipo de munición: Ningun
- Munición por disparo: Cero
- Frecuencia de disparo: Dos por Segundo
- Rango más efectivo: Distancia corta
- Mejor estrategia: Coge un arma mejor.

El disparo de la Blaster es de la misma materia que la HyperBlaster, pero no quema ningún célula. Los proyectiles se lanzan en dirección del “cross-hair” (nuestro puntero en la pantalla) y con una mínima variación alcanza el punto de destino.

El arma es igualmente eficaz en el punto blanco a mediano rango, pero a menudo es difícil alcanzar un objetivo lejano.

Super Shotgun

- Clase de arma: Arma de múltiples disparos
- Tipo de munición: Shells
- Munición por disparo: Dos Shells
- Frecuencia de disparo: Un disparo por segundo
- Rango más efectivo: Cualquier punto a corta distancia
- Mejor estrategia: Sólo exponerse al enemigo durante el turno de disparo

La Super Shotgun dispara dos cartuchos (uno por cada salida). Los dos cartuchos varían 5 grados hacia la izquerda y derecha del centro. La amplia difusión de los perdigones de ambos cartuchos amplían las posibilidades dañar al adversario incluso a larga distancia.

Debido a la amplia difusión de los perdigones, el arma es más eficaz en un rango cercano y es capaz de matar en un solo disparo a un adversario sin armadura. Por el contrario, esta difusión de perdigones provoca que a mayores distancias el daño disminuya.

Chaingun

- Clase de arma: Arma de disparos rapidos
- Tipo de munición: Bullets
- Munición por disparo: Un Bullet
- Frecuencia de disparo: Veinte Bullets por segundo
- Rango más efectivo: Distancia media - larga
- Mejor estrategia: Mantener la distancia y el puntero en el objetivo.

La chaingun funciona como una ametralladora capaz de disparar una gran cantidad de balas por segundo. Tiene una lenta puesta en marcha, con una baja tasa de fuego. Además, tiene un período de enfriamiento para que el cañón del arma termine de girar. Esto ralentiza operaciones de cambio de arma y te deja en una situación desfavorable.

Al girar a la máxima velocidad, el arma causa daños muy rápidamente a los objetivos cercanos al “crosshair”. El arma es muy eficaz utilizado en áreas abiertas donde se pueda mantener fijo el crooshair a un objetivo.

Grenade Launcher

- Clase de arma: Arma explosiva
- Tipo de munición: Granadas
- Munición por disparo: Una granada
- Frecuencia de disparo: Una por segundo
- Rango más efectivo: Distancia corta - media
- Mejor estrategia: Disparar durante el retiro, en las esquinas

Grenade launcher lanza granadas que viajan en forma de arco, dirección donde se esté apuntando. Ellos rebotarán hasta la detonación, (alrededor de 3 segundos después de ser lanzados). Estas se detonan instantáneamente al colisionar con algún jugador y ofrecen un largo radio de daño cuando explotan.

Rocket Launcher

- Clase de arma: Arma explosiva
- Tipo de munición: Rockets
- Munición por disparo: Un Rocket
- Frecuencia de disparo: Cinco Rockets cada cuatro segundos
- Rango más efectivo: Distancia corta - larga
- Mejor estrategia: Se describe a continuación.

Rocket launcher lanza cohetes que viajan del mismo modo que los disparos de la Blaster y a la misma velocidad. Tras la detonación, el cohete proporciona un pequeño radio de daño en la zona de la explosión. Esta arma puede ser utilizada para coger un gran impulso en el aire y acceder a otras zonas, aunque a cambio te quitará cierta cantidad de vida dependiendo de cómo se efectúe.

Hyper Blaster

- Clase de arma: Arma de energía
- Tipo de munición: Cells
- Munición por disparo: Un Cell
- Frecuencia de disparo: Diez Cells por segundo
- Rango más efectivo: Cualquier punto a distancia media
- Mejor estrategia: Persigue y cierra las distancia con el objetivo

La Hyper Blaster es una rápida variación de fuego de la conocida Blaster con un par de excepciones. En primer lugar, cada descarga utiliza una célula munición. En segundo lugar, cuando se deja de apretar el gatillo, el arma no puede ser disparado de nuevo hasta que el cañón se detenga (una revolución: alrededor de medio segundo). La pistola es más efectiva a corta distancia y requiere de mucha precisión saber dirigir los disparos.

Railgun

- Clase de arma: Arma especial
- Tipo de munición: Slugs
- Munición por disparo: Un slug
- Frecuencia de disparo: Un slug cada 1.5 segundos
- Rango más efectivo: Cualquier punto a distancia larga
- Mejor estrategia: Puntería.

La railgun es el arma más efectiva, pero con el mayor tiempo de recarga. Se trata de una arma única que requiere de una gran puntería para su uso. Tiene un efecto localizador y es igualmente efectivo en cualquier rango.

Nombre	Daño por disparo	Frecuencia disparo	Radio de daño	Velocidad disparo	Tipo de munición	Tiempo reaparición
Blaster	15	0.5	-	0.8	-	30 seg
Super Shotgun	6	1	-	10.5	Shell	30 seg
Chaingun	6	0.05	-	10.5	Bullet	30 seg
Grenade launcher	100	1	160	0.8	Grenade	30 seg
Rocket launcher	100	1.25	120	0.8	Rocket	30 seg
Hyper Blaster	20	0.1	-	0.8	Cell	30 seg
Railgun	100	1.5	-	10.5	Slug	30 seg

Puerta

Las puertas bloquean la línea de visión (obviamente), y también de sonido. Si no hay un pasadizo o una puerta abierta que conduzca a una zona adyacente, no se puede oír sonidos que suceden allí. Las puertas abren automáticamente cuando cualquier tipo de entidad está cerca de ellos.

Pad de saltos

Cuando un jugador pisa una pad de saltos, este impulsa al jugador al aire (con un ruido). Los Pads (incluyendo a los pads de aceleración) están, por lo general, destinadas a un lugar determinado, por lo que no todos los pads de saltos apuntan directamente hacia arriba. La gravedad en el entorno y tu estado actual de movimiento puede cambiar, un poco, la trayectoria del salto.

Pad de aceleración

El pad de aceleración siempre tienen un cierto ángulo de inclinación. Al tocarlo saldrás disparado hacia el espacio, en la dirección que apunta el pad y hacia arriba. Hay que tener cuidado ya que es muy fácil salirse de la trayectoria y caer fuera del escenario

Teletransportador

A través de un teletransportador podremos ir a un lugar distinto; diferentes teletransportadores en el mismo escenario pueden llevarte a diferentes áreas, pero cada uno siempre lleva al mismo lugar. El teletransportador no necesariamente te lleva a un lugar donde hay otro teletransportador, es decir, puede ser de un solo sentido.

Puerta

Una puerta es igual que teletransportador, excepto que se puede ver a través de él su destino. Si alguien está en la zona que la puerta le llevará, entonces lo podrá ver si mira a través de la puerta. La puerta no difiere mucho del teletransportador, sin embargo, se le puede sacar cierto partido. Puedes usarlo como vía de escape, aunque también, y dado que la puerta es transparente es posible que alguien te esté esperando detrás.

Audio

Esta sección proporciona una lista de sonidos y efectos. La mayoría de los recursos nombrados aparecen en diferentes niveles.

Música

La música de fondo en QuakeGarage es una banda sonora que intenta recrear el ambiente de combate.

Sonidos y efectos

En el transcurso del juego QuakeGarage se escucharán diferentes sonidos y efectos. Se agrupan en las siguientes categorías:

- Efectos vocales: Son los sonidos que emite el jugador cuando realiza cualquier acción, por ejemplo, sonido al caminar, al saltar, etc…

Efecto de sonido	Situación	Nota
Caminar	Sonido emitido al caminar	Sonido emitido al desplazarse de un lugar a otro pulsando alguna tecla de dirección (duración 1 seg).
Saltar	Sonido emitido al impulsarnos	Sonido emitido después de pulsar la tecla de salto (duración 1 seg).
Morir	Sonido emitido al morir	Sonido emitido después de morir (duración 1 seg).

- Sonidos de batalla: Se producen con el sonido de las armas al disparar y los efectos de los proyectiles al impactar contra cualquier cosa.

Efecto de sonido	Situación	Nota
Railgun	Sonido de la Railgun	Sonido emitido al disparar con la Railgun (duración larga)
Chaingun	Sonido de la Chaingun	Sonido emitido al disparar con

		la Chaingun (duración corta)
Super Shotgun	Sonido de la Super Shotgun	Sonido emitido al disparar con la Super Shotgun (duración larga)
Rocket launcher	Sonido de la Rocket launcher	Sonido emitido al disparar con la Rocket launcher (duración larga)
Grenade launcher	Sonido de la Grenada launcher	Sonido emitido al disparar con la Grenada launcher (duración larga)
Hyper Blaster	Sonido de la Hyper Blaster	Sonido emitido al disparar con la Hyper Blaster (duración corta)
Blaster	Sonido de la Blaster	Sonido emitido al disparar con la Blaster (duración media)
Impacto Rocket	Sonido al impactar un Rocket	Sonido emitido al impactar o estallar un Rocket (duración 1 seg)
Impacto	Sonido de impacto	Sonido emitido por al impactar un proyectil (duración 1 seg)

- Ruidos ambientales: Pueden suceder dependiendo de la zona y el momento en el que estemos, por ejemplo, el ruido al saltar con un Bounce Pad, el sonido al reaparecer en el escenario, al coger alguna entidad, etc…

Efecto de sonido	Situación	Nota
Munición	Sonido emitido al coge munición	Sonido emitido cuando se ha recogido cualquier tipo de munición (duración 1 seg).
Armadura	Sonido emitido al coger armadura	Sonido emitido cuando se ha recogido cualquier tipo de armadura (duración 1 seg).
Arma	Sonido emitido al coger arma	Sonido emitido cuando se ha recogido cualquier tipo de arma (duración 1 seg).

Vida	Sonido emitido al coger vida	Sonido emitido cuando se ha recogido cualquier tipo de vida (duración 1 seg).
Respawn	Sonido emitido al aparecer en el escenario	Sonido emitido cuando un jugador hace aparición en el escenario (duración 2 seg).
Pads	Sonido emitido al ser impulsado por un pad.	Sonido emitido tras ser impulsado por un pad (duración 2 seg).