



Universidad
de Cádiz



Escuela Superior
de Ingeniería

ESCUELA SUPERIOR DE INGENIERÍA
GRADO EN INGENIERÍA INFORMÁTICA

**DESARROLLO DE AGENTES
INTELIGENTES PARA UN JUEGO DE
ESTRATEGIA EN TIEMPO REAL**

AUTOR: José Joaquín Arias Gómez-Calcerrada

Puerto Real, septiembre 2019



**Escuela Superior
de Ingeniería**

**ESCUELA SUPERIOR DE INGENIERÍA
GRADO EN INGENIERÍA INFORMÁTICA**

**DESARROLLO DE AGENTES
INTELIGENTES PARA UN JUEGO DE
ESTRATEGIA EN TIEMPO REAL**

DIRECTOR: Alberto Gabriel Salguero Hidalgo
AUTOR: José Joaquín Arias Gómez-Calcerrada

Puerto Real, septiembre 2019

Agradecimientos

Quiero agradecerle, en primer lugar, a mi familia por el apoyo que me han dado desde siempre. Sin este, el proyecto no hubiera sido posible.

También quiero agradecer el apoyo de mis amigos, que siempre se han ofrecido a prestar ayuda y ánimos esenciales para lograr este objetivo.

Gracias a Alberto Salguero por su atención durante el transcurso del proyecto, así como su ayuda y aportación de ideas clave.

Gracias a ‘baranpirincal’, usuario de GitHub que ha contribuido a la creación del logo del proyecto.

Muchas gracias a todos.

Licencia

Este documento se ha liberado bajo la licencia GFDL 1.3 (GNU Free Documentation License). Los términos de esta están descritos al final del documento.

Copyright (c) 2019 José Joaquín Arias Gómez-Calcerrada

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

Resumen

Este Trabajo Fin de Grado consiste en un juego denominado *Space Hunt*, que consiste en la supervivencia de un grupo de astronautas contra una raza alienígena que habita en el planeta *Marte*.

Tradicionalmente, el control de las entidades dentro de un juego ha estado controlado siempre por el jugador. A medida que se fueron añadiendo enemigos a los juegos de estrategia, se fueron introduciendo los primeros algoritmos que contralarían de forma automática a estos enemigos.

Esta inteligencia artificial se ha ido mejorando poco a poco, pero el jugador ha seguido teniendo el control absoluto de sus unidades. Lo que se logra con este proyecto es la capacidad del control automático de las unidades del jugador, de forma que este pueda tomar decisiones sobre sus actos pero que sean capaces de tomar decisiones por ellas mismas en caso de que el jugador se ausente o que no le apetezca controlarlas.

Marte es el planeta en el que ocurre el juego. Su generación es procedural y el jugador puede configurarlo ligeramente, cambiando el tamaño y la semilla del planeta.

Al comienzo de cada partida, un grupo de ocho astronautas aparece en *Marte*. Aterrizan en el planeta rojo con la misión de eliminar a un grupo de ocho alienígenas. Para ir en su búsqueda, buscarán el punto más alto del planeta. Dado que sólo tienen conciencia de la altitud de las montañas sobre las que van pasando, tendrán que explorar el planeta para lograr buscar la montaña más óptima.

El jugador podrá elegir el porcentaje de exploración de los astronautas. Estos explorarán el planeta siguiendo el algoritmo *Particle Swarm Optimization*, por lo que una componente importante es la inercia, que irá decreciendo a medida que explora. Cuanto más cerca del 0 % sea el porcentaje de exploración, menor es la inercia y, por consiguiente, mayor es la probabilidad de que caigan en un máximo local cercano al punto de aparición. Hay más tendencia de máximo global si el porcentaje es mayor, pero un exceso de inercia puede causar que ignoren montañas altas cercanas al punto de aparición. Es trabajo del usuario decidir esta variable de exploración en función de la topología del planeta generado.

Una vez hayan llegado al punto más alto encontrado por ellos, el jugador le asignará a cada uno de ellos un arma. Esta podrá ser un escudo o una espada, y causarán que el astronauta desempeñe funciones distintas a la hora del combate con los alienígenas. El usuario se debe basar en la velocidad de cada astronauta y en su vida para decidir si es más apto de la espada o del escudo.

Mientras que los astronautas atacan cuerpo a cuerpo, los alienígenas atacan a distancia lanzando proyectiles en forma de bola de tierra marciana, por lo que la inteligencia artificial que controla cada uno de los dos grupos de entidades es diferente. A los alienígenas les

interesa mantener distancia y mantenerse alejados de sus compañeros, mientras que a los astronautas les interesa mantener una comunicación atacante-defensor, de forma que el defensor interponga los proyectiles y el atacante proceda a eliminar a los alienígenas.

El jugador podrá priorizar la defensa sobre ciertos astronautas, para evitar malas gestiones por parte de la inteligencia artificial. Asimismo, podrá cambiar la atención de los astronautas sobre los alienígenas para enfocar unos más asequibles.

La misión será exitosa si los astronautas eliminan a todos los alienígenas de Marte. Sin embargo, si los alienígenas eliminan a todos los atacantes, el juego se acabará y la misión resultará fallida.

■ **Palabras Claves**

Generación procedural de contenido, Agentes inteligentes, Juego.

Índice general

Índice de Figuras	2
Índice de Tablas	4
1. Planteamiento y Objetivos	5
1.1. Motivación	5
1.2. Objetivos	5
1.3. Alcance	6
1.4. Glosario	7
1.4.1. Acrónimos	7
1.4.2. Definiciones	7
1.5. Contenido de la memoria	8
2. Estado del Arte	9
2.1. Inteligencia artificial en videojuegos	9
2.1.1. Introducción	9
2.1.2. Historia	9
2.1.3. Videojuegos modernos	11
2.1.3.1. Algoritmo A^*	11
2.1.3.2. Particle Swarm Optimization (<i>PSO</i>)	12
2.2. Renderizado de grandes mapas	14
2.2.1. Generación procedural del mapa	14
2.2.2. Renderizado con nivel de detalle cambiante (<i>LOD</i>)	15
3. Planificación	17
3.1. Metodología de desarrollo	17
3.2. Etapas	18
3.3. Diagrama de Gantt	21
3.4. Presupuesto	22
4. Requisitos	23
4.1. Requisitos funcionales	23
4.2. Requisitos no funcionales	28

5. Análisis	30
5.1. Modelo conceptual	30
5.1.1. Menú principal	30
5.1.2. Juego	35
5.2. Modelo Casos de Uso	38
5.2.1. Diagrama de Casos de Uso	38
5.2.2. Especificación de Casos de Uso	40
6. Diseño	51
6.1. Arquitectura Física	51
6.2. Diseño Físico de Datos	51
6.3. Diseño detallado de la Interfaz de Usuario	53
7. Construcción del Sistema	56
7.1. Diseño Gráfico	56
7.2. Desarrollo del Juego	59
7.2.1. Generación del terreno	59
7.2.2. Actualización de los <i>chunks</i>	63
7.2.3. Inteligencia Artificial	66
7.2.3.1. PSO de los astronautas	66
7.2.3.2. Algoritmo de batalla de los astronautas	67
7.2.3.3. PSO de los alienígenas	70
7.2.3.4. Algoritmo de batalla de los alienígenas	71
8. Pruebas	72
8.1. Pruebas Unitarias	72
8.2. Pruebas de Integración	73
8.3. Pruebas de Sistema	73
8.3.1. Pruebas Funcionales	73
8.3.2. Pruebas No Funcionales	73
9. Conclusiones y Trabajo Futuro	75
9.1. Problemas encontrados	75
9.2. Conclusiones	75
9.2.1. Rendimiento y Carga de Terreno	75
9.2.2. Estadísticas	76
9.3. Trabajo Futuro	80
A. Anexos	81
A.1. Manual de Instalación	81
A.1.1. Introducción	81
A.1.2. Requisitos Previos	81
A.1.3. Procedimiento de instalación	81
A.2. Manual de Usuario	81

A.2.1. Introducción	81
A.2.2. Jugar	82
A.2.2.1. Escenario principal	82
A.2.2.2. Fase de exploración	83
A.2.2.3. Fase de preparación	83
A.2.2.4. Batalla final	84
A.2.3. Pantallas adicionales	86
B. GNU Free Documentation License	90

Índice de figuras

2.1. Juego Nim	10
2.2. Juego Space Invaders	11
2.3. Ejemplo de aplicación del algoritmo A^*	12
2.4. Actualización de la velocidad de una partícula	13
2.5. Generación de contenidos en el mapa de Minecraft	15
2.6. Diferencia de LOD entre diferentes porciones de un mismo terreno	16
3.1. Esquema de un ciclo Sprint	18
3.2. Diagrama de Gantt	21
5.1. Diagrama Conceptual del Menú Principal	31
5.2. Diagrama Conceptual del Menú de Opciones	32
5.3. Diagrama Conceptual del Menú de Volumen	33
5.4. Diagrama Conceptual del Menú de Generación	34
5.5. Diagrama Conceptual del Menú de Pausa	35
5.6. Diagrama Conceptual del Juego	36
5.7. Diagrama Conceptual de la Cámara	36
5.8. Diagrama Conceptual del Planeta	37
5.9. Diagrama Conceptual de las Entidades	38
5.10. Modelo de Casos de Uso	39
6.1. Configuración en formato XML	52
6.2. Estado de un <i>astronauta</i>	53
6.3. Estado de un <i>alienígena</i>	53
6.4. Menú principal del juego	54
6.5. Menú de configuración	54
6.6. Menú de configuración del volumen	55
6.7. Menú de configuración de la generación	55
7.1. Signo de selección	57
7.2. Signo de exclamación	57
7.3. Bandera de astronauta	58
7.4. Asignar arma	58
7.5. Signo de enfado	58

7.6.	Signo de tristeza	58
7.7.	Asignar espada	58
7.8.	Asignar escudo	58
7.9.	Atacar alienígena	58
7.10.	Defender astronauta	58
7.11.	Cursor arriba	59
7.12.	Cursor izq.	59
7.13.	Cursor centro	59
7.14.	Cursor der.	59
7.15.	Cursor abajo	59
7.16.	Cursor agarrando	59
7.17.	Cursor señalando	59
7.18.	Ruido de Perlín 2D	61
7.19.	Matrices de ruido	63
7.20.	Actualización de <i>chunks</i>	64
7.21.	Algoritmo PSO aplicado a los astronautas	67
7.22.	Algoritmo de defensa de un escudero a un espadachín	68
7.23.	Algoritmo de ataque de un espadachín a un alienígena	70
7.24.	Algoritmo PSO aplicado a los alienígenas	71
9.1.	Estadísticas de la trayectoria del astronauta número 0	77
9.2.	Estadísticas de la trayectoria del astronauta número 4	78
9.3.	Estadísticas de la mejor puntuación global de la trayectoria de los astronautas	79
9.4.	Valor de la inercia a lo largo de la trayectoria de los astronautas	79
A.1.	Directorio base de la aplicación	82
A.2.	Escenario principal	82
A.3.	Fase de exploración	83
A.4.	Fase de preparación	84
A.5.	Comienzo de la batalla	85
A.6.	Botones de priorización de ataque (ninguno seleccionado)	85
A.7.	Botones de priorización de ataque (seleccionado ataque)	86
A.8.	Botones de priorización de ataque (seleccionado defensa)	86
A.9.	Menú de pausa	87
A.10.	Mission Success	87
A.11.	Game Over	88

Índice de tablas

3.1. Costes	22
4.1. Requisito Funcional 4.2.1 - Mover la cámara por el mundo	23
4.2. Requisito Funcional 4.2.2 - Hacer zoom	23
4.3. Requisito Funcional 4.2.3 - Cambiar la cámara a la posición de manejo global del mundo	24
4.4. Requisito Funcional 4.2.4 - Cambiar cámara a la posición de un determinado astronauta	24
4.5. Requisito Funcional 4.2.5 - Selección de astronautas	24
4.6. Requisito Funcional 4.2.6 - Mostrar información de cada astronauta	25
4.7. Requisito Funcional 4.2.7 - Indicar la defensa de un astronauta	25
4.8. Requisito Funcional 4.2.8 - Indicar el ataque a un alienígena	25
4.9. Requisito Funcional 4.2.9 - Asignar escudo a un astronauta	25
4.10. Requisito Funcional 4.2.10 - Asignar espada a un astronauta	26
4.11. Requisito Funcional 4.2.11 - Elegir el porcentaje de exploración de los astronautas	26
4.12. Requisito Funcional 4.2.12 - Configurar la generación del mundo	26
4.13. Requisito Funcional 4.2.13 - Configurar el volumen del juego	26
4.14. Requisito Funcional 4.2.14 - Iniciar una partida	27
4.15. Requisito Funcional 4.2.15 - Pausa	27
4.16. Requisito Funcional 4.2.16 - Salir del juego	27
4.17. Requisito Funcional 4.2.17 - Reproducir música	27
4.18. Requisito Funcional 4.2.18 - Reproducir efectos de sonido	28
4.19. Requisito Funcional 4.2.19 - Lectura de la configuración	28
4.20. Requisito Funcional 4.2.20 - Colocar una bandera a un astronauta	28
4.21. Requisito No Funcional 4.3.1 - Fiabilidad	28
4.22. Requisito No Funcional 4.3.2 - Eficiencia	29
4.23. Requisito No Funcional 4.3.3 - Accesibilidad	29
4.24. Requisito No Funcional 4.3.4 - Seguridad	29
5.1. ATC-01 - Jugador	40
5.2. CU-01 - Iniciar Partida	40
5.3. CU-02 - Configuración	41

ÍNDICE DE TABLAS

4

5.4. CU-03 - Salir	41
5.5. CU-04 Pausar Juego	41
5.6. CU-05 Elegir porcentaje de exploración	42
5.7. CU-06 Selección de un determinado astronauta	42
5.8. CU-07 Asignar bandera a un determinado astronauta	43
5.9. CU-08 Mover cámara por el mundo	43
5.10. CU-09 Hacer zoom	44
5.11. CU-10 Cambiar foco de la cámara	45
5.12. CU-11 Indicar la defensa de un determinado astronauta	46
5.13. CU-12 Indicar el ataque a un determinado alienígena	47
5.14. CU-13 Asignar espada a un determinado astronauta	48
5.15. CU-14 Asignar escudo a un determinado astronauta	48
5.16. CU-15 Configurar efectos	49
5.17. CU-16 Configurar banda sonora	49
5.18. CU-17 Configurar semilla	50
5.19. CU-18 Configurar tamaño	50

Capítulo 1

Planteamiento y Objetivos

En este capítulo, se describirá la motivación que me ha impulsado a realizar este proyecto, además de demás puntos introductorios al mismo como los objetivos de este, una breve descripción de su contenido y su alcance.

1.1. Motivación

En el transcurso de ingeniería informática y sobre todo en la rama de computación, a los estudiantes nos han dotado de mucho conocimiento en cuanto al diseño de algoritmos y sistemas inteligentes.

Mi mayor motivación para llevar a cabo este proyecto ha sido el empleo de estos algoritmos para automatizar tareas dentro de un juego. Ver los resultados de la toma de decisiones de una entidad manejada por estos algoritmos siempre ha sido algo que me ha llamado la atención.

Asimismo, también me motiva el empleo de algoritmos que no había aprendido en la carrera para mejorar el rendimiento del juego, así que se aplican técnicas de renderizado del terreno como el *Dynamic Level Of Detail*, que explicaré más adelante en este documento. De esta forma, el videojuego puede ejecutarse en una amplia gama de computadores, sin necesidad de tener altas prestaciones a nivel de hardware.

1.2. Objetivos

El objetivo principal del TFG es el desarrollo de un videojuego usando la herramienta Unity3D [1], con predominancia de la *IA* y con un diseño y arquitectura eficiente, de forma que tareas complejas como la carga constante de *chunks*, es decir, fragmentos de terreno, no supongan la necesidad de un equipo con altas prestaciones. Para ello, debe cumplirse lo siguiente:

- **Diseño de modelos en Blender, un programa de edición 3D:** Diseñar los modelos de los astronautas y los alienígenas, además de sus animaciones para los diversos eventos que se darán en el transcurso del juego.
- **Diseño del terreno:** Diseño de programas para lograr lo siguiente:
 - La generación del planeta es procedural y usa el ruido de Perlín. Su generación puede ser ligeramente configurada por el usuario, de forma que pueda elegir su tamaño y semilla.
 - Diseñar un terreno que se genere por *chunks*, para fragmentar el número de triángulos renderizados por parte de Unity3D. Los *chunks* que no vea la cámara principal no serán renderizados.
 - Estos *chunks* tendrán un *Nivel de Detalle* (LOD) dinámico, que significa que serán capaces de reajustar su número de triángulos según haga falta para mejorar el rendimiento de la aplicación.
- **Diseño de texturas para el ratón:** El ratón tiene un conjunto de texturas personalizadas, que se irán cambiando en función de la dirección y velocidad de este.
- **Implantación de IA:** Las entidades de **Space Hunt** están todas controladas por diferentes sistemas inteligentes. Para el caso de los astronautas, en su primera fase (exploración) usan el *Particle Swarm Optimization* (PSO) mientras que para la batalla usan suma de vectores para la toma de decisión de la dirección y distancia mínima para el ataque. Para el caso de los alienígenas, usan *PSO* para la búsqueda de los astronautas y distancia mínima para el ataque.
- **Sonido:** El sonido usado en **Space Hunt** se divide en banda sonora y sonidos de las entidades:
 - **Banda sonora:** La banda sonora del juego proviene de creaciones de audio usando el programa FL Studio. Las bandas sonoras irán alternándose aleatoriamente. Cuando llega el momento de la batalla final, empezará a sonar un segundo conjunto de bandas sonoras de estilo Hip Hop.
 - **Sonidos de las entidades:** Las entidades emitirán un sonido cada vez que den un paso. Cuanto más rápido caminen, mayor será la frecuencia de emisión de estos sonidos.

1.3. Alcance

Space Hunt es un videojuego 3D de un único jugador, disponible para cualquier usuario que posea un ordenador con el sistema operativo Windows. Este videojuego dispone de las siguientes funcionalidades:

- Opciones de generación del planeta. Se puede elegir la dimensión del planeta que se generará cuando se pulse el botón de **Play**. Hay tres tamaños a elegir: 100, 200 y 400 unidades. También se puede introducir la semilla de generación del planeta.

- Generación procedural del planeta. El planeta está compuesto de una serie de montañas y valles que son generadas proceduralmente usando el ruido de Perlín para mantener la consistencia entre distintos *chunks*.
- Ajuste inteligente de los *chunks*. Dependiendo del zoom que esté haciendo el usuario, se renderizarán más o menos *chunks* para evitar el renderizado de partes del terreno que no ve la cámara. Además, hacer zoom alterará el nivel de detalle de estos *chunks*. Estos cambios suponen una mejora del rendimiento del juego, ya que se evita el renderizado y carga de triángulos en la escena.
- Agentes controlados por inteligencia artificial. Tanto los astronautas como los alienígenas están controlados por diversos algoritmos que permiten su control autónomo sin necesidad de intervención del usuario. Entre estos, destaca el *PSO*, usado para la exploración del planeta y la búsqueda del punto más alto.

1.4. Glosario

1.4.1. Acrónimos

Los acrónimos usados durante el transcurso de la memoria se describen a continuación.

- **PSO.** Particle Swarm Optimization.
- **IA.** Inteligencia Artificial.
- **LOD.** Level Of Detail.
- **NPC.** Non-Player Character.
- **RTS.** Real Time Strategy.
- **UML.** Unified Modeling Language.

1.4.2. Definiciones

- **Generación procedural.** Método de generación de contenido a través de algoritmos, en oposición a un método de creación manual, y se lo aplica tanto en simulaciones gráficas por computadora como en videojuegos, instalaciones, programación y en música.
- **Agente inteligente.** Entidad capaz de percibir su entorno, procesar la información recibida y actuar en consecuencia de manera racional, intentando maximizar algún parámetro.

1.5. Contenido de la memoria

La memoria actual está dividida en una serie de capítulos en los que se describen diferentes partes del proyecto, pudiendo estar cada una de ellas, asimismas, divididas en más partes. Estas son las siguientes.

1. **Planteamientos y Objetivos:** Descripción de los objetivos, motivación que me ha impulsado a realizar el proyecto y también una descripción de la distribución de este documento.
2. **Estado del Arte:** En este punto se describe el estado del arte de los pilares básicos sobre los que se sustenta el proyecto.
3. **Planificación:** Descripción de las metodologías empleadas para realizar el proyecto y de la distribución de las partes en las que este se ha subdividido durante su desarrollo.
4. **Requisitos:** Descripción de las necesidades y requisitos funcionales y no funcionales del proyecto.
5. **Análisis:** Análisis del sistema, describiendo sus casos de uso y modelo conceptual acompañado de diagramas UML.
6. **Diseño:** Descripción de la arquitectura física, diseño físico de datos y del diseño detallado de la interfaz de usuario.
7. **Construcción del Sistema:** Se explica la toma de decisiones a nivel de diseño, además de la explicación del funcionamiento de los algoritmos troncales del proyecto.
8. **Pruebas:** Descripción de las pruebas realizadas en el transcurso del proyecto, para asegurar su funcionamiento correcto.
9. **Conclusiones y Trabajo Futuro:** En este capítulo se exponen los problemas encontrados en el transcurso del desarrollo del proyecto, además de sus conclusiones y algunas ideas para su trabajo futuro.
10. **Anexos:** En los anexos se incluye el manual de instalación y el manual de usuario, en el que se describirá al mismo la funcionalidad de los elementos del proyecto y cómo usarlos.

Capítulo 2

Estado del Arte

2.1. Inteligencia artificial en videojuegos

2.1.1. Introducción

En los videojuegos, la inteligencia artificial se ha usado para generar comportamientos que se adapten de forma automática o inteligente, principalmente en los *Non-Player Characters* (NPCs). La inteligencia artificial ha formado una parte esencial en los juegos desde su inicio en la década de 1950. [2]

El papel de la *IA* en los videojuegos se ha ampliado bastante desde su introducción. Los juegos modernos a menudo implementan técnicas existentes en el campo de la *IA*, como el (búsqueda del camino óptimo) y los árboles de decisión para guiar las acciones de los *NPCs*.

Además de todo esto, la *IA* también puede implementarse en los videojuegos para el desarrollo de mecanismos que no son visibles de forma inmediata para el usuario, como la minería de datos y la generación de contenido procedural.

2.1.2. Historia

Los videojuegos como tal comenzaron siendo un área de investigación en *IA*. Uno de los primeros ejemplos de la *IA* es el juego **Nim**, creado en 1951 y publicado un año más tarde. El juego consiste en lo siguiente:

Dos jugadores colocan un número arbitrario de fichas, que pueden ser cualquier elemento, sobre una superficie. Estas fichas están divididas en grupos. El primer jugador toma un número arbitrario de fichas de un grupo. Solo se pueden coger fichas de un grupo. El segundo jugador realiza una jugada similar. Gana el jugador que retira la última ficha.

Como se puede ver en la figura 2.1, cada fila de cerillas representaría un grupo.

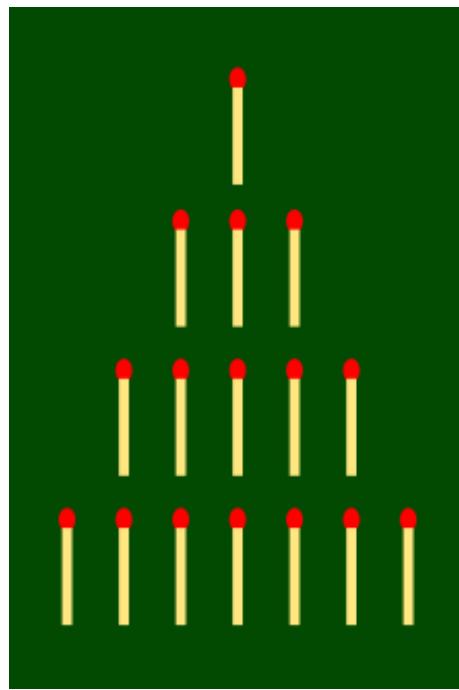


Figura 2.1: Juego Nim

Esencialmente, este juego es un árbol de decisiones. Aquí es donde la *IA* empezó a ganar interés. En 1951, utilizando la máquina *Ferranti Mark 1* de la Universidad de Manchester, Christopher Strachey escribió un programa de damas y Dietrich Prinz escribió uno para ajedrez.

Estos programas estuvieron entre los primeros que jamás se escribieron para los computadores. Los algoritmos utilizados aún no tenían un alto nivel de *IA*. La *IA* empezó a ganar habilidad cuando Arthur Samuel desarrolló un programa de damas a mediados de los años 50, que fue capaz de desafiar a un aficionado respetable.

Los juegos de un único jugador con enemigos comenzaron a aparecer en la década de 1970. Ejemplos importantes son *Taito Speed Race* (1974), un videojuego de carreras, *Atari Qwak*, un juego de caza de patos y *Pursuit*, un simulador de combates de aviones. El movimiento de las entidades enemigas se basaban, en aquel entonces, en patrones almacenados. La incorporación de microprocesadores permitiría un mayor nivel de cómputo y elementos aleatorios superpuestos en patrones de movimiento.

Más tarde llegó la era de oro de los videojuegos, y fue cuando la idea de los oponentes con *IA* se popularizó. Esto fue, en gran parte, por el éxito de *Space Invaders* (2.2) en 1978. Este juego representaba un nivel de dificultad creciente y distintos patrones de movimiento.

Galaxian (1979) agregó movimientos enemigos más complejos y variados, incluidas las maniobras de enemigos individuales que escapan de la formación. En 1980, *Pac-Man* in-

trodujo los patrones de *IA* en los juegos de laberinto. Estos patrones se fueron extendiendo a más tipos de juegos, como los de lucha (*Karate Champ*, en 1984).

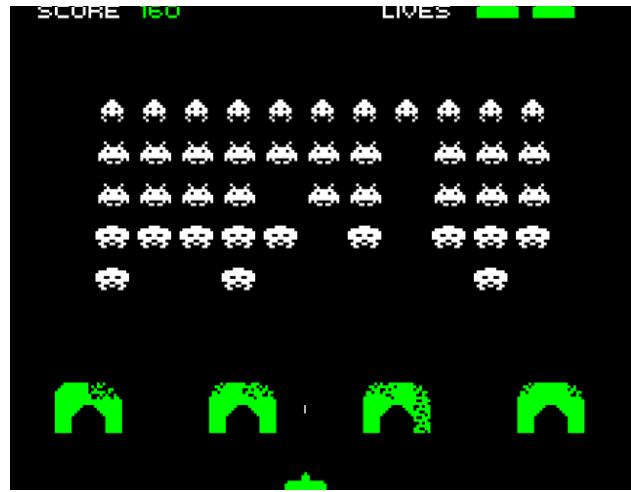


Figura 2.2: Juego Space Invaders

2.1.3. Videojuegos modernos

A día de hoy, la *IA* es usada ampliamente en bastantes juegos. No solo eso, si no que es usada en una amplia variedad de campos dentro de cada uno de estos juegos. El más obvio está en el control de cualquier NPC en el juego.

Los algoritmos de *pathfinding* son otro uso común de la *IA* a día de hoy, y se ve ampliamente en los juegos de estrategia en tiempo real. *pathfinding* es el método para determinar como obtener el camino de una entidad que desea ir de un punto a otro en el mapa. Estos algoritmos tienen en cuenta el mapa y los obstáculos. Algunos algoritmos interesantes de *pathfinding* son el algoritmo *A** y el *PSO*. [3] [4]

2.1.3.1. Algoritmo *A**

Este algoritmo fue presentado por Peter E. Hart, Nils J. Nilsson y Bertram Raphael en el año 1968, y se clasifica dentro de los algoritmos de búsqueda en grafos. Su objetivo es encontrar el camino de menor costo entre un nodo origen y uno objetivo, teniendo en cuenta posibles encuentros con obstáculos en el camino (nodos a los que el jugador no puede acceder). Se puede ver un ejemplo de su aplicación en la figura 2.3.

El algoritmo utiliza una función de evaluación $f(n) = g(n) + h(n)$. $g(n)$ indica la distancia del camino desde el nodo origen s a n . $h(n)$ expresa la distancia estimada desde el nodo n hasta el nodo destino t .

$h(n)$ se trata de una función heurística, luego expresa una estimación de cómo de lejos está el destino. La elección de esta función dependerá del problema a aplicar el algoritmo A^* .

7	6	5	6	7	8	9	10	11		19	20	21	22
6	5	4	5	6	7	8	9	10		18	19	20	21
5	4	3	4	5	6	7	8	9		17	18	19	20
4	3	2	3	4	5	6	7	8		16	17	18	19
3	2	1	2	3	4	5	6	7		15	16	17	18
2	1	0	1	2	3	4	5	6		14	15	16	17
3	2	1	2	3	4	5	6	7		13	14	15	16
4	3	2	3	4	5	6	7	8		12	13	14	15
5	4	3	4	5	6	7	8	9	10	11	12	13	14
6	5	4	5	6	7	8	9	10	11	12	13	14	15

Figura 2.3: Ejemplo de aplicación del algoritmo A^*

El algoritmo tiene el siguiente esquema:

1. Se establece el nodo s como el origen. Por lo tanto, $f(s) = 0$ y $f(i) = \infty, \forall i \neq s$. Se inicializa un conjunto Q a \emptyset .
2. Calcular el valor de $f(s)$ y añadir s al conjunto Q .
3. Seleccionar el nodo i con menor valor $f(i)$ del conjunto Q y eliminarlo del conjunto.
4. Analizar los nodos vecinos j de i . Para cada enlace (i, j) con coste c_{ij} hacer:
 - a) Calcular $f(j)' = g(i) + c_{ij} + h(j)$
 - b) Si $f(j)' < f(j)$:
 - 1) Actualizar $f(j)$ a $f(j) = g(i) + c_{ij} + h(j)$
 - 2) Insertar nodo j en Q
5. Si Q está vacío, el algoritmo acaba. Si no, volver al punto 3.

2.1.3.2. Particle Swarm Optimization (*PSO*)

Este algoritmo está inspirado en la interacción social en especies animales. Un determinado número de agentes (partículas) simulan el movimiento de una población en búsqueda de la mejor solución.

Cada partícula se trata como un punto en un espacio N-dimensional que ajusta su movimiento en cada iteración en función de su propio movimiento (inercia) y del de las demás partículas, por esto se considera un algoritmo social.

Estas partículas buscan una mejor posición global. La función que determina que una posición es mejor que otra es la función de evaluación. Esta función dependerá del problema. Por ejemplo, en un algoritmo donde se busca el punto más alto de una superficie irregular, la función de evaluación de un punto en esta superficie sería la altitud en ese punto.

Los elementos fundamentales en este algoritmo y que serán la clave para la actualización de la posición en cada iteración de las partículas son los siguientes:

- **Mejor puntuación personal, $pbest$:** Cada partícula guarda las coordenadas de su mejor posición, es decir, de la posición en la que se alcanzó una mejor función de evaluación hasta ese instante.
- **Mejor puntuación global, g_{best} :** Este valor es el mejor valor obtenido por todas las partículas. Tiene gran repercusión en la toma de decisiones de las partículas.

Esencialmente, las partículas decidirán en cada iteración hacia donde dirigirse cambiando su velocidad con el objetivo de que encuentren el máximo global. Aún así, pueden converger en un máximo local dado que el algoritmo *PSO* no tiene muchos mecanismos de escape de óptimos locales.

Una partícula x_i se mueve hacia la mejor posición en el espacio de búsqueda recordando la mejor posición particular alcanzada, p_i y la mejor posición global de la población, g como se puede ver en la figura 2.4.

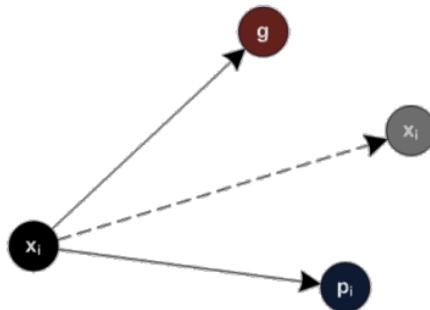


Figura 2.4: Actualización de la velocidad de una partícula

El algoritmo tiene el siguiente esquema:

1. Generación de una población aleatoria de partículas.
2. Cálculo de la función de evaluación $f(n)$ de cada partícula.
3. Cálculo de $pbest$ y g_{best} .
4. Actualización de la velocidad y posición de cada partícula, en función de:
 - a) Posiciones actuales S_i^k

- b) Velocidades actuales V_i^k :
- c) Distancia entre la posición actual y $pbest$ ($S_{pbest_i} - S_i^k$)
- d) Distancia entre la posición actual y $gbest$ ($S_{gbest} - S_i^k$)

Los parámetros principales se actualizan de acuerdo a las siguientes fórmulas:

$$V_i^{k+1} = wV_i^k + c_p r_p (S_{pbest_i} - S_i^k) + c_g r_g (S_{gbest} - S_i^k)$$

$$S_i^{k+1} = S_i^k + V_i^{k+1}$$

Donde k es la iteración, S_i es la posición de la partícula, V_i es la velocidad de la partícula y, por consiguiente, S_{gbest} es la mejor posición global y S_{pbest_i} es la mejor posición para la partícula i .

La constante de inercia, w , define la importancia de la dirección de la partícula en cada iteración. A mayor w , mayor es la probabilidad de escapar de los óptimos locales. Las constantes de aceleración (c_p , c_g) definen la importancia de las mejores posiciones para cada una de las partículas frente a la mejor posición global sobre la ecuación de velocidad. Finalmente, r_p y r_g son números aleatorios que añaden aleatoriedad a esta ecuación, cosa que también ayuda a evitar los óptimos locales.

2.2. Renderizado de grandes mapas

Una de las grandes problemáticas del desarrollo de videojuegos en los que el mapa en el que se desarrolla la acción es grande (por ejemplo, los videojuegos *RTS*) es la generación y renderizado del mismo.

Una de las alternativas a la generación de todo el terreno en la primera instancia del videojuego para lograr mayor rendimiento son la generación procedural del mismo o el control del nivel de detalle del terreno en función de la posición del jugador principal.

2.2.1. Generación procedural del mapa

La generación procedural es un método de creación de contenidos a través de diversos algoritmos, en oposición a la creación manual de estos contenidos. [5] [6] [7]

Proteus, *No Man's Sky*, *Nowhere* o *Minecraft* son ejemplos de juegos 3D en los que el escenario donde se desarrollan se genera de manera procedural. Explorar los mapas dentro de estos juegos significa excitar la generación de los mismos por medio de algoritmos procedurales.

En la figura 2.5 se puede observar una gran porción del mapa que ya ha sido generado de forma procedural, mientras que las regiones grises aún no han sido exploradas. A medida que el jugador se acerque a esas regiones, el contenido se generará de forma que

mantenga correlación con su entorno, creando nuevo contenido a partir de los algoritmos de generación.

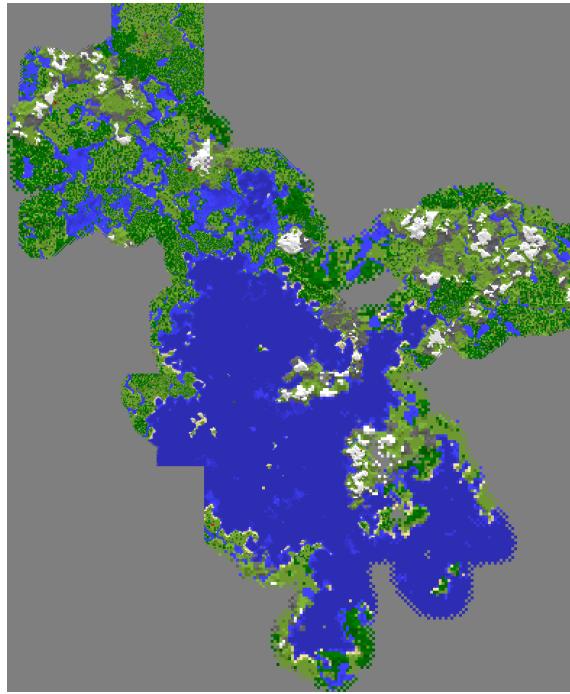


Figura 2.5: Generación de contenidos en el mapa de Minecraft

2.2.2. Renderizado con nivel de detalle cambiante (*LOD*)

Para renderizar mucho terreno en el mapa de un videojuego, se pueden emplear diversas técnicas basadas en el *LOD*. El objetivo de cambiar el *LOD* de un terreno (puede ser una porción de mapa o *Chunk*) es reducir el número de polígonos que conforman esta porción, reduciendo también el nivel de carga y la complejidad de procesos acoplados a los polígonos como, por ejemplo, el sombreado. Al reducir el *LOD* de un terreno, también se intenta mantener el detalle del mismo. Con esto, se consigue un aumento sustancial en el rendimiento del videojuego. [8]

Como se puede observar en la figura 2.6, en esta porción de terreno hay distintos niveles de detalle. Este nivel aumenta o disminuye según le interese al diseñador del videojuego. Por ejemplo, hay porciones con más información que no interesa que se poligonen demasiado o se verán pixeladas, mientras que hay porciones más monótonas o continuas como una meseta o el agua en las que el nivel de detalle puede ser reducido significativamente sin perder mucha información visual, y ganando rendimiento.

Otra forma de aplicar los algoritmos de reescalado del *LOD* a diferentes porciones del terreno es en función de la lejanía del jugador con respecto a este terreno. Cuanto más

lejos esté de él, a pesar de que el terreno contenga mucha información, ocupará menos espacio en la pantalla y, por consiguiente, no le interesaría observarlo con tanto detalle. Sin embargo, las porciones de terreno más próximas al usuario interesa que se rendericen con mayor nivel de detalle, sacrificando rendimiento, con el fin de que el usuario pueda disfrutar del 100 % de la calidad gráfica que pueda ofrecer el videojuego.

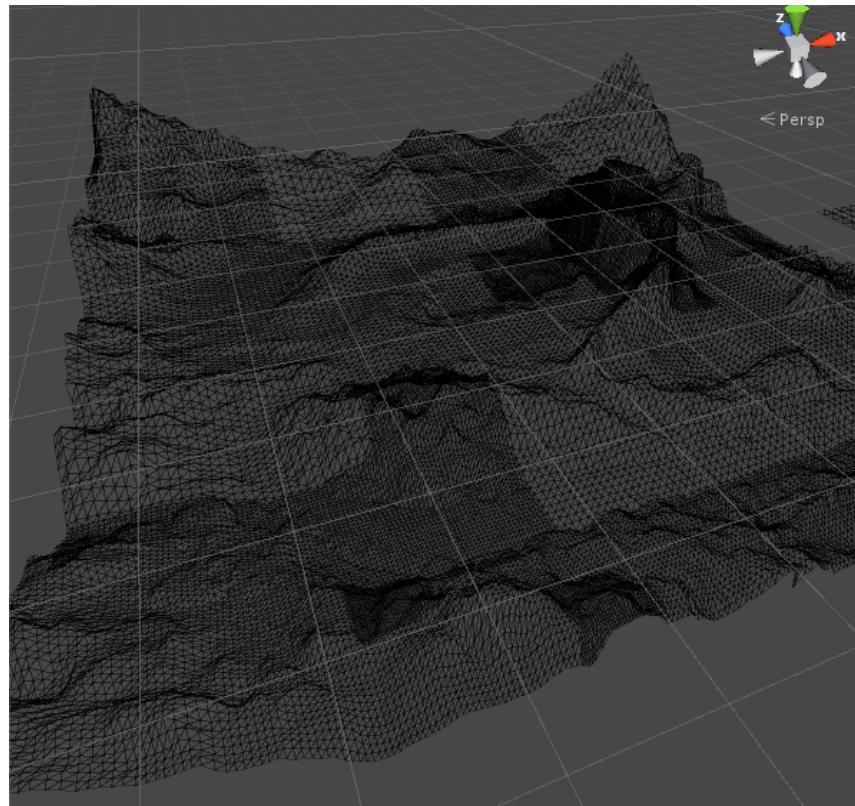


Figura 2.6: Diferencia de *LOD* entre diferentes porciones de un mismo terreno

Capítulo 3

Planificación

En este capítulo, se describe la metodología usada para el desarrollo del producto descrito en este proyecto. Además, se explicará el tiempo dedicado a cada tarea dentro de su correspondiente Sprint en un diagrama de Gantt. Finalmente, se realiza un presupuesto de todo el proyecto.

3.1. Metodología de desarrollo

Para el desarrollo de **Space Hunt**, se ha seguido la metodología ágil Scrum. Es una metodología común en proyectos en los que el nivel de incertidumbre es alto. Dado que los requisitos de este proyecto evolucionan de forma muy cambiante a lo largo de su desarrollo, hace que la implementación de esta metodología sea la idónea.

El proyecto se ha dividido en grandes bloques dependientes entre sí, llamados Sprints. Cada uno de estos Sprints abarca una parte importante del producto, que será necesaria para el desarrollo del siguiente Sprint.

Cada uno de estos grandes bloques consta de un grupo de tareas más pequeñas, con el fin de que entre todas lleguen a cumplir el objetivo del Sprint. Estas tareas pueden ser o no paralelizables, es decir, pueden llegar a ser independientes entre sí.

Los Sprints se definen al inicio del desarrollo del producto, pero dada su propiedad dinámica, pueden cambiar a lo largo de este desarrollo. En la figura 3.1 se puede ver el esquema básico de un Sprint de la metodología ágil Scrum.

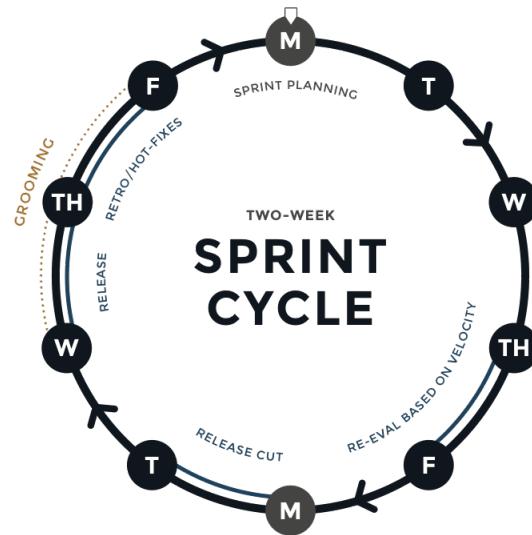


Figura 3.1: Esquema de un ciclo Sprint

3.2. Etapas

Para el desarrollo de este proyecto, ha sido necesaria su división en una serie de Sprints.

- **Sprint 1.** Se realiza un primer análisis del proyecto y de sus requisitos. El objetivo de este Sprint es reunir la documentación necesaria para el desarrollo de forma eficiente de las bases del juego por medio de la investigación. Una vez reunida esta documentación, será usada en los posteriores Sprints.
- **Sprint 2.** El objetivo de este Sprint es el desarrollo del terreno principal de **Space Hunt** usando *Unity 3D*. Este Sprint está dividido en las siguientes subtareas:
 - **Tarea 1.** Generación del planeta en forma de cubo de distintos tamaños por medio de la escritura de *scripts*.
 - **Tarea 2.** Esferificar el cubo que conforma el planeta. (Depende de Tarea 1)
 - **Tarea 3.** Generar distintas altitudes en el terreno por medio de la implementación del ruido de Perlín.
 - **Tarea 4.** Añadir configurabilidad a los parámetros del ruido de Perlín, como puede ser la semilla, frecuencia y amplitud. (Depende de Tarea 3)
- **Sprint 3.** El objetivo de este Sprint es la modularización del terreno para que renderice sus componentes de forma inteligente en función de la posición de la cámara del jugador. Este Sprint está dividido en las siguientes subtareas:

- **Tarea 1.** División del planeta en distintos *Chunks* (objetos independientes que conforman, en su totalidad, el planeta).
 - **Tarea 2.** Renderización y desrenderización de *Chunks* en función de la posición de la cámara del jugador. (Depende de Tarea 1)
 - **Tarea 3.** Implementación de diferentes niveles de *LOD* a cada uno de los *Chunks* mediante scripting. (Depende de Tarea 1)
- **Sprint 4.** El objetivo de este Sprint es el diseño del modelo del astronauta en Blender y de sus animaciones.
 - **Sprint 5.** El objetivo de este Sprint es la importación de los astronautas en Unity 3D, además de la posterior implementación de una serie de algoritmos para aplicar tanto la gravedad como la rotación correcta a estos astronautas en el planeta.
 - **Sprint 6.** El objetivo de este Sprint es el desarrollo de la inteligencia artificial de los astronautas. Este Sprint está dividido en las siguientes subtareas:
 - **Tarea 1.** Implementación del algoritmo *PSO* para la primera fase de exploración de los astronautas.
 - **Tarea 2.** Implementación de una serie de algoritmos de combate para la última fase de los astronautas (batalla contra los alienígenas).
 - **Sprint 7.** El objetivo de este Sprint es el diseño del modelo del alienígena en Blender y de sus animaciones.
 - **Sprint 8.** El objetivo de este Sprint es la importación de los alienígenas en Unity 3D, además de la reutilización de los algoritmos de aplicación de gravedad y rotación de los astronautas a estos nuevos modelos.
 - **Sprint 9.** El objetivo de este Sprint es el desarrollo de la inteligencia artificial de los alienígenas. Este Sprint está dividido en las siguientes subtareas:
 - **Tarea 1.** Implementación del algoritmo *PSO* para la primera fase de exploración de los alienígenas.
 - **Tarea 2.** Implementación de una serie de algoritmos de combate para la última fase de los alienígenas (batalla contra los astronautas).
 - **Sprint 10.** El objetivo de este Sprint es el diseño de *scripts* para el manejo de la cámara.
 - **Sprint 11.** El objetivo de este Sprint es el diseño de los menús del juego y de la interfaz de usuario.
 - **Sprint 12.** El objetivo de este Sprint es el desarrollo e implementación de la música de fondo y efectos de sonido, además del diseño de los *scripts* que manejan estos datos.

- **Sprint 13.** El objetivo de este Sprint es el desarrollo de *scripts* para la escritura de logs, que sirven para su posterior análisis y generación de estadísticas.

3.3. Diagrama de Gantt

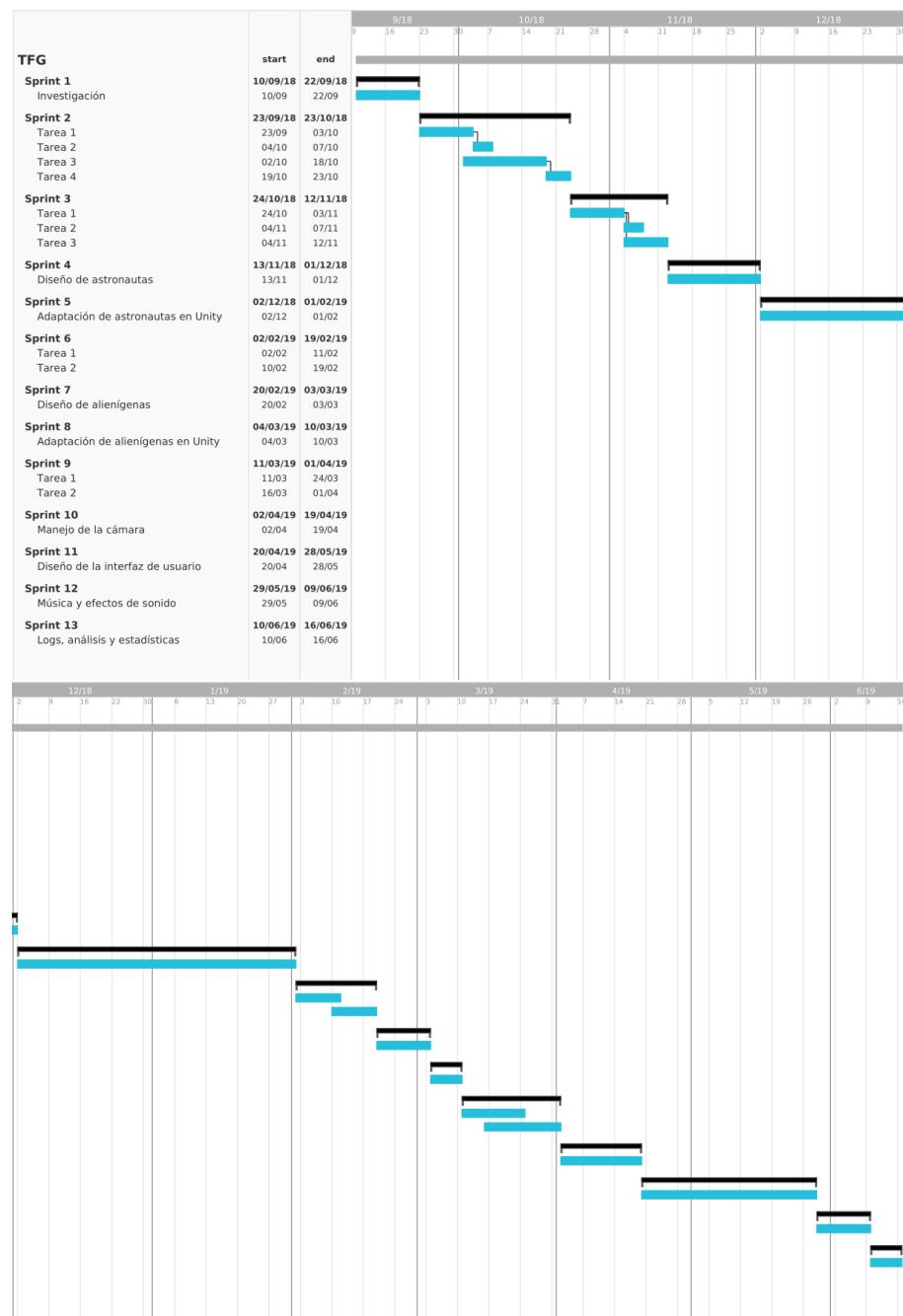


Figura 3.2: Diagrama de Gantt

3.4. Presupuesto

El proyecto se ha realizado a lo largo de 10 meses (desde septiembre del año 2018 hasta junio de 2019), trabajando una media de 5 días a la semana y 3,5 horas cada día.

Esto da un total de $225 \text{ días} \times 3,5 \text{ horas/día} = 787,5 \text{ horas}$ de trabajo en el *TFG*.

Un ingeniero informático junior cobra una media de 25000 euros al año (1800 horas), luego el precio de la mano de obra en el TFG es de $787,5 \times 25000/1800 = 10937,5$ euros.

A esto hay que sumarle un coste adicional, que sería el hardware en el que se ha desarrollado y probado el proyecto. La tabla de costes se puede ver en 3.1.

Descripción	Precio
Desarrollo del software (787,5 horas)	10.937,5 €
Computador sobre el que se ha desarrollado el proyecto	850 €
Total	11.787,5 €

Tabla 3.1: Costes

Por consiguiente, el presupuesto total del proyecto es 11.787,5 €.

Capítulo 4

Requisitos

A partir de los objetivos de *Space Hunt*, se definen una serie de requisitos que debe cumplir el software a desarrollar, llamados requisitos funcionales. Estos requisitos son declaraciones de los servicios que proveerá el software, de forma que se define la manera en que éste reaccionará a entradas particulares.

En cuanto a los requisitos no funcionales, se definirán aquellos que no se refieren directamente a las funciones específicas del sistema, sino a las propiedades emergentes de este.

4.1. Requisitos funcionales

RF 4.2.1	
Mover la cámara por el mundo	
Descripción	El sistema debe ser capaz de poder mover la cámara a donde le indique el usuario
Prioridad	Muy alta
Estado	Acabado
Dependencias	

Tabla 4.1: Requisito Funcional 4.2.1 - Mover la cámara por el mundo

RF 4.2.2	
Hacer zoom	
Descripción	El sistema debe ser capaz de aplicar distintos niveles de zoom sobre el mundo
Prioridad	Alta
Estado	Acabado
Dependencias	

Tabla 4.2: Requisito Funcional 4.2.2 - Hacer zoom

RF 4.2.3	
Cambiar la cámara a la posición de manejo global del mundo	
Descripción	El sistema debe ser capaz de cambiar el modo de manejo de la cámara a “global”
Prioridad	Alta
Estado	Acabado
Dependencias	RF 4.2.1

Tabla 4.3: Requisito Funcional 4.2.3 - Cambiar la cámara a la posición de manejo global del mundo

RF 4.2.4	
Cambiar cámara a la posición de un determinado astronauta	
Descripción	El sistema debe ser capaz de cambiar el modo de manejo de la cámara a “astronauta”, que se aplicará sobre el astronauta seleccionado
Prioridad	Alta
Estado	Acabado
Dependencias	RF 4.2.5

Tabla 4.4: Requisito Funcional 4.2.4 - Cambiar cámara a la posición de un determinado astronauta

RF 4.2.5	
Selección de astronautas	
Descripción	El sistema debe ser capaz de seleccionar un astronauta en particular, bien porque el usuario hace clic encima de él o lo selecciona en la barra lateral
Prioridad	Muy alta
Estado	Acabado
Dependencias	

Tabla 4.5: Requisito Funcional 4.2.5 - Selección de astronautas

RF 4.2.6	
Mostrar información de cada astronauta	
Descripción	El sistema debe ser capaz de mostrar información acerca de todos los astronautas en una barra lateral. Esta información contiene vida, id del astronauta, bandera y escudo o espada
Prioridad	Alta
Estado	Acabado
Dependencias	RF 4.2.20

Tabla 4.6: Requisito Funcional 4.2.6 - Mostrar información de cada astronauta

RF 4.2.7	
Indicar la defensa de un astronauta	
Descripción	El sistema debe ser capaz de dotar a un astronauta de la prioridad de ser defendido, por medio de un botón que podrá ser presionado por el usuario
Prioridad	Alta
Estado	Acabado
Dependencias	

Tabla 4.7: Requisito Funcional 4.2.7 - Indicar la defensa de un astronauta

RF 4.2.8	
Indicar el ataque a un alienígena	
Descripción	El sistema debe ser capaz de dotar a un alienígena de la prioridad de ser atacado, por medio de un botón que podrá ser presionado por el usuario
Prioridad	Alta
Estado	Acabado
Dependencias	

Tabla 4.8: Requisito Funcional 4.2.8 - Indicar el ataque a un alienígena

RF 4.2.9	
Asignar escudo a un astronauta	
Descripción	El sistema debe ser capaz de asignar a un determinado astronauta un escudo
Prioridad	Muy alta
Estado	Acabado
Dependencias	

Tabla 4.9: Requisito Funcional 4.2.9 - Asignar escudo a un astronauta

RF 4.2.10	
Asignar espada a un astronauta	
Descripción	El sistema debe ser capaz de asignar a un determinado astronauta una espada
Prioridad	Muy alta
Estado	Acabado
Dependencias	

Tabla 4.10: Requisito Funcional 4.2.10 - Asignar espada a un astronauta

RF 4.2.11	
Elegir el porcentaje de exploración de los astronautas	
Descripción	El sistema debe ser capaz de aplicar un porcentaje de exploración a los astronautas con el fin de que comiencen la primera fase del juego con ese valor configurado
Prioridad	Alta
Estado	Acabado
Dependencias	

Tabla 4.11: Requisito Funcional 4.2.11 - Elegir el porcentaje de exploración de los astronautas

RF 4.2.12	
Configurar la generación del mundo	
Descripción	El sistema debe ser capaz de configurar los parámetros de generación del mundo. Estos son la semilla y el tamaño del mundo. Una vez configurados generará proceduralmente el terreno acorde a ellos
Prioridad	Alta
Estado	Acabado
Dependencias	RF 4.2.19

Tabla 4.12: Requisito Funcional 4.2.12 - Configurar la generación del mundo

RF 4.2.13	
Configurar el volumen del juego	
Descripción	El sistema debe ser capaz de configurar tanto el volumen de los efectos de sonido dentro del juego como el de las bandas sonoras
Prioridad	Media
Estado	Acabado
Dependencias	RF 4.2.19

Tabla 4.13: Requisito Funcional 4.2.13 - Configurar el volumen del juego

RF 4.2.14	
Iniciar una partida	
Descripción	El sistema debe ser capaz de aplicar la configuración establecida por el usuario en los menús principales e iniciar una partida
Prioridad	Muy alta
Estado	Acabado
Dependencias	RF 4.2.12, RF 4.2.13

Tabla 4.14: Requisito Funcional 4.2.14 - Iniciar una partida

RF 4.2.15	
Pausa	
Descripción	El sistema debe ser capaz de pausar el juego en cualquier momento, una vez iniciado el mismo, y abrir el menú de pausa
Prioridad	Media
Estado	Acabado
Dependencias	RF 4.2.14

Tabla 4.15: Requisito Funcional 4.2.15 - Pausa

RF 4.2.16	
Salir del juego	
Descripción	El sistema debe ser capaz de salir del juego, desde el menú de pausa
Prioridad	Alta
Estado	Acabado
Dependencias	RF 4.2.15

Tabla 4.16: Requisito Funcional 4.2.16 - Salir del juego

RF 4.2.17	
Reproducir música	
Descripción	El sistema debe ser capaz de reproducir música, una vez iniciado el juego, que irá alternando de forma aleatoria
Prioridad	Alta
Estado	Acabado
Dependencias	RF 4.2.14

Tabla 4.17: Requisito Funcional 4.2.17 - Reproducir música

RF 4.2.18	
Reproducir efectos de sonido	
Descripción	El sistema debe ser capaz de reproducir efectos de sonido, emitidos por las entidades
Prioridad	Alta
Estado	Acabado
Dependencias	RF 4.2.14

Tabla 4.18: Requisito Funcional 4.2.18 - Reproducir efectos de sonido

RF 4.2.19	
Lectura de la configuración	
Descripción	El sistema debe ser capaz de leer los datos de la configuración por defecto en formato XML
Prioridad	Media
Estado	Acabado
Dependencias	

Tabla 4.19: Requisito Funcional 4.2.19 - Lectura de la configuración

RF 4.2.20	
Colocar una bandera a un astronauta	
Descripción	El sistema debe ser capaz de colocar una bandera a un determinado astronauta
Prioridad	Media
Estado	Acabado
Dependencias	

Tabla 4.20: Requisito Funcional 4.2.20 - Colocar una bandera a un astronauta

4.2. Requisitos no funcionales

RNF 4.3.1	
Fiabilidad	
Descripción	El sistema debe comportarse de acuerdo a sus especificaciones y producir el mínimo número de errores
Prioridad	Alta

Tabla 4.21: Requisito No Funcional 4.3.1 - Fiabilidad

RNF 4.3.2	
Eficiencia	
Descripción	El sistema debe rendir de la mejor forma posible a la hora de recibir las peticiones por parte del usuario y debe ser capaz de cargar el terreno de forma fluida
Prioridad	Alta

Tabla 4.22: Requisito No Funcional 4.3.2 - Eficiencia

RNF 4.3.3	
Accesibilidad	
Descripción	El sistema debe ser capaz de ejecutarse en una amplia gama de computadores, indistintamente del hardware que posean, gracias al requisito RNF 4.3.2, y en cualquier sistema Windows
Prioridad	Media

Tabla 4.23: Requisito No Funcional 4.3.3 - Accesibilidad

RNF 4.3.4	
Seguridad	
Descripción	El sistema debe ser capaz de ser seguro y de garantizar que los datos del usuario no son modificados por parte de terceros
Prioridad	Alta

Tabla 4.24: Requisito No Funcional 4.3.4 - Seguridad

Capítulo 5

Análisis

En esta sección se describirá el análisis del software desarrollado. Esto se realizará mediante el modelo de casos de uso y el modelo conceptual. Para representar las distintas partes de las que se compone el software, se usará el lenguaje unificado de modelado (UML).

5.1. Modelo conceptual

A partir de los requisitos del sistema se han generado una serie de diagramas de clases UML, divididos en *Menú Principal* e *Interfaz del Juego*.

5.1.1. Menú principal

En el siguiente grupo de figuras se describen los modelos conceptuales que representan el menú principal del juego.

El juego tiene un menú principal (5.1) controlado por la clase **MainMenu**, que está compuesta por tres botones. Estos botones son: **PlayButton**, que representa el botón que ejecutará la escena principal de la aplicación e iniciará el juego, **OptionsButton**, que llevará al usuario al menú de opciones del juego, y **QuitButton**, que representa el botón que dará fin a la ejecución de la aplicación.

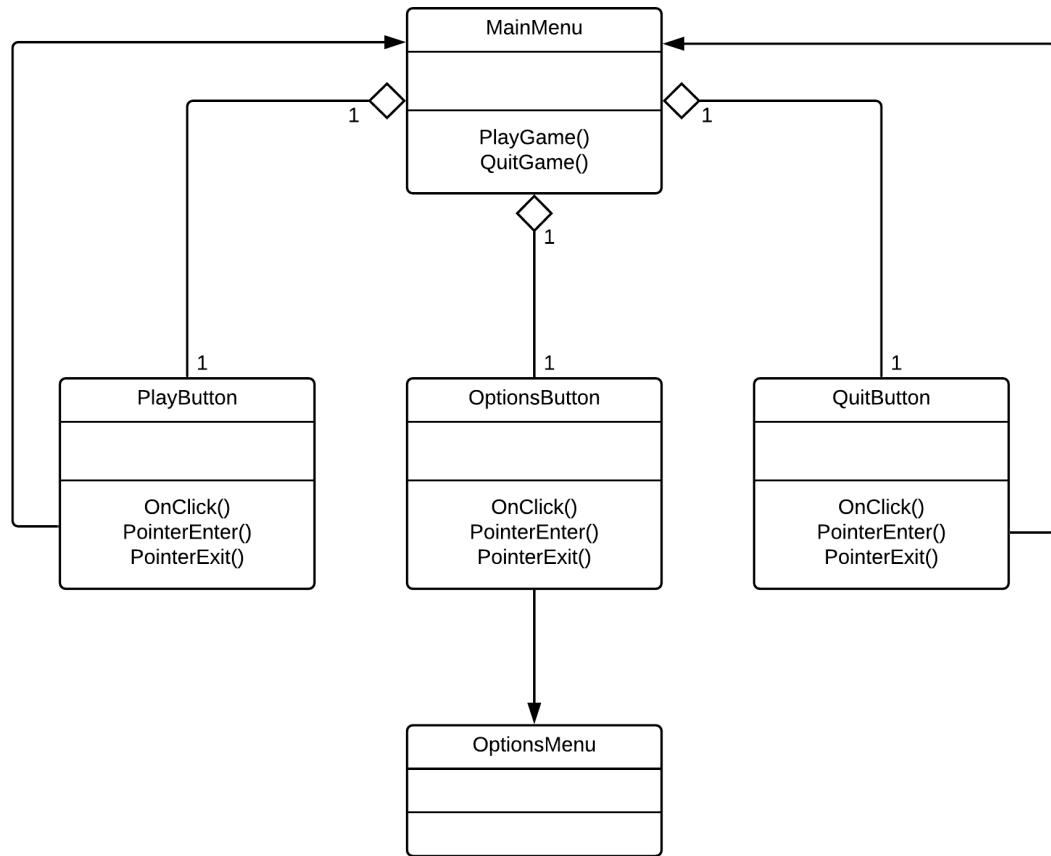


Figura 5.1: Diagrama Conceptual del Menú Principal

El botón **OptionsButton** cargará el menú de opciones, representado por el diagrama conceptual 5.2. Este menú está compuesto de tres botones: **VolumeButton**, que llevará al menú de control de volumen del juego (**VolumeMenu**), **GenerationButton**, que llevará al menú desde el cual se podrá configurar la generación del planeta, y **BackButton**, que representa el botón para volver al menú anterior, que en este caso es **MainMenu**.

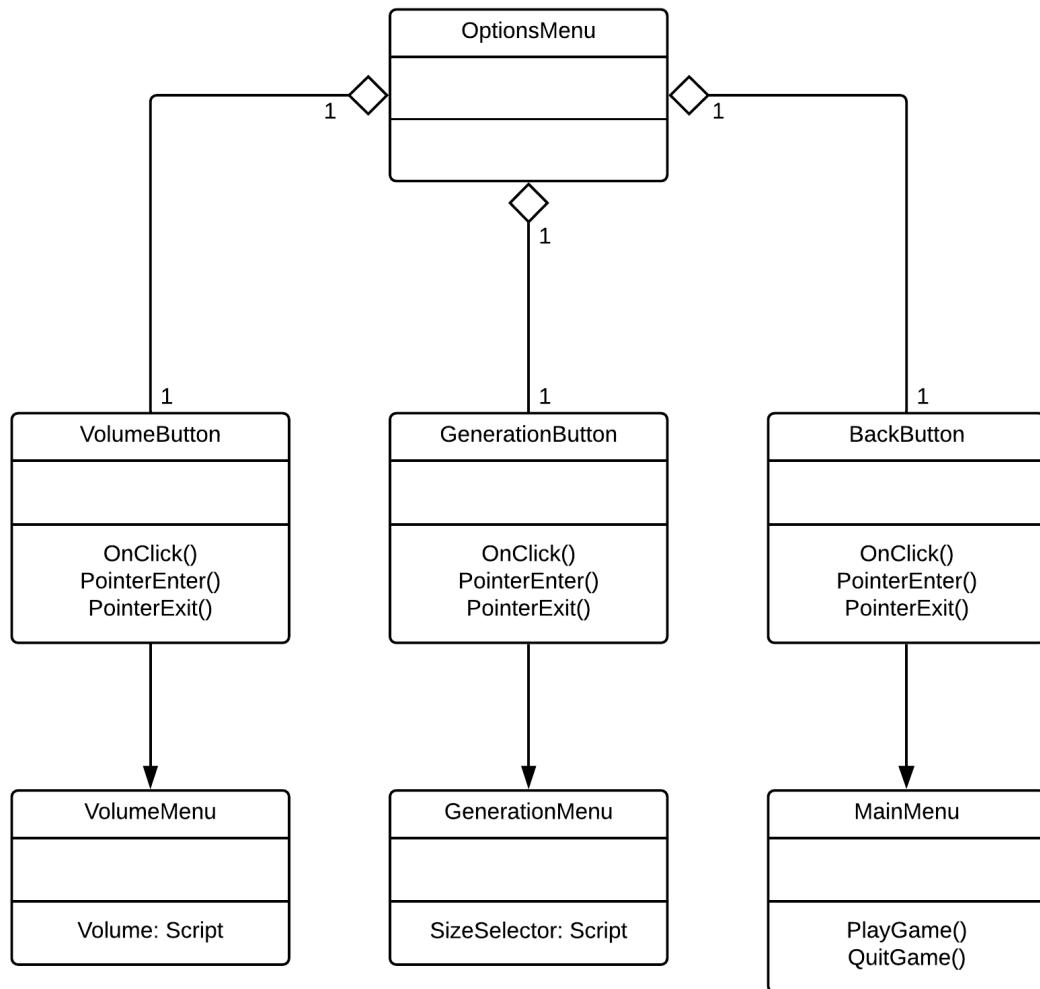


Figura 5.2: Diagrama Conceptual del Menú de Opciones

El botón **VolumeButton** cargará el menú de control de volumen, representado por el diagrama conceptual 5.3. Este menú está compuesto de tres botones: **MusicSlider**, que representa una barra horizontal desde la que el usuario podrá establecer el nivel de la música ambiental del juego, **SoundSlider**, que representa una barra horizontal desde la que el usuario podrá establecer el nivel de los sonidos producidos dentro del juego (como los pasos de los astronautas), y **BackButton**, que representa el botón para volver al menú anterior, que en este caso es **OptionsMenu**.

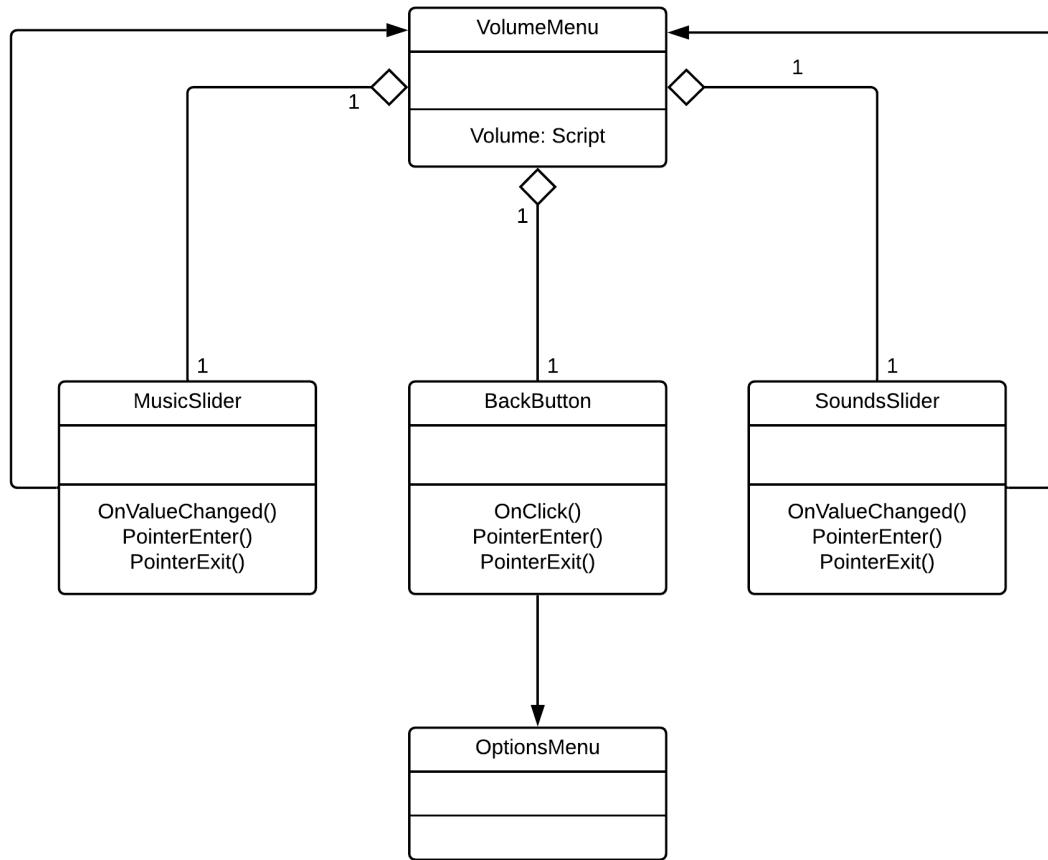


Figura 5.3: Diagrama Conceptual del Menú de Volumen

El botón **GenerationButton** cargará el menú de opciones de generación del planeta, representado por el diagrama conceptual 5.4. Este menú está compuesto de cinco botones: tres botones de selección del tamaño de generación del planeta (**Size100Button**, **Size200Button**, **Size400Button**), un campo para introducir la semilla del planeta representado por **SeedInputField**, y un botón para retroceder y volver al menú anterior (**BackButton**), que en este caso es **OptionsMenu**.

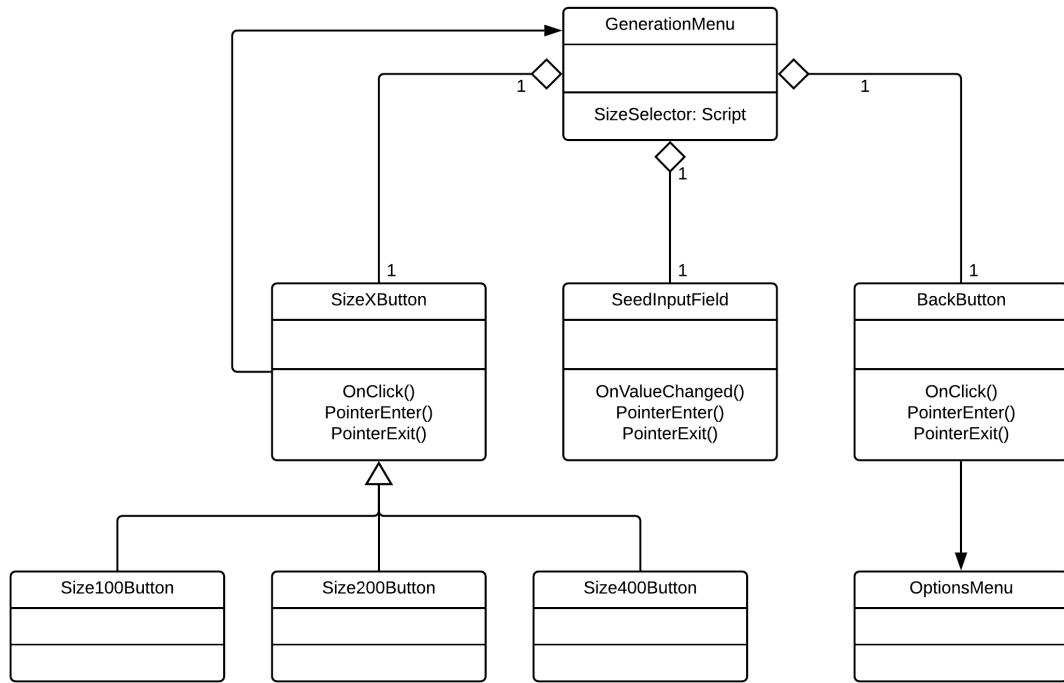


Figura 5.4: Diagrama Conceptual del Menú de Generación

En la siguiente figura (5.5), se puede apreciar el modelo conceptual de las interfaces relacionadas con el menú de pausa del juego. Este menú está compuesto por dos botones. **ResumeButton** se encarga de controlar que el juego vuelva a su ejecución normal y que desaparezca el menú de pausa, y **QuitButton** es el botón que cerrará la aplicación y pondrá fin al juego. Los menús **GameOverMenu** y **GameWonMenu**, que representan los menús en los que puede finalizar el juego (pantalla de partida ganada o perdida), también tienen el botón **QuitButton** para que el usuario pueda salir de la aplicación en todo momento.

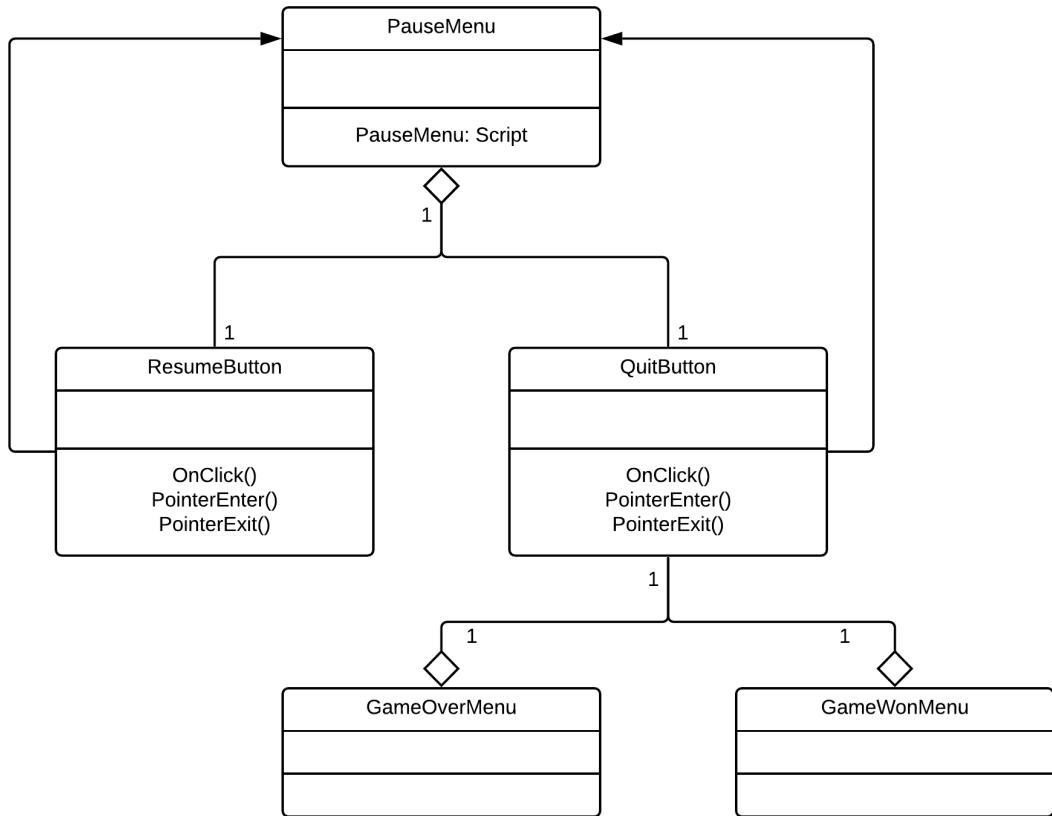


Figura 5.5: Diagrama Conceptual del Menú de Pausa

5.1.2. Juego

El diagrama conceptual del juego se puede dividir en tres partes independientes, como se puede ver en la figura 5.6. La primera es el menú de pausa, que se encarga de controlar que el juego se pause cuando el usuario lo solicite. La generación y actualización de los *chunks* que conforman el terreno están controlados por el *script* *CubeSphere* de *Sphere*. Este *script* también tiene control sobre la posición de la cámara. La última parte se encarga de controlar a las entidades que se moverán por el terreno generado. *AstronautManager* maneja a los astronautas, mientras que *AlienManager* maneja a los alienígenas.

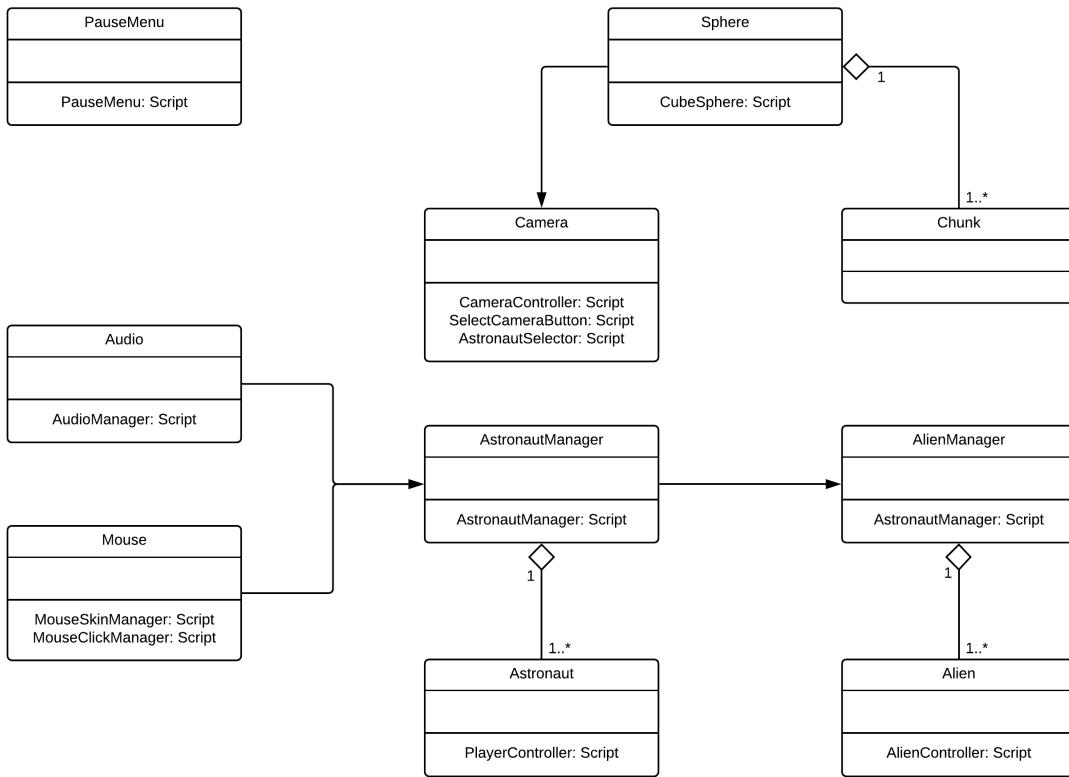


Figura 5.6: Diagrama Conceptual del Juego

En la siguiente figura (5.7), se puede apreciar el modelo conceptual de las interfaces relacionadas con la cámara del juego. **Camera** representa el objeto que controla la cámara del juego, gracias al script **CameraController**. **ChangeCameraButton** controla al botón que maneja el cambio de cámara entre el astronauta seleccionado y la posición de la cámara global, mientras que **AstronautButton** controla la selección de astronautas por parte del usuario.

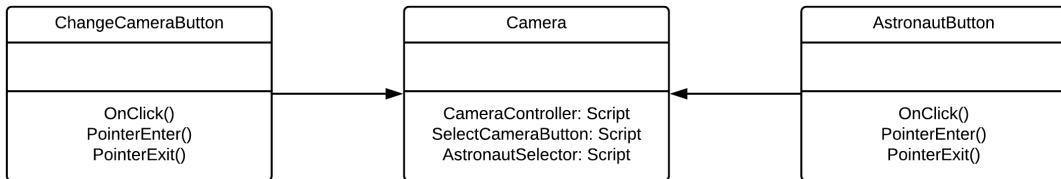


Figura 5.7: Diagrama Conceptual de la Cámara

En cuanto a la generación del mundo y a la rotación del Sol y de la oscuridad que tendrán

lugar a la hora de pulsar el botón “PLAY”, se especifican el el siguiente diagrama UML (5.8). **Sphere** hace referencia al control del planeta sobre el que ocurren los sucesos del juego. La generación del planeta está controlada por el *script* *CubeSphere*. **SunOrbit** y **NightOrbit** hacen referencia a los objetos que controlan el ciclo del día y de la noche en el juego.

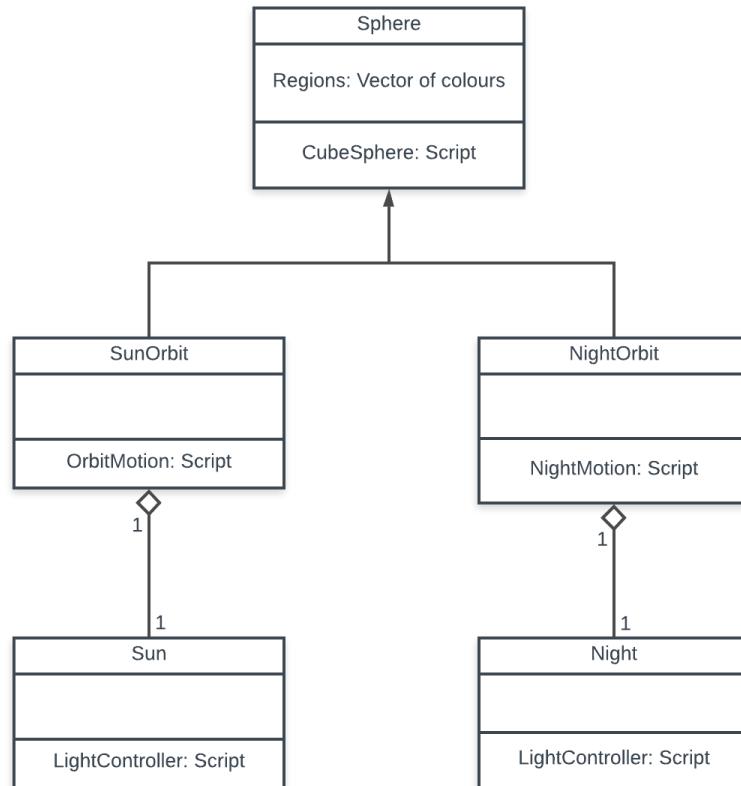


Figura 5.8: Diagrama Conceptual del Planeta

Finalmente, el UML que describe las interacciones entre los manejadores de entidades está descrito en la figura 5.9. **AstronautManager** se encarga de controlar las decisiones de los astronautas en cada momento, gracias al *script* *AstronautManager*. Lo mismo sucede con **AlienManager** y su *script* *AlienManager* para el control de los alienígenas. Estas entidades hacen uso continuo del sistema de audio (controlado por **Audio**) y de las entradas del ratón por parte del usuario (controladas por **Mouse**).

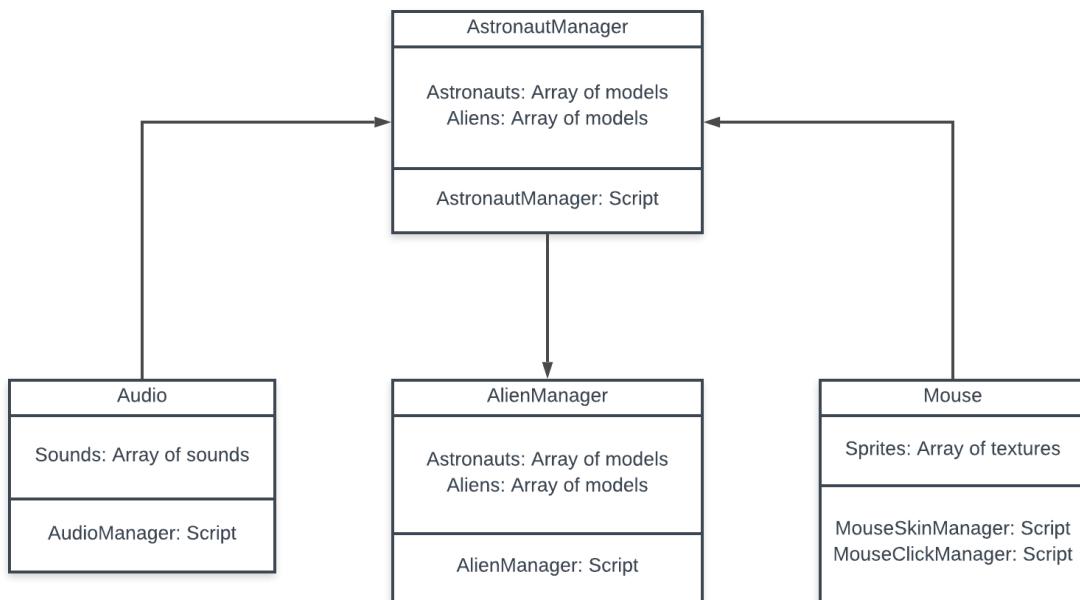


Figura 5.9: Diagrama Conceptual de las Entidades

5.2. Modelo Casos de Uso

5.2.1. Diagrama de Casos de Uso

El diagrama de casos de uso se ha creado a partir de los requisitos funcionales del sistema. Se muestra de forma detallada en 5.10.

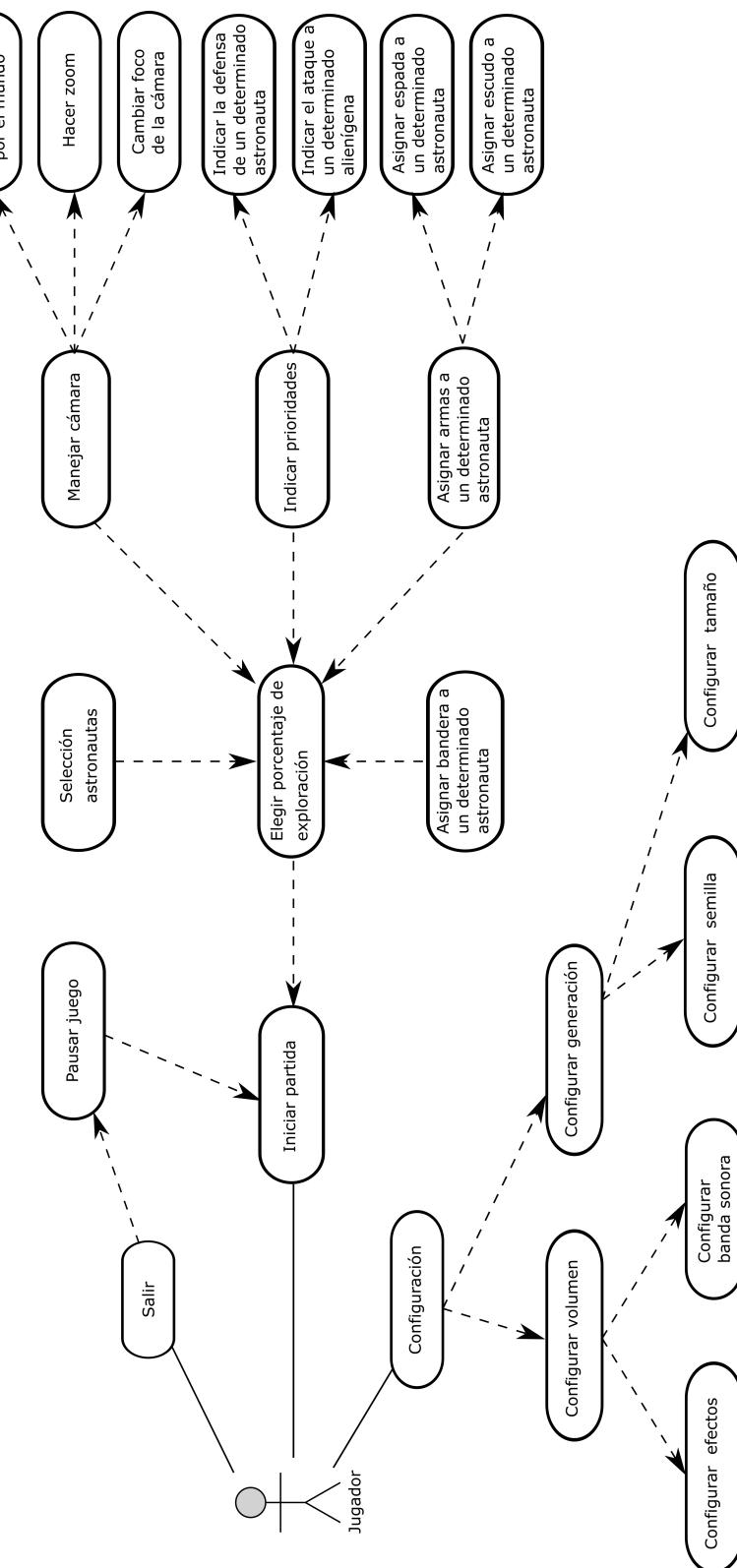


Figura 5.10: Modelo de Casos de Uso

5.2.2. Especificación de Casos de Uso

El sistema está diseñado para que lo controle un único actor, que es además el único jugador que soporta el software desarrollado. Se describe a continuación.

ATC-01	
Jugador	
Descripción	Único y principal usuario del sistema. Este jugador será el único que interactúe con el sistema.

Tabla 5.1: ATC-01 - Jugador

A continuación se especifican los casos de uso que componen la aplicación, con una estructura similar a la usada en la descripción del actor.

CU-01 Iniciar Partida	
Descripción	Botón que comienza a la partida cuando el usuario pulsa sobre él.
Actor	Jugador.
Precondiciones	El sistema se encuentra en el menú principal del juego.
Postcondiciones	El sistema inicia el juego con la configuración actual.
Escenario principal	<ol style="list-style-type: none"> 1. El usuario inicia la aplicación por primera vez. 2. El sistema muestra el menú principal, con una serie de opciones. Iniciar partida entre ellas. 3. El usuario presiona el botón “Jugar”. 4. El sistema inicia el juego con la configuración actual.

Tabla 5.2: CU-01 - Iniciar Partida

CU-02 Configuración	
Descripción	Botón que lleva al menú de configuración cuando el usuario pulsa sobre él.
Actor	Jugador.
Precondiciones	El sistema se encuentra en el menú principal del juego.
Postcondiciones	El sistema entra en el menú de configuración.
Escenario principal	<ul style="list-style-type: none"> 1. El usuario presiona el botón “Opciones”. 2. El sistema muestra el menú de las opciones del juego.

Tabla 5.3: CU-02 - Configuración

CU-03 Salir	
Descripción	Botón que finalizará el juego.
Actor	Jugador.
Precondiciones	El sistema se encuentra en el menú principal del juego o en el menú de pausa.
Postcondiciones	El sistema sale del juego, finalizándolo.
Escenario principal	<ul style="list-style-type: none"> 1. El usuario presiona el botón “Salir”. 2. El sistema sale del juego.

Tabla 5.4: CU-03 - Salir

CU-04 Pausar Juego	
Descripción	Acción que pausa el juego por completo.
Actor	Jugador.
Precondiciones	El sistema debe de haber iniciado el juego.
Postcondiciones	El sistema muestra el menú de pausa, congelando el juego mientras se mantenga en dicho menú.
Escenario principal	<ul style="list-style-type: none"> 1. El usuario presiona la tecla “Esc”. 2. El sistema muestra el menú de pausa, congelando el juego.

Tabla 5.5: CU-04 Pausar Juego

CU-05 Elegir porcentaje de exploración	
Descripción	Pequeño menú sobre el juego compuesto por un deslizador y un botón, en el que el jugador puede configurar el valor de exploración de los astronautas.
Actor	Jugador.
Precondiciones	El sistema debe de haber iniciado el juego.
Postcondiciones	El sistema da comienzo a la exploración de los astronautas con el valor elegido por el jugador.
Escenario principal	<ol style="list-style-type: none"> 1. El usuario desliza el slider al valor que considere oportuno. 2. El sistema actualiza el valor que selecciona el jugador por pantalla, mostrándolo en un porcentaje. 3. El usuario presiona el botón “Ok”. 4. El sistema aplica el valor seleccionado y da lugar a la exploración por parte de los astronautas. 5. El sistema elimina el menú de selección de porcentaje de exploración.

Tabla 5.6: CU-05 Elegir porcentaje de exploración

CU-06 Selección de un determinado astronauta	
Descripción	El sistema selecciona un determinado astronauta por medio de pulsar directamente encima suya o bien seleccionándolo en el menú de astronautas.
Actor	Jugador.
Precondiciones	El sistema debe de haber iniciado el juego y se debe haber elegido el porcentaje de exploración.
Postcondiciones	El sistema selecciona un determinado astronauta.
Escenario principal	<ol style="list-style-type: none"> 1. El usuario pincha encima de un astronauta o lo selecciona usando el menú vertical de los astronautas. 2. El sistema selecciona el astronauta elegido por el usuario.

Tabla 5.7: CU-06 Selección de un determinado astronauta

CU-07 Asignar bandera a un determinado astronauta	
Descripción	El sistema selecciona le asigna una bandera a un determinado astronauta, indicando al jugador que es un astronauta destacado.
Actor	Jugador.
Precondiciones	El sistema debe de haber iniciado el juego y se debe haber elegido el porcentaje de exploración.
Postcondiciones	El sistema le asigna una bandera a un determinado astronauta.
Escenario principal	<ol style="list-style-type: none"> 1. El usuario pincha en el icono “bandera” de un determinado astronauta. 2. El sistema le asigna una bandera al astronauta elegido por el usuario. Si ese astronauta ya tenía asignada una bandera antes, se la desasignará.

Tabla 5.8: CU-07 Asignar bandera a un determinado astronauta

CU-08 Mover cámara por el mundo	
Descripción	El sistema mueve la cámara a donde le indica el jugador dentro del mundo.
Actor	Jugador.
Precondiciones	El sistema debe de haber iniciado el juego y se debe haber elegido el porcentaje de exploración.
Postcondiciones	El sistema mueve la cámara a donde le indica el jugador dentro del mundo.
Escenario principal	<ol style="list-style-type: none"> 1. El usuario arrastra el ratón hacia la dirección que desea que se mueva la cámara. 2. El sistema se mueve hacia la dirección deseada por el usuario, manteniendo el zoom y ajustando la rotación de esta de forma que apunte al centro del mundo.

Tabla 5.9: CU-08 Mover cámara por el mundo

CU-09 Hacer zoom	
Descripción	El sistema aplica el zoom a la cámara que desea el usuario, dentro de los límites establecidos por el juego.
Actor	Jugador.
Precondiciones	El sistema debe de haber iniciado el juego y se debe haber elegido el porcentaje de exploración.
Postcondiciones	El sistema aplica el zoom a la cámara que desea el usuario, dentro de los límites establecidos por el juego.
Escenario principal	<ol style="list-style-type: none"> 1. El usuario mueve la rueda del ratón para hacer zoom in o zoom out hacia donde apunta la cámara. 2. El sistema aplica el zoom deseado. Si el usuario solicita un zoom fuera de los límites establecidos, este no se aplicará.

Tabla 5.10: CU-09 Hacer zoom

CU-10 Cambiar foco de la cámara	
Descripción	El sistema alterna el modo de la cámara entre el de foco del astronauta seleccionado y el de la cámara global.
Actor	Jugador.
Precondiciones	El sistema debe de haber iniciado el juego y se debe haber elegido el porcentaje de exploración.
Postcondiciones	El sistema alterna entre dos modos de cámara, la de selección de un astronauta y la global.
Escenario principal	<ol style="list-style-type: none"> 1. El usuario pulsa el botón “Cambiar cámara”. 2. El sistema evalúa el modo de cámara actual. <ol style="list-style-type: none"> a) Si es modo “astronauta”, se cambiará a modo global y los controles se pondrán en modo global (el usuario puede mover la cámara de sitio, además de poder efectuar zoom). La cámara no se mueve de sitio. b) Si es modo “global”, se cambiará a modo astronauta y los controles se pondrán en modo limitado (solo se puede hacer zoom, no se puede desplazar). La cámara se mueve a la posición del astronauta seleccionado en ese momento.

Tabla 5.11: CU-10 Cambiar foco de la cámara

CU-11 Indicar la defensa de un determinado astronauta	
Descripción	El sistema indica la defensa de un determinado astronauta con el fin de darle más prioridad en el momento de la batalla final del juego.
Actor	Jugador.
Precondiciones	El sistema debe de haber iniciado el juego, se debe haber elegido el porcentaje de exploración y se deben haber asignado armas a todos los astronautas.
Postcondiciones	El sistema le da más prioridad de defensa al astronauta seleccionado.
Escenario principal	<ol style="list-style-type: none"> 1. El usuario pulsa el botón “Defensa” arriba a la derecha en el juego. 2. El sistema cambia al modo de “defensa”, y cambiará la skin del ratón si este pasa por encima de un astronauta que posea espada. 3. El usuario pulsa encima de un astronauta con espada para darle más prioridad de defensa. 4. El sistema aplica la prioridad sobre el astronauta elegido y resetea la skin del ratón.

Tabla 5.12: CU-11 Indicar la defensa de un determinado astronauta

CU-12 Indicar el ataque a un determinado alienígena	
Descripción	El sistema indica el ataque a un determinado alienígena con el fin de darle más prioridad en el momento de la batalla final del juego.
Actor	Jugador.
Precondiciones	El sistema debe de haber iniciado el juego, se debe haber elegido el porcentaje de exploración y se deben haber asignado armas a todos los astronautas.
Postcondiciones	El sistema le da más prioridad de ataque a un determinado alienígena.
Escenario principal	<ol style="list-style-type: none"> 1. El usuario pulsa el botón “Ataque” arriba a la derecha en el juego. 2. El sistema cambia al modo de “ataque”, y cambiará la skin del ratón si este pasa por encima de un alienígena. 3. El usuario pulsa encima de un alienígena para darle más prioridad de ataque por parte de los astronautas con espada. 4. El sistema aplica la prioridad sobre los astronautas con espada contra el alienígena seleccionado y resetea la skin del ratón.

Tabla 5.13: CU-12 Indicar el ataque a un determinado alienígena

CU-13 Asignar espada a un determinado astronauta	
Descripción	El sistema le asigna una espada a un determinado astronauta elegido por el jugador.
Actor	Jugador.
Precondiciones	El sistema debe de haber iniciado el juego y se debe haber elegido el porcentaje de exploración.
Postcondiciones	El sistema le asigna una espada a un astronauta, elegido por el jugador.
Escenario principal	<ol style="list-style-type: none"> 1. El usuario pulsa el botón “Espada” en el menú de astronautas. 2. El sistema aplica la selección al astronauta elegido y muestra encima de él el ícono de una espada para notificar al jugador de su elección.

Tabla 5.14: CU-13 Asignar espada a un determinado astronauta

CU-14 Asignar escudo a un determinado astronauta	
Descripción	El sistema le asigna un escudo a un determinado astronauta elegido por el jugador.
Actor	Jugador.
Precondiciones	El sistema debe de haber iniciado el juego y se debe haber elegido el porcentaje de exploración.
Postcondiciones	El sistema le asigna un escudo a un astronauta, elegido por el jugador.
Escenario principal	<ol style="list-style-type: none"> 1. El usuario pulsa el botón “Escudo” en el menú de astronautas. 2. El sistema aplica la selección al astronauta elegido y muestra encima de él el ícono de un escudo para notificar al jugador de su elección.

Tabla 5.15: CU-14 Asignar escudo a un determinado astronauta

CU-15 Configurar efectos	
Descripción	Slider que permite al sistema configurar el volumen de los efectos producidos por las entidades del juego.
Actor	Jugador.
Precondiciones	El sistema se encuentra en el menú de configuración del volumen del juego.
Postcondiciones	El sistema configura el volumen de los efectos del juego.
Escenario principal	<ol style="list-style-type: none"> 1. El usuario modifica el slider de los efectos al nivel que desee. 2. El sistema aplica el nivel configurado por el usuario al juego.

Tabla 5.16: CU-15 Configurar efectos

CU-16 Configurar banda sonora	
Descripción	Slider que permite al sistema configurar el volumen de la banda sonora del juego.
Actor	Jugador.
Precondiciones	El sistema se encuentra en el menú de configuración del volumen del juego.
Postcondiciones	El sistema configura el volumen de la banda sonora del juego.
Escenario principal	<ol style="list-style-type: none"> 1. El usuario modifica el slider de la banda sonora al nivel que desee. 2. El sistema aplica el nivel configurado por el usuario al juego.

Tabla 5.17: CU-16 Configurar banda sonora

CU-17 Configurar semilla	
Descripción	Celda en la que el jugador puede introducir la semilla de generación del mundo que desee.
Actor	Jugador.
Precondiciones	El sistema se encuentra en el menú de configuración de generación del juego.
Postcondiciones	El sistema configura la semilla de generación del juego.
Escenario principal	<ol style="list-style-type: none"> 1. El usuario modifica la celda de la semilla de generación del mundo al nivel que desee. 2. El sistema aplica el valor configurado por el usuario al juego.

Tabla 5.18: CU-17 Configurar semilla

CU-18 Configurar tamaño	
Descripción	El jugador puede elegir un tamaño de una serie de tres botones para diferentes tamaños de generación.
Actor	Jugador.
Precondiciones	El sistema se encuentra en el menú de configuración de generación del juego.
Postcondiciones	El sistema configura el tamaño de generación del mundo.
Escenario principal	<ol style="list-style-type: none"> 1. El usuario pincha en uno de los tres botones del menú, <i>100, 200 o 400</i>. 2. El sistema aplica el valor configurado por el usuario al juego y cambia el color del botón seleccionado para notificar al usuario su elección.

Tabla 5.19: CU-18 Configurar tamaño

Capítulo 6

Diseño

En este apartado se describe la arquitectura del proyecto. La arquitectura física especifica los elementos que componen la infraestructura tecnológica necesaria para dar soporte al proyecto.

6.1. Arquitectura Física

La arquitectura física es aquella que define los elementos hardware que componen el sistema entero. En el caso de este proyecto, el único componente es el computador en el cual se ejecuta la aplicación, dado que es el único soporte que da el software de la aplicación.

Space Hunt puede ser jugado en cualquier computador con el sistema operativo Windows, en cualquiera de sus versiones a partir de *Windows 7* e independientemente de las especificaciones a nivel de hardware del computador. A mejores especificaciones del dispositivo, mejor será la experiencia del juego en términos de rendimiento, fluidez y tiempos de carga (para la generación de los mundos a la hora de comenzar a jugar).

6.2. Diseño Físico de Datos

Para manejar los datos del juego, estos siguen una estructura que describiré a continuación para mantener la consistencia en el modelo de datos del videojuego.

La configuración del juego se carga por medio de un archivo guardado en formato XML. Este archivo se encarga de portar la configuración en una primera instancia y llenar los ajustes por defecto, de forma que el usuario pueda jugar sin necesidad de tener que configurar estos ajustes.

La estructura de este documento es la siguiente.

```

1   <main>
2     <options>
3       <volume>
4         <music> 0.5 </music>
5         <sounds> 0.85 </sounds>
6       </volume>
7       <generation>
8         <size> 200 </size>
9         <seed> 139823 </seed>
10      </generation>
11    </options>
12    <pso>
13      <astronaut>
14        <c1> 1 </c1>
15        <c2> 1.5 </c2>
16      </astronaut>
17      <alien>
18        <c1> 1 </c1>
19        <c2> 1.5 </c2>
20      </alien>
21    </pso>
22    <entity>
23      <astronaut>
24        <maxSpeed> 5 </maxSpeed>
25      </astronaut>
26      <alien>
27        <maxSpeed> 2.5 </maxSpeed>
28        <maxDistanceToShoot> 20 </maxDistanceToShoot>
29      </alien>
30    </entity>
31    <world>
32      <heightMultiplier> 10 </heightMultiplier>
33      <noise>
34        <scale> 30 </scale>
35        <octaves> 4 </octaves>
36        <persistence> 0.5 </persistence>
37        <lacunarity> 1 </lacunarity>
38      </noise>
39    </world>
40  </main>

```

Figura 6.1: Configuración en formato XML

En cuanto a las entidades del juego, cada tipo de entidad posee un diseño de datos para que otros componentes del juego puedan interactuar y modificar sus estados internos.

El estado de un *astronauta* tiene la siguiente estructura de datos. **life** representa la vida actual del astronauta, **speed** representa la velocidad máxima a la que puede ir, **isDead** es un condicional que indica si está muerto o no, **hasSword** es un condicional que indica si el astronauta es un atacante (su arma es una espada) o no, y **hasShield** es un condicional que indica si el astronauta es un defensor (su arma es un escudo) o no.

```

1   astronaut : {
2       float life;
3       float speed;
4       bool isDead;
5       bool hasSword;
6       bool hasShield;
7   }

```

Figura 6.2: Estado de un *astronauta*

Sin embargo, el estado de un *alienígena* tiene una estructura de datos más simple. Tan solo necesita guardar su velocidad máxima, **speed**, su vida actual, **life**, y el condicional que indica si está muerto o no, **isDead**.

```

1   alien : {
2       float life;
3       float speed;
4       bool isDead;
5   }

```

Figura 6.3: Estado de un *alienígena*

6.3. Diseño detallado de la Interfaz de Usuario

La interfaz de usuario consiste en una serie de menús, a través de los cuales se irá navegando con el objetivo de jugar el juego con unos ajustes a gusto del usuario. En el inicio de la aplicación (6.4), se muestra la primera pantalla desde la cual se puede interactuar con tres botones.



Figura 6.4: Menú principal del juego

Si el usuario desea configurar las opciones del juego, puede presionar el botón “OPTIONS”, lo que le llevará al siguiente menú (6.5). Desde aquí, puede configurar los niveles de audio del juego y cambiar la semilla de generación del planeta, además de las dimensiones del mismo.

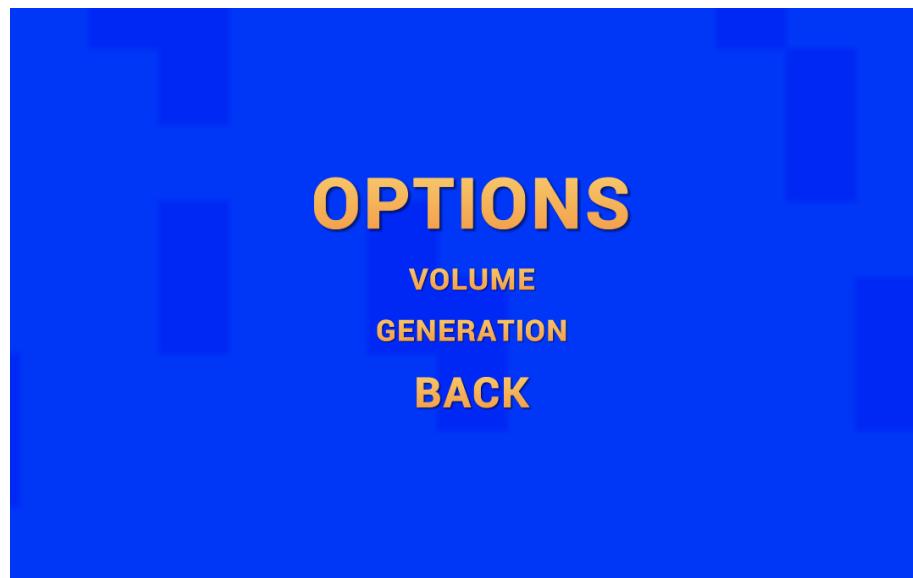


Figura 6.5: Menú de configuración

Si el usuario presiona el botón “VOLUME”, le llevará al menú 6.6, que se muestra a

continuación.

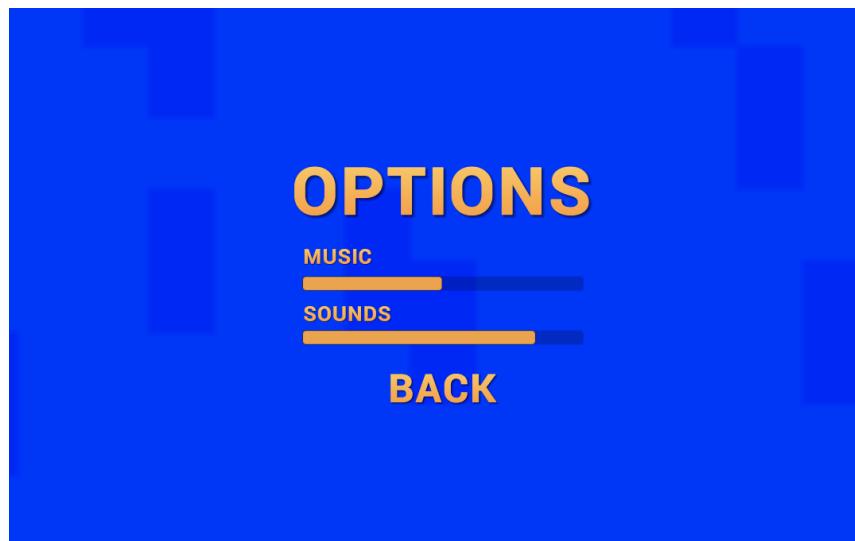


Figura 6.6: Menú de configuración del volumen

Sin embargo, si el usuario presiona en el botón “GENERATION”, le llevará al menú 6.7, mostrado ahora a continuación.

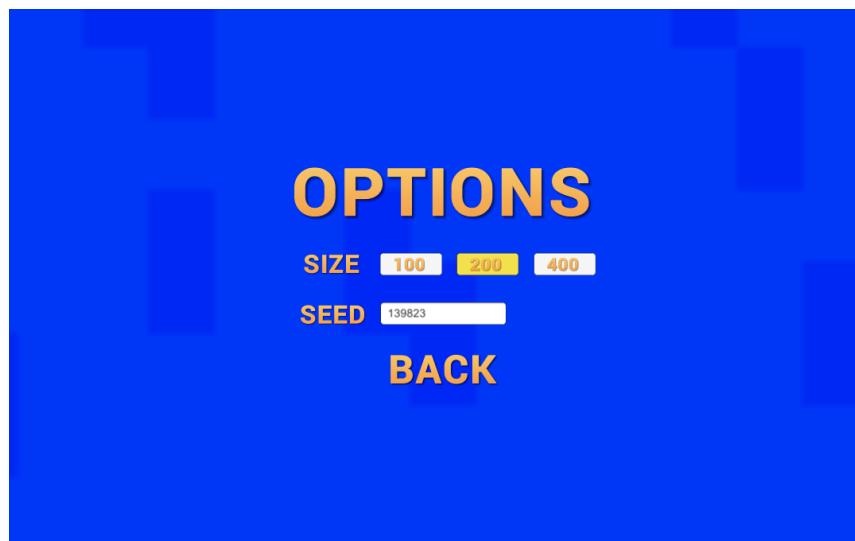


Figura 6.7: Menú de configuración de la generación

Una vez presionado el botón “PLAY” en el menú principal (6.4), el usuario puede pausar el juego con la tecla *ESC*, llevándolo al menú de pausa.

Capítulo 7

Construcción del Sistema

En este capítulo se explicarán con detalle los algoritmos usados y la implementación del proyecto en sí, además de los sprites y gráficos creados. Además de esto, se hará incapié en la arquitectura del sistema.

7.1. Diseño Gráfico

En el videojuego desarrollado, se utilizan una serie de sprites y gráficos para facilitarle al usuario los eventos que están ocurriendo en todo momento. Por ejemplo, cuando los astronautas van a dirigirse al punto más alto encontrado, aparece un sprite de admiración rojo para indicar al usuario de que ese es el evento que tienen en mente los astronautas en ese momento. Esto se hace, en mayor parte, porque muchos de los eventos no los puede controlar el usuario, luego es necesario una explicación por parte del sistema (en este caso, el sprite de símbolo de admiración rojo). El diseño de los sprites se ha realizado usando la aplicación **Paint.NET**.

Los eventos que el sistema mostrará en el juego se describen a continuación, acompañados del sprite que lo acompaña.

1. **Signo de selección:** El signo 7.1 es mostrado por el sistema cuando el usuario sitúa el cursor encima de un astronauta. El fin de este signo es notificar al usuario que puede pinchar para seleccionar el astronauta que lo está mostrando.
2. **Signo de exclamación:** El signo 7.2 es mostrado por el sistema cuando los astronautas deciden dirigirse hacia la montaña más alta que ellos han encontrado. Esto sucede cuando el tiempo de exploración ha acabado o cuando el usuario ha pulsado el botón “Stop Exploring”.
3. **Bandera de astronauta:** La imagen 7.3 es mostrado por el sistema cuando el usuario presiona el botón bandera en un determinado astronauta. El botón determina que ese astronauta en concreto es importante para el usuario. Un astronauta es más importante que otro cuando su velocidad y su vida es superior a la de otro astronauta.

4. **Asignar arma:** La imagen 7.4 es mostrado por el sistema cuando los astronautas han llegado al punto más alto de la montaña. Esto ocurre para notificar al usuario de que le asigne a cada uno de los astronautas un arma (una espada o un escudo) en función de las observaciones del usuario.
5. **Signo de enfado:** El signo 7.5 es mostrado por el sistema para notificar que un alienígena está enfadado. Esto ocurre cuando un astronauta está demasiado cerca de un alienígena, causando que este huya y que los alienígenas cercanos entren en modo *enfado*.
6. **Signo de tristeza:** El signo 7.6 es mostrado por el sistema para notificar que un alienígena está triste. Esto ocurre cuando un astronauta está demasiado cerca de este alienígena, causando que este huya y que deje de disparar, entrando en el estado *triste*.
7. **Asignar espada:** La imagen 7.7 es mostrado por el sistema cuando el usuario le asigna una espada a un determinado astronauta.
8. **Asignar escudo:** La imagen 7.8 es mostrado por el sistema cuando el usuario le asigna un escudo a un determinado astronauta.
9. **Atacar alienígena:** La imagen 7.9 es mostrado por el sistema cuando el usuario selecciona el botón “Atacar” y sitúa el cursor sobre un alienígena. De esta forma, el sistema indica al usuario que si hace click sobre ese alienígena, los astronautas con espada priorizarán su ataque sobre él.
10. **Defender astronauta:** La imagen 7.10 es mostrado por el sistema cuando el usuario selecciona el botón “Defender” y sitúa el cursor sobre un astronauta con espada. De esta forma, el sistema indica al usuario que si hace click sobre ese astronauta, los astronautas con escudo priorizarán su defensa sobre él.

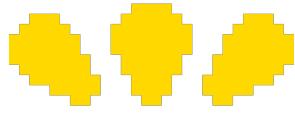


Figura 7.1: Signo de selección



Figura 7.2: Signo de exclamación

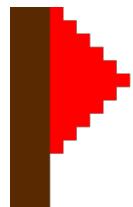


Figura 7.3: Bandera de astronauta

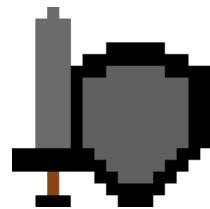


Figura 7.4: Asignar arma

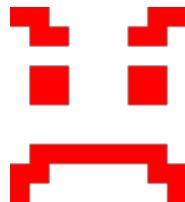


Figura 7.5: Signo de enfado

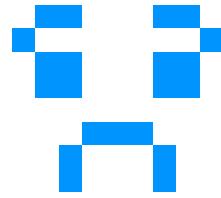


Figura 7.6: Signo de tristeza



Figura 7.7: Asignar espada



Figura 7.8: Asignar escudo



Figura 7.9: Atacar alienígena



Figura 7.10: Defender astronauta

En cuanto al cursor, el sistema intercambia una serie de sprites sobre este en función de la velocidad y dirección del cursor, de forma que al usuario le parezca que está moviendo un cursor tridimensional pero con estilo retro (dados que el diseño es pixelado).

El sprite por defecto del cursor es 7.13, mientras que al moverlo arriba, abajo, izquierda, derecha el sprite por defecto se transforma en los siguientes, respectivamente: 7.11, 7.15, 7.12, 7.14. Al pulsar el botón izquierdo del cursor, el sprite del cursor cambiará a 7.16 para indicar al usuario que al mover el cursor, se desplazará a lo largo del mundo (moverá la cámara). Cuando el cursor se sitúe encima de algún botón, el sprite cambiará a 7.17.

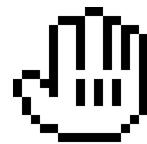


Figura 7.11: Cursor arriba



Figura 7.12: Cursor izq.



Figura 7.13: Cursor centro



Figura 7.14: Cursor der.

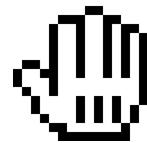


Figura 7.15: Cursor abajo



Figura 7.16: Cursor agarrando



Figura 7.17: Cursor señalando

7.2. Desarrollo del Juego

El juego está dividido en varios bloques, desde el punto de vista algorítmico. Se explicarán los aspectos del juego más relevantes para comprender el funcionamiento del sistema.

7.2.1. Generación del terreno

El planeta a generar es una esfera que surge de la esferificación de un cubo compuesto por 6 matrices cuadradas de $n \times n$ dimensiones. Esta dimensión es por defecto 200, pero el usuario puede seleccionar otro valor. Los valores de n pueden ser: 100, 200 o 400. Por consiguiente, el planeta tendrá una de esas dimensiones.

En una primera instancia, el sistema lee la dimensión de las matrices del script de configuración (por defecto se rellena con el XML de configuración, y algunos de sus valores cambiarán en caso de que lo haya hecho el usuario), además de la semilla de generación. Luego, genera los valores de cada una de las seis matrices que conforman el mundo. Estos valores determinarán la altitud en cada punto del planeta.

Para generar los valores de cada vértice del planeta, se usa el ruido de Perlín 2D. Para cada matriz, se generan sus valores usando este ruido de forma que las matrices adyacentes mantengan la consistencia de sus valores. El problema aquí reside en que, al conformar las seis matrices un cubo, hay aristas que no pueden mantener esta consistencia usando este método.

Para resolver esta problemática, una opción sería usar el ruido de Perlín 3D. Pero este es menos configurable que su versión 2D, así que decidí hacer un postprocesado de las matrices de forma que se cosan las inconsistencias en las uniones de ciertas matrices (aristas). Este algoritmo lo explicaré más adelante. [9]

Primero se calculan los valores de cada matriz, para lo que se llama al algoritmo del ruido de Perlín 2D. Este algoritmo se describe a continuación.

```

1  public static float[,] GenerateNoiseMap( int mapWidth, int mapHeight, int
2      seed, float scale, int octaves, float persistance, float lacunarity,
3      Vector2 offset)
4  {
5      float[,] noiseMap = new float[mapWidth, mapHeight];
6
7      System.Random prng = new System.Random(seed);
8      Vector2[] octaveOffsets = new Vector2[octaves];
9
10     float maxPossibleHeight = 0;
11     float amplitude = 1;
12     float frequency = 1;
13
14     for (int i = 0; i < octaves; i++)
15     {
16         float offsetX = prng.Next(-100000, 100000) + offset.x;
17         float offsetY = prng.Next(-100000, 100000) - offset.y;
18         octaveOffsets[i] = new Vector2(offsetX, offsetY);
19
20         maxPossibleHeight += amplitude;
21         amplitude *= persistance;
22     }
23
24     if (scale <= 0)
25     {
26         scale = 0.0001f;
27     }
28
29     float maxLocalNoiseHeight = float.MinValue;
30     float minLocalNoiseHeight = float.MaxValue;
31
32     float halfWidth = mapWidth / 2f;
33     float halfHeight = mapHeight / 2f;
34
35     for (int y = 0; y < mapHeight; y++)
36     {
37         for (int x = 0; x < mapWidth; x++)
38         {
39             amplitude = 1;
40             frequency = 1;
41             float noiseHeight = 0;
42
43             for (int i = 0; i < octaves; i++)
44             {
45                 float sampleX = (x - halfWidth + octaveOffsets[i].x) /
46                     scale * frequency;
47                 float sampleY = (y - halfHeight + octaveOffsets[i].y) /
48                     scale * frequency;
49
50                 float perlinValue = Mathf.PerlinNoise(sampleX, sampleY)
51                     * 2 - 1;
52                 noiseHeight += perlinValue * amplitude;
53
54                 amplitude *= persistance;
55                 frequency *= lacunarity;
56             }
57         }
58     }
59
60     return noiseMap;
61 }
```

Figura 7.18: Ruido de Perlín 2D

```

1         if ( noiseHeight > maxLocalNoiseHeight )
2         {
3             maxLocalNoiseHeight = noiseHeight ;
4         }
5         else if ( noiseHeight < minLocalNoiseHeight )
6         {
7             minLocalNoiseHeight = noiseHeight ;
8         }
9         noiseMap [x, y] = noiseHeight ;
10    }
11}
12
13 for (int y = 0; y < mapHeight; y++)
14{
15    for (int x = 0; x < mapWidth; x++)
16    {
17        float normalizedHeight = (noiseMap [x, y] + 1) / (
18            maxPossibleHeight );
19        noiseMap [x, y] = Mathf.Clamp( normalizedHeight , 0, int .
20            MaxValue );
21    }
22}
23 return noiseMap ;
}

```

Figura 7.18: Ruido de Perlín 2D

En este caso, *mapWidth* sería el ancho de la matriz, *mapHeight* sería el alto de la matriz, *seed* es la semilla, *scale* es un parámetro que define la frecuencia de la generación del ruido en sí. Cuanto mayor sea esta frecuencia, más juntas estarán las montañas. *Octaves* es el parámetro que define la resolución del ruido generado; cuanto menor sea el valor, más simple y monótono será el ruido generado. *Persistance* es el valor utilizado para reducir la contribución de pases adicionales, de modo que obtendrá un patrón de ruido mayor con características y desviaciones adicionales agregadas. Cuanto más pequeño se vuelve, menos importantes son las octavas posteriores y se crea una imagen más simple. *Lacunarity* es el parámetro que, mientras menor sea, más suaviza el ruido generado. La función *GenerateNoiseMap* devuelve la matriz de ruido generado.

Las seis matrices generadas usando este script serán, más adelante, modificadas ligeramente debido a la problemática explicada anteriormente. El ruido se extiende correctamente a lo largo de las matrices, como se puede ver correctamente en la figura 7.19. El problema reside en las aristas azules (véase en la misma figura), que no coinciden a la hora de cerrar el cubo.

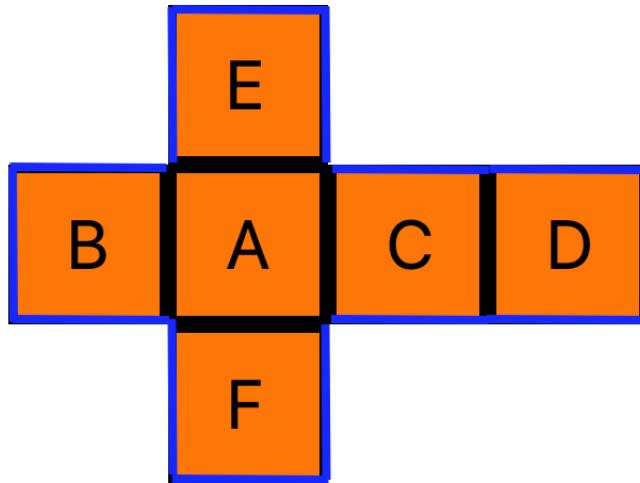


Figura 7.19: Matrices de ruido

Para eso, las matrices B , C , D , E , F recalcularán sus aristas externas problemáticas usando un algoritmo iterativo que consiste en hallar la media entre los dos puntos de distintas matrices y extender estos cálculos hasta una profundidad en la que no sea notable el “cosido” de matrices.

Una vez generadas las matrices, lo harán sus texturas y los *chunks*. Las texturas contendrán valores que oscilan en una escala de colores anaranjada, en función del valor del ruido en ese punto (entre 0 y 1). Cada matriz se dividirá en una serie de objetos, llamados *chunks*, con el objetivo de granularizar su renderizado y resolución. El número de *chunks* por matriz es de 625, independientemente de las dimensiones de esta.

7.2.2. Actualización de los *chunks*

En el momento de la generación del terreno del planeta (después de su generación descrita en el apartado anterior), el objetivo del sistema es renderizar únicamente los *chunks* cercanos a la cámara. No solo esto, si no hacerlo a un determinado *LOD* (en función de la cercanía de la cámara al planeta, efectuada por la ampliación que el usuario haga).

Para hacer esto, los *chunks* son consultados en función de la distancia de su punto intermedio a la posición de la cámara. El algoritmo que el sistema sigue para actualizar los *chunks* es el siguiente.

```
1  private IEnumarator UpdateChunks()
2  {
3      float zoomPercentage = camera.GetComponent<CameraController>().zoomPercentage;
4
5      int maxChunksToRender;
6      int minChunksToRender;
7      switch(gridSize)
8      {
9          case 100: maxChunksToRender = 1500; minChunksToRender = 200;
10         break;
11         case 200: maxChunksToRender = 1300; minChunksToRender = 110;
12         break;
13         case 400: maxChunksToRender = 1000; minChunksToRender = 50;
14         break;
15     }
16     int chunksToRender = (int)(minChunksToRender + (zoomPercentage / 100) * (maxChunksToRender - minChunksToRender));
17
18     int maxResToRender;
19     if(zoomPercentage <= 20f)
20     {
21         maxResToRender = 1;
22     }
23     else
24     {
25         if(zoomPercentage <= 40f)
26         {
27             maxResToRender = 2;
28         }
29         else
30         {
31             if(zoomPercentage <= 60f)
32             {
33                 maxResToRender = 4;
34             }
35             else
36             {
37                 maxResToRender = 8;
38                 if(gridSize == 100)
39                 {
40                     maxResToRender = 4;
41                 }
42             }
43         }
44     }
45
46     Chunk closestChunk = chunks[0];
```

Figura 7.20: Actualización de *chunks*

```

1      foreach (Chunk chunk in chunks)
2      {
3          if (chunk.IsClosestChunk())
4          {
5              closestChunk = chunk;
6              closestChunk.UpdateLOD(maxResToRender);
7          }
8      }
9
10     int adjacentCount = 1;
11     foreach (Chunk adjacent in closestChunk.GetAdjacentChunks())
12     {
13         if (adjacentCount >= chunksToRender)
14         {
15             adjacent.Render(false);
16         }
17         else
18         {
19             adjacent.UpdateLOD(maxResToRender);
20             ++adjacentCount;
21         }
22     }
23
24     yield return null;
25 }
```

Figura 7.20: Actualización de *chunks*

En primera instancia, se obtiene el zoom que está realizando el jugador (**zoomPercentage**) y, en función de eso y del tamaño del planeta (**gridSize**), se determina el número de *chunks* a renderizar (**chunksToRender**). Cuanto más zoom se haga, menos *chunks* se renderizarán.

Luego, se determinará el fraccionado del *LOD* de cada *Chunk* en función del zoom. **maxResToRender** contendrá el valor por el que se dividirán las aristas de un *chunk*. Por ejemplo, si **maxResToRender** vale la unidad, el *LOD* del *chunk* no se verá modificado, pero si vale 2 se dividirá entre 2, resultando en un *chunk* de $n/2 \times n/2$ dimensiones (siendo n la dimensión del *chunk*). Un *chunk* se podrá fraccionar hasta un máximo de 8 veces (siempre que la dimensión del terreno sea 200 o 400).

En el bucle de la línea 36, se actualiza el *LOD* de cada *chunk* adyacente al más cercano a la cámara hasta llegar al máximo número de *chunks* a renderizar. Si el *chunk* está demasiado lejos, no se renderizará (**adjacent.Render(false)**).

La función descrita en la figura 7.20 se ejecutará cada vez que la cámara se haya movido 0,5 unidades y siempre que el *chunk* más cercano a esta haya cambiado. Esto se hace porque, de ser llamada constantemente, se perdería demasiado rendimiento inútilmente.

7.2.3. Inteligencia Artificial

El comportamiento de los astronautas está definido por una serie de algoritmos. Cada uno funciona para una fase distinta de estos. En una primera instancia, cuando el jugador presiona el botón de “PLAY”, ocho astronautas se generan en una posición concreta del mundo, flotando, a la espera de que el jugador seleccione el porcentaje de exploración y presione el botón “Ok” para que comience la exploración y la puesta en marcha del primero de los algoritmos, el *PSO*. [10] [11]

7.2.3.1. PSO de los astronautas

El algoritmo que sigue cada uno de los ocho astronautas en el momento de la búsqueda de la montaña más alta del planeta es el de la figura 7.21. Este algoritmo devolverá **true** si los astronautas han terminado de explorar, y devolverá **false** si aún no han terminado.

Cada iteración en la exploración se irá contando con el contador **iteration_astronaut**. Si se llega al máximo de iteraciones o bien el usuario presiona el botón de “Stop Exploring”, los astronautas se dirigirán a la montaña más alta encontrada hasta el momento por todos ellos (**GoToGlobalMaxAstronaut()**). Si no, cada astronauta se moverá en la dirección en la que se esté dirigiendo en ese momento y anotará la puntuación de la altura en la que esté (**controller.Move()**, siendo **controller** el objeto que maneja al astronauta, y **controller.UpdatePersonalScore()**).

La función **UpdateGlobalScoreAstronaut()** en la línea 20 se encarga de actualizar y comunicar a todos los astronautas la mejor posición hasta el momento. La siguiente función, **UpdateTrajectoryAstronaut(...)** recibe tres argumentos. El primero de ellos es la inercia del astronauta en ese momento, **Wcurrent_astronaut**, valor que se irá reduciendo con el tiempo de exploración y que determina la importancia de la dirección de su movimiento frente a la dirección del movimiento que determina las posiciones de las mejores posiciones personales y globales. Los últimos dos argumentos son constantes que determinan la importancia de las mejores posiciones personales y globales; **c1_astronaut** vale 1 y se aplica sobre la mejor posición personal y **c2_astronaut** vale 1,5 y se aplica sobre la mejor posición global, luego la componente resultante apuntará más hacia el máximo global que el local. Por último, se ejecuta la función **UpdateWeightsAstronaut()** para actualizar el peso de la inercia de cada uno de los astronautas, que esencialmente la reduce iteración por cada iteración.

```

1   public bool UpdateAstronauts()
2   {
3       if(iteration_astronaut == (maxIterations_astronaut +
4           maxIterWithWmin_astronaut) || stopExploring_astronaut)
5       {
6           GoToGlobalMaxAstronaut();
7           CheckGlobalMaxAstronaut();
8           bool weaponsAssigned = AllWeaponsAssigned();
9           if(weaponsAssigned)
10          {
11              return true;
12          }
13      else
14      {
15          foreach (PlayerController controller in astronautControllers)
16          {
17              controller.Move();
18              controller.UpdatePersonalScore();
19          }
20          UpdateGlobalScoreAstronaut();
21          UpdateTrajectoryAstronaut(Wcurrent_astronaut, c1_astronaut,
22             c2_astronaut);
23          UpdateWeightsAstronaut();
24          ++iteration_astronaut;
25      }
26      return false;
}

```

Figura 7.21: Algoritmo PSO aplicado a los astronautas

7.2.3.2. Algoritmo de batalla de los astronautas

Una vez los astronautas han alcanzado la montaña más alta y cada uno tiene su arma asignada, aparecerán ocho alienígenas en un punto cercano del planeta. En este instante, los astronautas ejecutan un algoritmo de organización y ataque que se explicará a continuación, con fragmentos de código.

Los astronautas se dividen en dos grupos importantes. Los primeros son los que el jugador ha dotado de escudos, llamados escuderos. Los que poseen espada son los espadachines. Cada grupo sigue un algoritmo diferente en el momento del combate.

Para el caso de los escuderos, el algoritmo se puede observar en la figura 7.22. La función **Defend(...)** recibe como argumentos dos **PlayerController**, que son los controladores que manejan el modelo de los astronautas. El primero es el escudero y el segundo es el astronauta que éste defenderá. Para lograr defenderlo, el defensor debe interponerse entre el astronauta a defender y el alienígena que le está disparando.

Para ello, itera sobre las bolas en la línea 7 en búsqueda de la más cercana al astronauta a defender (cada alienígena tiene asignado una bola, por eso itera sobre controladores de alienígenas). En las siguientes líneas, se calcula la dirección que debe tomar el defensor para dirigirse a una posición óptima e interponerse entre el alienígena y el astronauta para parar la bola (**direction**).

En el bucle de la línea 8 de la segunda figura, se itera sobre los demás astronautas defensores para comprobar que la dirección anterior deseada no acabaría en una colisión con otro defensor (o acabarían todos los defensores en el mismo punto). De ser así, la dirección tomará el sentido opuesto mientras se ubique demasiado cerca de otros defensores. Lo que se consigue con esto, es que si un espadachín tiene varios defensores, no vayan a defender a un mismo punto sino que haya un margen entre ellos para que le dé más cobertura a la hora de parar el proyectil procedente del alienígena.

Finalmente, en las líneas 26-28 de la segunda figura se actualiza la trayectoria del astronauta defensor.

```

1  private void Defend( PlayerController controller , PlayerController
2  	defender )
3  {
4  	float minimumDistanceToBall = 1000f ;
5  	Vector3 positionOfAlien = Vector3.zero ;
6  	AlienController closestBall = null ;
7
8  	foreach ( AlienController alien in alienControllers )
9  	{
10   	if( defensorBall[ alien ] == -1 && alien . ball . activeInHierarchy )
11   	{
12    	float distanceToBall = Vector3.Distance( defender . transform .
13    	position , alien . ball . transform . position );
14    	if ( distanceToBall < minimumDistanceToBall )
15    	{
16    		closestBall = alien ;
17    		minimumDistanceToBall = distanceToBall ;
18    		positionOfAlien = alien . transform . position ;
19    	}
20   	}
21
22   	if( closestBall != null)
23   	{
24    	defensorBall[ closestBall ] = controller . id ;
25    }

```

Figura 7.22: Algoritmo de defensa de un escudero a un espadachín

```

1     Vector3 defenderToAlien = positionOfAlien - defender.transform.
2         position;
3     float multiplier = distanceToDefender / defenderToAlien.magnitude;
4
5     Vector3 direction = defender.transform.position + defenderToAlien *
6         multiplier;
7
8     float minimumDistanceToDefender = 1000f;
9     Vector3 defenderAway = Vector3.zero;
10    foreach (PlayerController player in astronautControllers)
11    {
12        if (player.GetWeapon() == "shield" && player.id != controller.id)
13        {
14            float distance = Vector3.Distance(player.transform.position,
15                controller.transform.position);
16            if (distance < minimumDistanceToDefender)
17            {
18                minimumDistanceToDefender = distance;
19                defenderAway = (controller.transform.position - player.
20                    transform.position).normalized * 2f;
21            }
22        }
23
24        if (minimumDistanceToDefender < separationBetweenDefensors)
25        {
26            direction += defenderAway;
27        }
28
29    }

```

Figura 7.22: Algoritmo de defensa de un escudero a un espadachín

Para los espadachines es distinto. Su algoritmo (figura 7.23) consiste en dirigirse hacia el alienígena que desean atacar y, cuando estén lo suficientemente cerca, efectuar el ataque.

La función **Attack(...)** recibe dos argumentos. El primero es el controlador del modelo del astronauta espadachín, mientras que el segundo es el controlador del modelo del alienígena que será atacado. En la línea 5, se comprueba si la distancia al alienígena es la suficiente como para atacar. En caso contrario, el astronauta actualizará su trayectoria hacia el alienígena (líneas 17-19).

```

1  private void Attack(PlayerController controller, AlienController alien)
2  {
3      Vector3 attackerToAlien = alien.transform.position - controller.
4          transform.position;
5      float distanceToAlien = Vector3.Distance(controller.transform.
6          position, alien.transform.position);
7      if(distanceToAlien <= minDistanceToAttackAlien)
8      {
9          timeSinceAttack[controller.id] += Time.deltaTime;
10         if(timeSinceAttack[controller.id] > minTimeToAttack)
11         {
12             timeSinceAttack[controller.id] = 0f;
13             alien.Hit();
14         }
15     }
16     else
17     {
18         Vector3 direction = controller.transform.position +
19             attackerToAlien;
20         controller.UpdateTrajectoryDirection(direction);
21         controller.SetMove(true);
22         controller.Move();
23     }
24 }
```

Figura 7.23: Algoritmo de ataque de un espadachín a un alienígena

7.2.3.3. PSO de los alienígenas

Una vez se generan los alienígenas en un punto del planeta, estos siguen un algoritmo de *PSO* similar al de que siguen los alienígenas en la exploración, pero ligeramente modificado. El valor a optimizar, en este caso, es la cercanía a los alienígenas.

Este algoritmo de exploración se puede observar en la figura 7.24. Devolverá **true** si los alienígenas han encontrado a los astronautas, y **false** en caso contrario. El cuerpo del algoritmo se encuentra en la línea 12, en la que se actualizarán las puntuaciones y, posteriormente, las trayectorias de los alienígenas, al igual que se hacen con los astronautas.

```

1  public bool UpdateAliens()
2  {
3      if (positionCloseToAstronauts != Vector3.zero)
4      {
5          if (AllCloseEnoughToPosition(positionCloseToAstronauts))
6          {
7              return true;
8          }
9      }
10     else
11     {
12         foreach (AlienController controller in alienControllers)
13         {
14             controller.Move();
15             controller.UpdatePersonalScore();
16         }
17         UpdateGlobalScoreAlien();
18         UpdateTrajectoryAlien(Wcurrent_alien, c1_alien, c2_alien);
19         UpdateWeightsAlien();
20         ++iteration_alien;
21     }
22     return false;
23 }
```

Figura 7.24: Algoritmo PSO aplicado a los alienígenas

7.2.3.4. Algoritmo de batalla de los alienígenas

Una vez los alienígenas han encontrado a los astronautas, ejecutan su algoritmo de batalla. Este consiste en lo siguiente: Cada alienígena puede encontrarse en tres estados (0, 1 y 2) diferentes, solo uno al mismo tiempo. El estado 0 consiste en atacar a los astronautas. En este estado, la probabilidad de atacar al astronauta más cercano es del 70 %, mientras que la de golpear a cualquier otro astronauta (aleatorio) es del 30 %. En caso de que el astronauta objetivo esté demasiado lejos, irá tras él. El estado 1 consiste en huir de un astronauta. Este estado se da cuando un astronauta se encuentra demasiado cerca al alienígena, causándole que huya. Cuando un alienígena se encuentra en este estado, avisa a los alienígenas cercanos para que se cambien al estado 2 y ataquen al astronauta que le ha puesto en ese estado. El estado 2 es como el 0, solo que el astronauta objetivo está definido, no puede ser aleatorio.

A parte de esto, los alienígenas se moverán por composición de una serie de vectores de dirección y, cuando estén demasiado cerca entre sí, estos vectores se invertirán, consiguiendo que los alienígenas vayan a donde se proponen pero evitando estar demasiado cerca de otros alienígenas.

Capítulo 8

Pruebas

Para la realización de pruebas durante el desarrollo del software de este proyecto, se han realizado múltiples ejecuciones en modo *Debug* del juego, de forma que se generan una serie de ficheros (logs) en los que se guardan estadísticas que serán analizadas posteriormente.

En cuanto a la experiencia jugando, las pruebas de ejecución que se han realizado son las siguientes (tomando como dispositivo que compone la arquitectura física del sistema mi ordenador personal):

- El computador posee un procesador **Intel Core i5-6300HQ**, y una gráfica **Nvidia GeForce GTX 960M**.
- La media de fotogramas por segundo es de 49.34 fps. Esta media se mantiene para los 3 tamaños disponibles de mapa, debido a que el número de triángulos renderizados en todo momento es, aproximadamente, el mismo (gracias a la carga de Chunks usando *LOD* dinámico).

8.1. Pruebas Unitarias

Las pruebas unitarias se han desarrollado con el objetivo de comprobar que el comportamiento de cada componente del software da un resultado esperado frente a una serie de entradas.

Las pruebas se han realizado en distintas partes de la aplicación, mayormente en la generación del terreno debido a que es crucial que no haya ningún *chunk* malformado o podría causar que alguna entidad se quede atascada en el mismo terreno. Estas pruebas consisten en la programación de una sucesión de patrones de movimiento para los astronautas y alienígenas sobre el terreno generado, y la comprobación de que en cada instante se cumplen casos como los siguientes:

1. La posición de la entidad se encuentra fuera del terreno generado.
2. Los alienígenas mantienen la distancia con sus objetivos.

3. El objetivo de un alienígena no debe ser un astronauta que esté muerto.

El procedimiento que se ha sacado para determinar la corrección de las salidas del juego frente a una serie de entradas de prueba es la salida de logs en tiempo de ejecución de la aplicación, haciendo uso de *Debug.Log(datos a comprobar)* en Unity 3D.

8.2. Pruebas de Integración

Las pruebas de integración consisten en aquellas que comprueban que todos y cada uno de los componentes del elemento conjunto a probar funcionan correctamente. En el caso de este proyecto, las pruebas de integración se han realizado de forma progresiva, de forma que las pruebas unitarias de cada componente que esté compuesto de otro componente con sus correspondientes pruebas, tendrán otras pruebas similares que conformarán las pruebas de integración de ambos componentes.

Para comprobar la integración de todos y cada uno de los componentes, se ha utilizado Unity 3D de forma que en la escena principal del juego se ejecutan todos los módulos de los que este se compone al mismo tiempo. Una vez se ejecutan todos estos módulos, se comprueba que la aplicación funciona correctamente con ayuda de Unity 3D y de su impresión de logs.

8.3. Pruebas de Sistema

8.3.1. Pruebas Funcionales

Las pruebas funcionales consisten en la comprobación de que el sistema se comporta como debe en los escenarios descritos por su modelo de casos de uso, anteriormente descrito.

Para ello, se ha usado el computador antes descrito para ejecutar la aplicación en su versión final y comprobar que los casos de uso actúan tal y como se especifican en esta memoria.

8.3.2. Pruebas No Funcionales

Para las pruebas no funcionales se han comprobado todos y cada uno de los requisitos no funcionales, obteniendo las siguientes conclusiones.

1. **RNF-01 Fiabilidad:** El sistema se comporta de acuerdo a sus especificaciones y produce el mínimo número de errores.
2. **RNF-02 Eficiencia:** El sistema rinde de la mejor forma posible a la hora de recibir peticiones por parte del usuario y, por consiguiente, es capaz de cargar el terreno de forma fluida.

3. **RNF-03 Accesibilidad:** El sistema es capaz de ejecutarse en una amplia gama de computadores, indistintamente del hardware que estos posean, debido al requisito **RNF-02**, siempre que sea en cualquier sistema Windows.
4. **RNF-04 Seguridad:** El sistema es seguro y garantiza que los datos del usuario no son modificados por parte de terceros.

Capítulo 9

Conclusiones y Trabajo Futuro

9.1. Problemas encontrados

Durante el desarrollo del software de este proyecto me he encontrado con diversos problemas que me han alargado el este proceso de desarrollo.

Uno de los mayores problemas ha sido la curva de aprendizaje con el software Unity 3D. Dado que la mayor parte del proyecto se ha desarrollado usando esta plataforma y nunca antes la había usado, vi difícil aplicar las ventajas de este software al principio del desarrollo.

Como ejemplo de estos problemas, comentaré un par de ellos. Uno de estos problemas fue familiarizarme con la *API* de Unity 3D para efectuar rotaciones complejas. Otro de los problemas fue efectuar el renderizado de objetos *Mesh* de Unity 3D de forma eficiente cuando se trataba de generar distintos niveles de *LOD*.

9.2. Conclusiones

9.2.1. Rendimiento y Carga de Terreno

En **Space Hunt**, una de las mayores problemáticas es la carga de terreno. El mundo a generar es un cubo esferificado formado a partir de 6 matrices con dimensiones que oscilan entre 100×100 y 400×400 unidades. Esto significa que, en el peor caso, el juego estará renderizando $(400+1) \times (400+1) \times 6 = 964806$ vértices. Esto son $400 \times 400 \times 2 \times 6 = 1920000$ triángulos.

La problemática de la carga elevada de triángulos lleva a su separación en objetos independientes, llamados *chunks*. Cada uno de estos *chunks* es interpretado como un *GameObject* distinto por Unity. Esto significa que un *chunk* contiene su propio mesh, texturas y *colliders*.

Así, el conjunto de matrices que forman el mundo se compone de *chunks* de forma que cada matriz se divide en $25 \times 25 = 625$ *chunks*. En total, el mundo está dividido en $625 \times 6 = 3750$ *chunks*. Si sólo se renderizan los *chunks* próximos a la cámara (los visibles por parte del jugador), se consigue reducir este elevado número de triángulos que Unity es responsable de cargar. En una situación en la que la cámara está haciendo un zoom al 50 % sobre un mundo de 400×400 unidades, se estarán cargando 525 *chunks*. Un *chunk* está encargado de renderizar $1920000/3750 = 512$ triángulos, luego en este caso se renderizarían $512 \times 525 = 268800$ triángulos, que aún son bastantes.

Aquí es donde entra el juego el nivel de detalle dinámico. Cuanto menos zoom se haga sobre el mundo, menos triángulos se renderizarán en esos *chunks* y tendrá, por consiguiente, menor nivel de detalle. Al mismo tiempo, se estarán renderizando más *chunks* (todos con el mismo nivel de detalle). Volviendo al mismo caso (50 % de zoom sobre un mundo de 400×400 unidades), se reajustará el número de vértices de forma que se formen 4 veces menos triángulos. Esto da como resultado $268800/4 = 67200$ triángulos, un $1920000/67200 = 28,5714 \equiv 2857,14\%$ menos que sin el uso de estos *chunks*.

9.2.2. Estadísticas

A continuación se pueden observar las estadísticas generadas por los astronautas en el momento de la exploración usando el algoritmo PSO (primera instancia del juego).

Cada astronauta (de los ocho en total) genera datos a su paso por las montañas del planeta, de forma que es posible ver los máximos globales y personales que se usan en el algoritmo PSO para determinar las trayectorias fácilmente. Cada astronauta escribe dos datos por iteración, que son la puntuación actual y su mejor puntuación hasta el momento (máximo personal). Aparte, se generan datos correspondientes al máximo global y de la inercia en cada iteración.

Se ha realizado una ejecución del juego para generar estadísticas, con las dimensiones de generación del planeta de 200, para explicar las estadísticas generadas.

En la figura 9.1 se puede ver fácilmente la trayectoria que ha ido siguiendo el astronauta número 0, junto con su máximo personal. Conforme van pasando las iteraciones, la puntuación personal va convergiendo debido a las variables que están en juego en el algoritmo PSO. Para poner otro ejemplo de estadísticas de astronautas, también se muestra la del astronauta número 4 (figura 9.2).

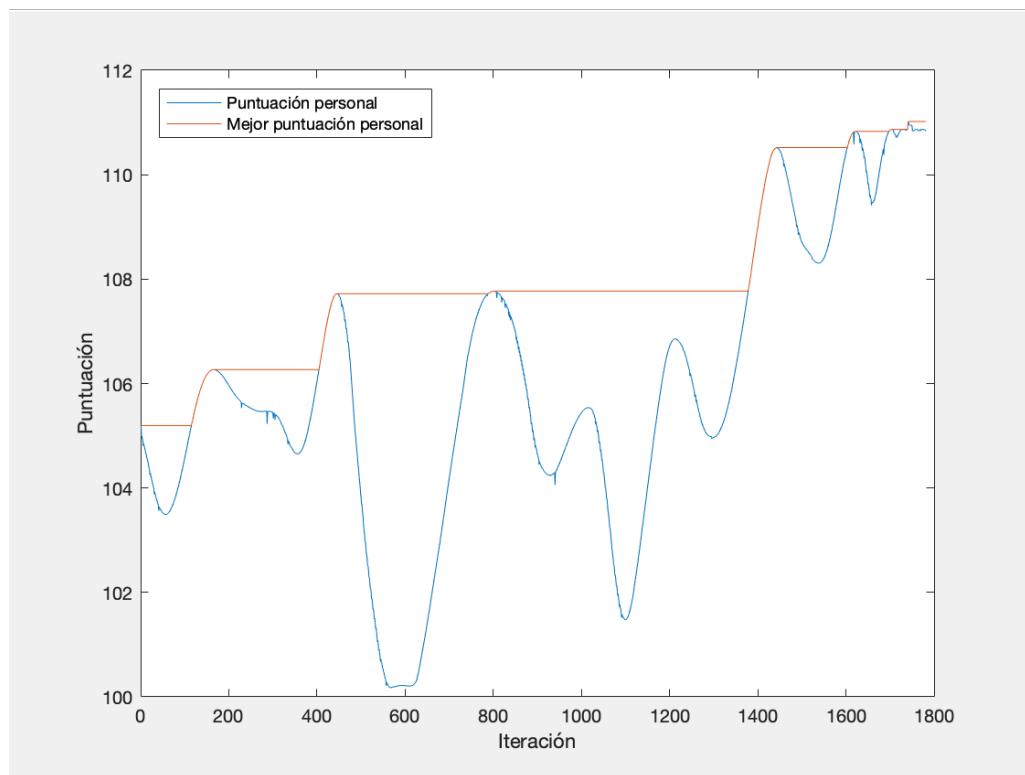


Figura 9.1: Estadísticas de la trayectoria del astronauta número 0

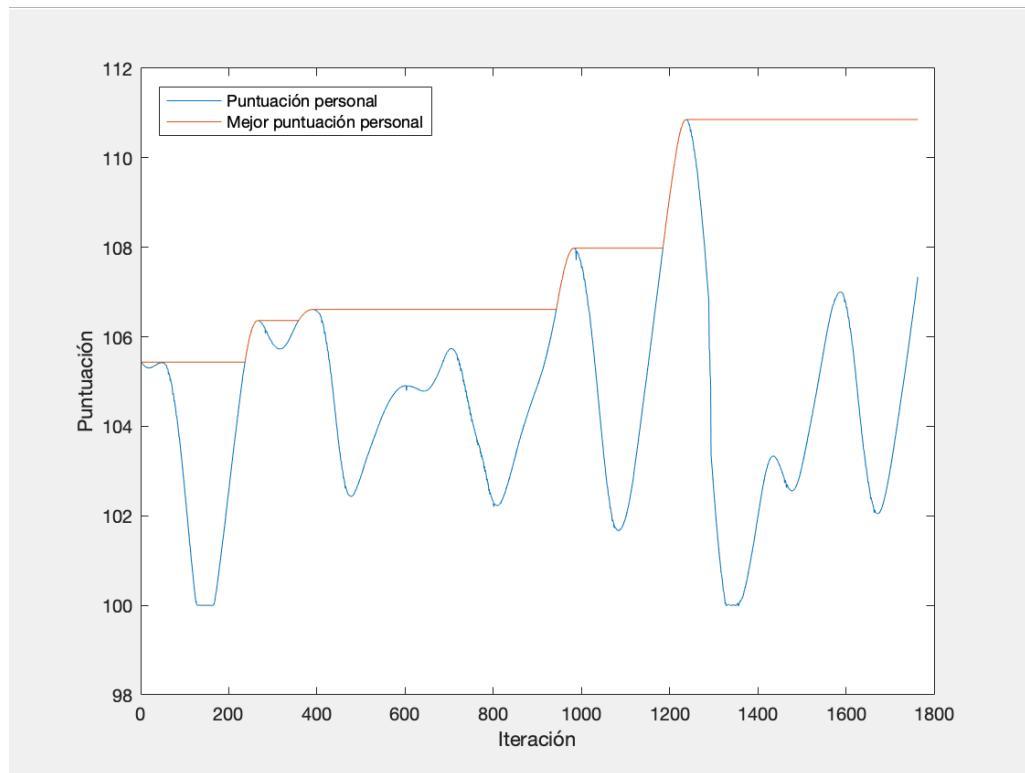


Figura 9.2: Estadísticas de la trayectoria del astronauta número 4

Los datos correspondientes al máximo global se pueden observar en la figura 9.3, mientras que los de la inercia se pueden ver en la figura 9.4. Para entender los datos de la inercia, debo destacar que es un valor que va decreciendo conforme pasan las iteraciones hasta que se estabiliza en un mínimo (valor cercano a 0) durante un número constante de iteraciones al final. El objetivo de hacer esto es darle más importancia a las trayectorias iniciales que a los máximos globales y locales, de forma que se eviten los posibles mínimos locales en la zona de aparición de los ocho astronautas. El valor inicial de la inercia dependerá del que haya seleccionado el usuario.

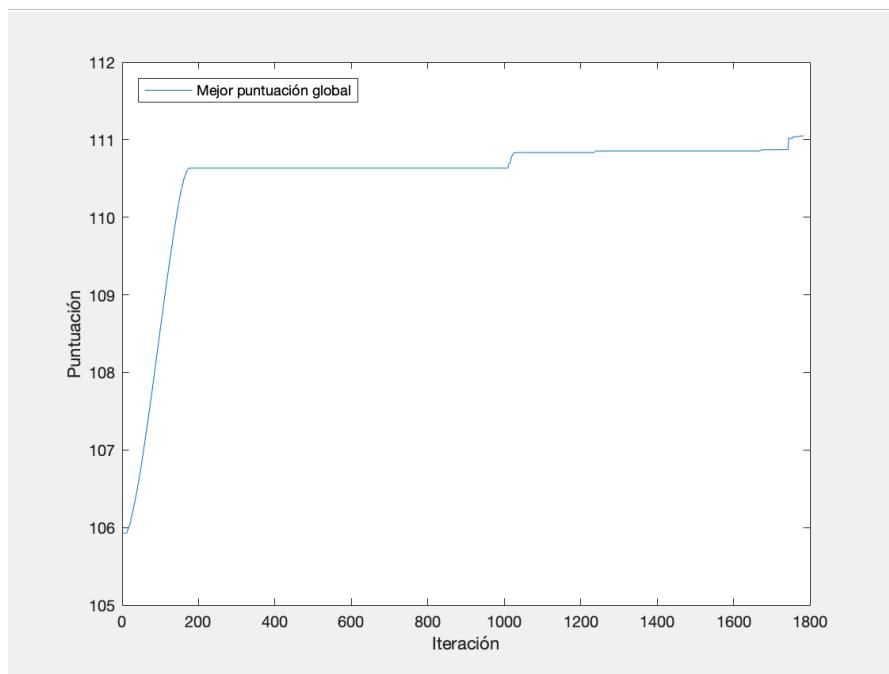


Figura 9.3: Estadísticas de la mejor puntuación global de la trayectoria de los astronautas

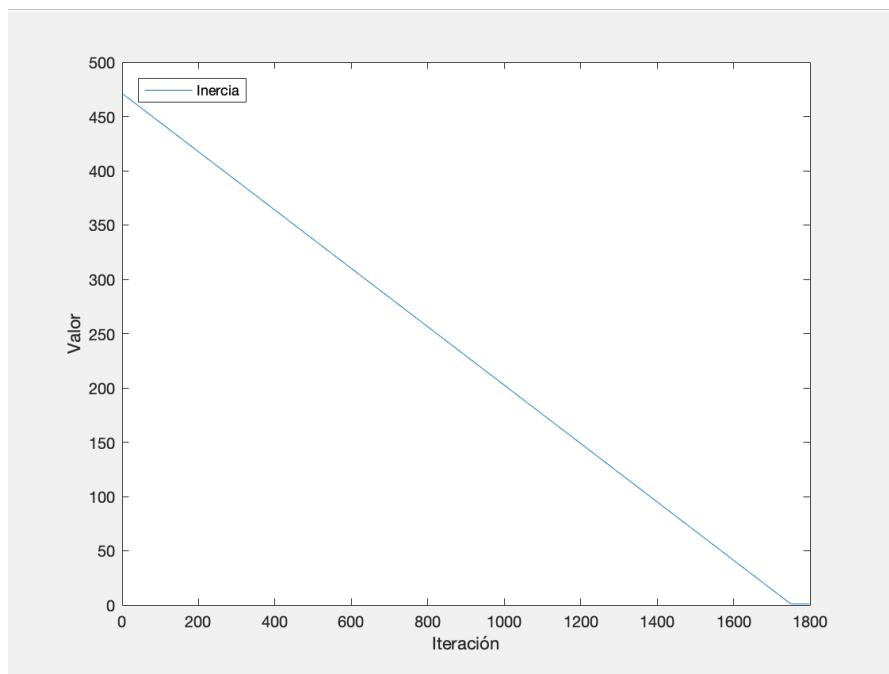


Figura 9.4: Valor de la inercia a lo largo de la trayectoria de los astronautas

9.3. Trabajo Futuro

En un futuro, se podría mejorar el software de **Space Hunt** con acciones como las siguientes:

1. Subir el juego a una plataforma como Steam, de forma que más gente podría disfrutar de su contenido.
2. Corrección intensiva de sus bugs, dado que siempre quedan pequeños errores por pulir.
3. Crear una funcionalidad multijugador, de forma que más usuarios puedan jugar entre sí en línea.
4. Crear un sistema de contribuciones integrado en el juego, de forma que la gente pueda publicar sus opiniones y posibles mejoras en algunos aspectos del juego.
5. Crear un tutorial. Esta mejora tendría más sentido si se lleva a cabo la funcionalidad multijugador dado que el juego se haría más complejo y difícil de entender.

Apéndice A

Anexos

A.1. Manual de Instalación

A.1.1. Introducción

A continuación se detalla el procedimiento a seguir para instalar el software desarrollado en el proyecto, **SpaceHunt**.

A.1.2. Requisitos Previos

Para instalar el software, se deben cumplir una serie de condiciones.

- El dispositivo debe ser un computador. No puede ser un dispositivo móvil.
- El sistema operativo del computador debe ser *Windows*. En concreto, una versión igual o posterior a *Windows 7*.
- Es recomendable que el computador posea una tarjeta gráfica dedicada.

A.1.3. Procedimiento de instalación

Se dispone de un directorio (A.1) en el que se ubican: el ejecutable del juego, *SpaceHunt.exe*, un directorio en el que se leen y guardan datos relacionados con el juego, *SpaceHunt_Data*, y *UnityPlayer.dll*, necesario para la ejecución de la aplicación.

Para jugar, hay que ejecutar *SpaceHunt.exe*.

A.2. Manual de Usuario

A.2.1. Introducción

En este manual se detallan las instrucciones que debe seguir el usuario para un uso correcto de la aplicación.



Figura A.1: Directorio base de la aplicación

A.2.2. Jugar

Una vez configuradas las opciones al gusto en el menú principal de la aplicación (detallado en la sección 6.3), y pulsado el botón “PLAY”, el juego se iniciará.

A.2.2.1. Escenario principal

En primera instancia, veremos al planeta **Marte** generado en mitad del espacio con las dimensiones y semilla especificadas en el menú de generación (A.2). Un grupo de ocho astronautas se ha generado en círculo y están a la espera de que el usuario pulse el botón “OK” ubicado en medio de la pantalla.

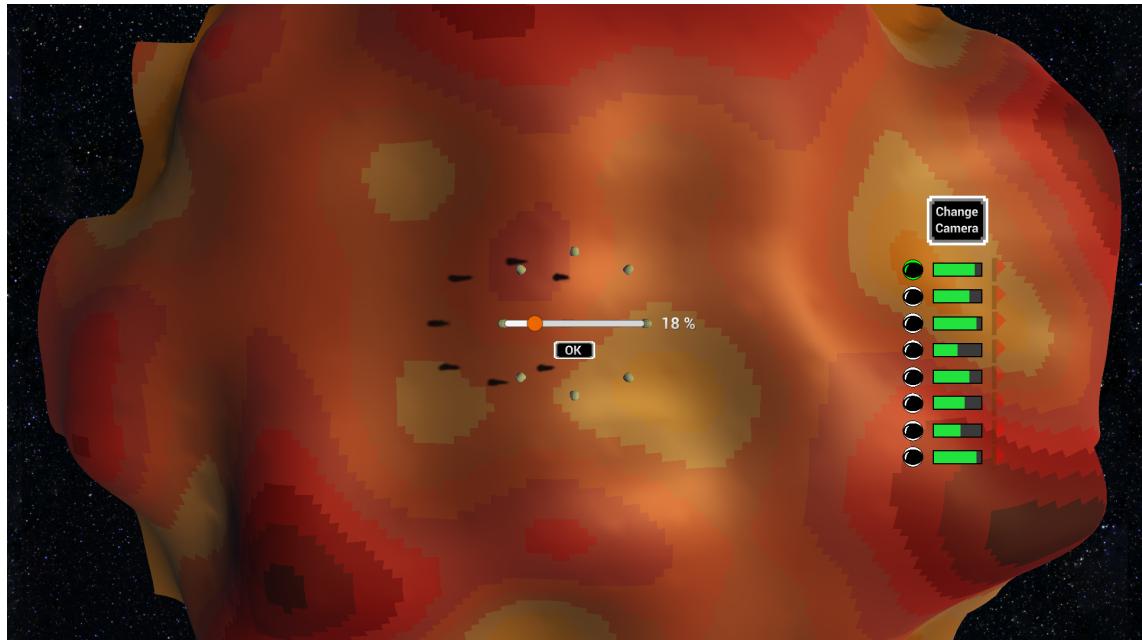


Figura A.2: Escenario principal

Antes de pulsar el botón, el usuario puede mover la cámara por el planeta usando el ratón para observar la topología de este y decidir cuánto merece la pena que valga la inercia de los astronautas. Una vez sabido este porcentaje de exploración, podrá establecerlo en el control deslizante encima del botón “OK”.

A.2.2.2. Fase de exploración

Una vez pulsado el botón “OK”, los astronautas caerán al planeta y comenzarán a explorarlo. La barra horizontal en la parte superior central de la pantalla indica la completitud de la fase de exploración de los astronautas (A.3).

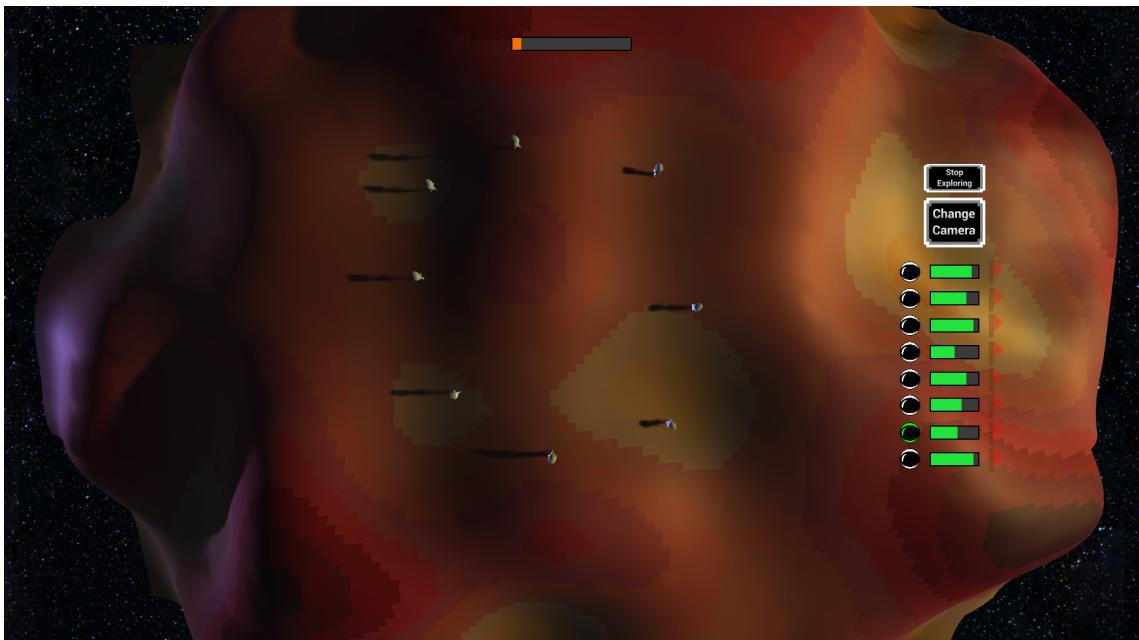


Figura A.3: Fase de exploración

El usuario puede esperar a que completen su exploración, o puede pararla en cualquier momento pulsando el botón “Stop Exploring”, que se puede ver situado arriba a la derecha en la figura A.3. Luego, se dirigirán al punto más alto encontrado en **Marte** y comenzarán los preparativos para la batalla contra los alienígenas.

A.2.2.3. Fase de preparación

Cada astronauta tiene una velocidad y vida distinta, que hay que tener en cuenta a la hora de asignarle las armas. Para ello, el usuario tiene a su disposición una bandera a la derecha de cada astronauta en la barra lateral a la derecha que podrá usar a modo de marcador.

El usuario también tiene a su disposición una serie de ocho botones con las caras de los astronautas y un botón “Change camera” para centrar la atención en el astronauta seleccionado y mover la cámara a su ubicación.

Una vez observado el comportamiento de los astronautas en la fase de exploración, es el momento de asignarles las armas para la batalla final. A cada astronauta se le podrá asignar una espada o un escudo. El usuario puede gestionar el número de atacantes y defensores como desee, siempre y cuando haya al menos un atacante (debido a que el juego finalizará cuando mueran todos los atacantes).

Para asignar las armas, vale con pinchar en el ícono del arma que deseé en la barra lateral de la derecha para cada uno de los ocho astronautas (A.4). Una vez se hayan asignado las ocho armas, comenzará la batalla.



Figura A.4: Fase de preparación

A.2.2.4. Batalla final

En el momento en que se asignan todas las armas, un grupo de ocho alienígenas se genera en un punto cercano a la ubicación de los astronautas.

Los alienígenas comenzarán a moverse en grupo hacia la posición de los astronautas. Una vez estén suficientemente cerca, entrarán en formación, seleccionarán sus objetivos y comenzarán a atacar. Estos alienígenas son las entidades verdes que se ven en la parte superior de la figura A.5. Su vida está representada por una barra amarilla.

Por otra parte, los astronautas esperarán hasta que los alienígenas se hayan acercado lo suficiente. Entonces, estos entrarán en formación.

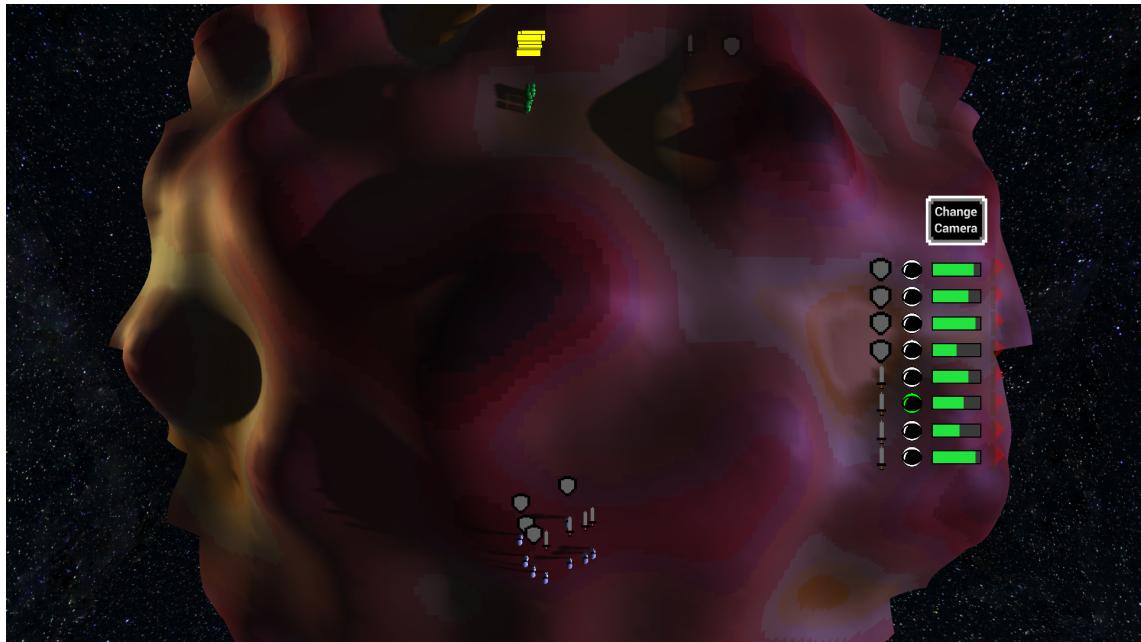


Figura A.5: Comienzo de la batalla

Los astronautas empezarán a organizarse de forma automática, de forma que no necesiten atención obligatoria por parte del usuario. Sin embargo, el usuario posee un par de herramientas para tomar decisiones en la batalla.

Esta herramienta consiste en un par de botones (A.6). El botón de la izquierda sirve para priorizar el ataque sobre un alienígena en concreto.

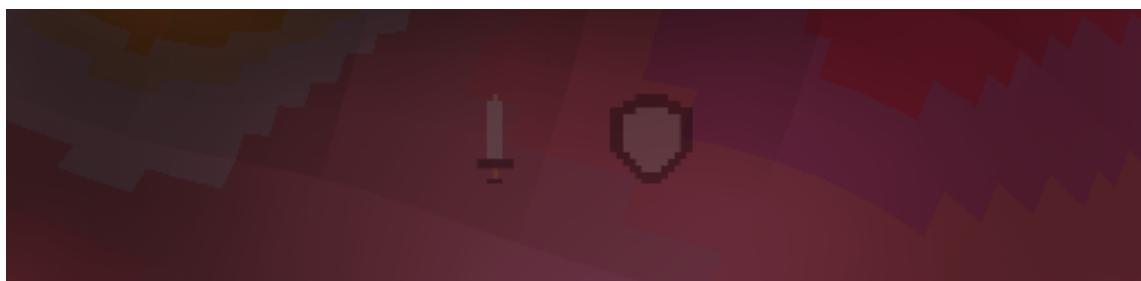


Figura A.6: Botones de priorización de ataque (ninguno seleccionado)

Al seleccionarlo, bastará con pulsar sobre el alienígena que se desee (A.7).

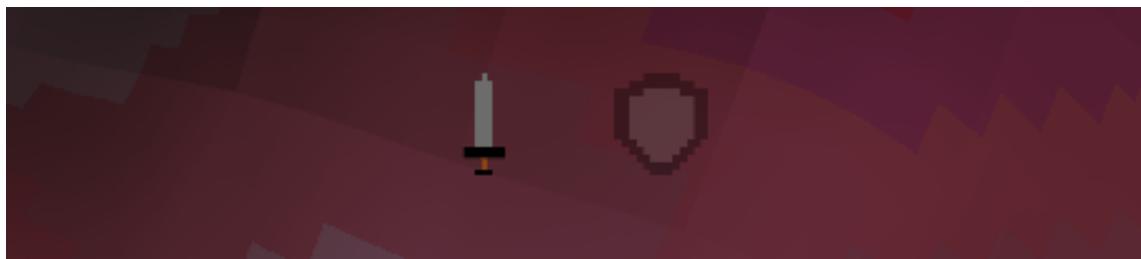


Figura A.7: Botones de priorización de ataque (seleccionado ataque)

El botón de la derecha sirve para priorizar la defensa sobre un astronauta en concreto. Los astronautas que sean defensores se organizarán teniendo en cuenta el astronauta seleccionado.

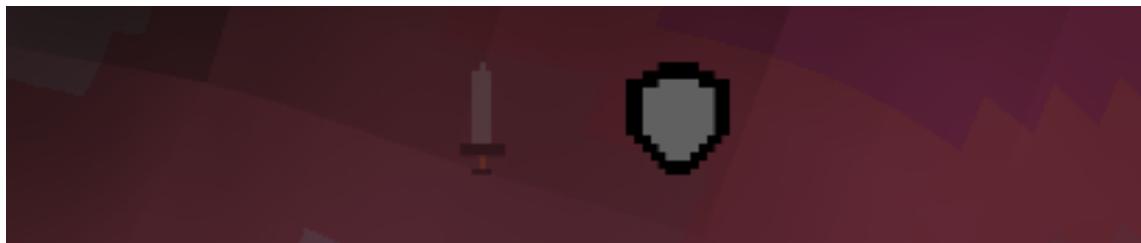


Figura A.8: Botones de priorización de ataque (seleccionado defensa)

A.2.3. Pantallas adicionales

En cualquier momento, el usuario puede pausar el juego. Esto se realizará con la tecla “ESC” (*escape*). Esto le llevará al menú de pausa (A.9). Para salir de él, el usuario puede pulsar de nuevo la tecla “ESC” o puede darle al botón “RESUME”.

Si los astronautas consiguen eliminar a todos los alienígenas, saltará la pantalla de *Mission Success* (A.10) y el usuario tendrá la opción de salir del juego.

Sin embargo, si los alienígenas consiguen eliminar a todos los astronautas con espada, el juego acabará con la pantalla *Game Over* (A.11).

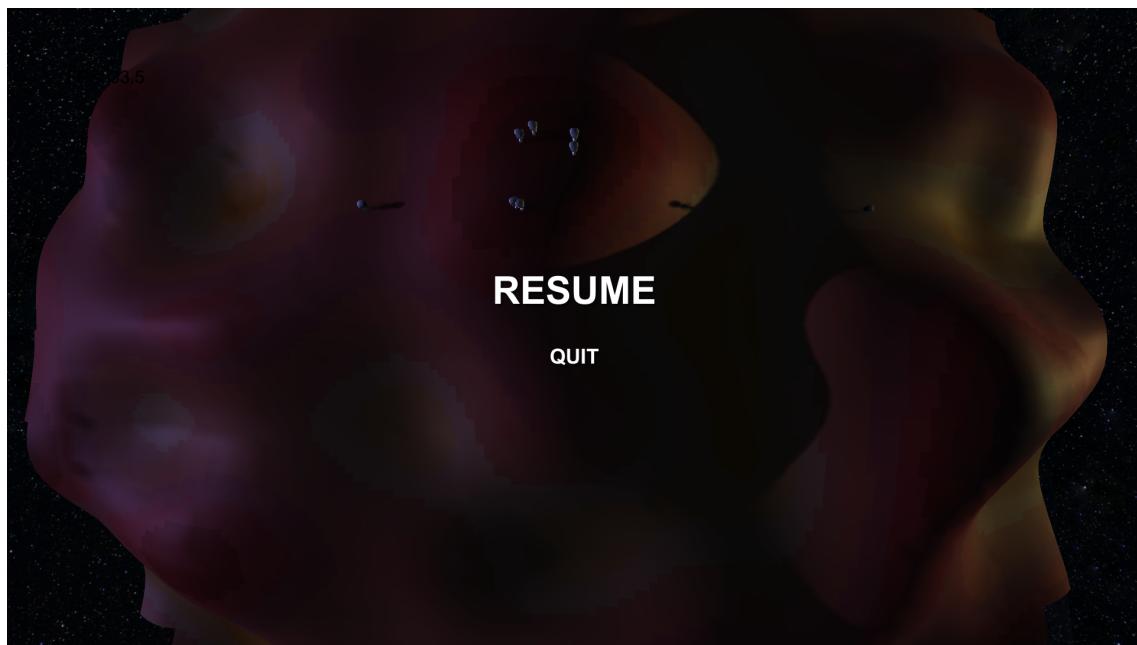


Figura A.9: Menú de pausa

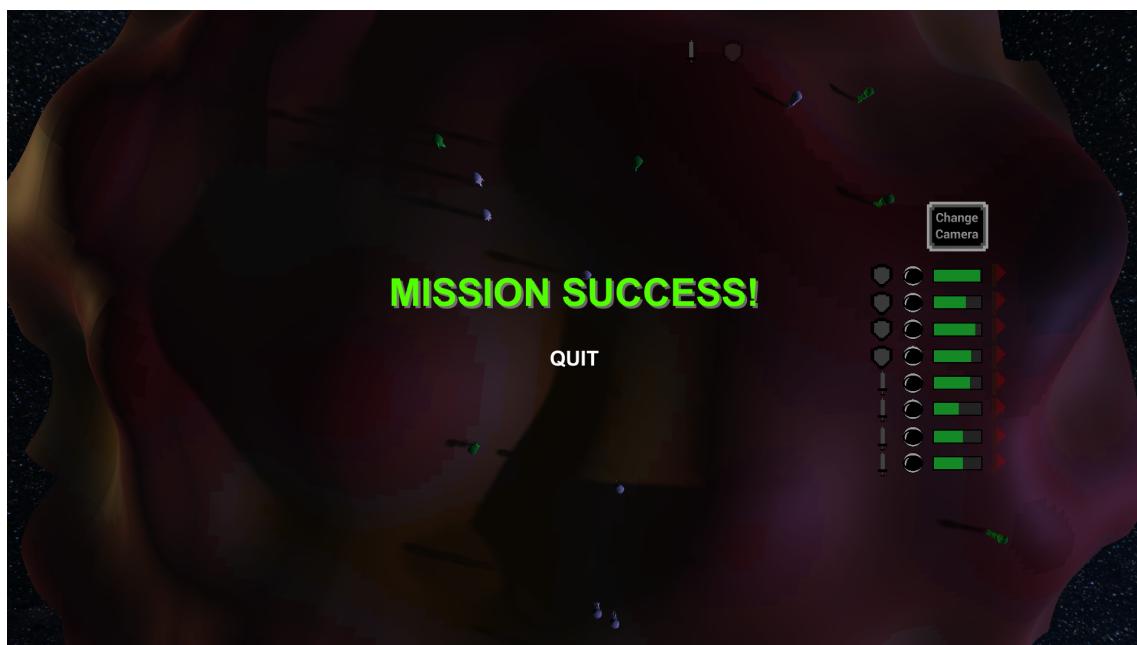


Figura A.10: Mission Success

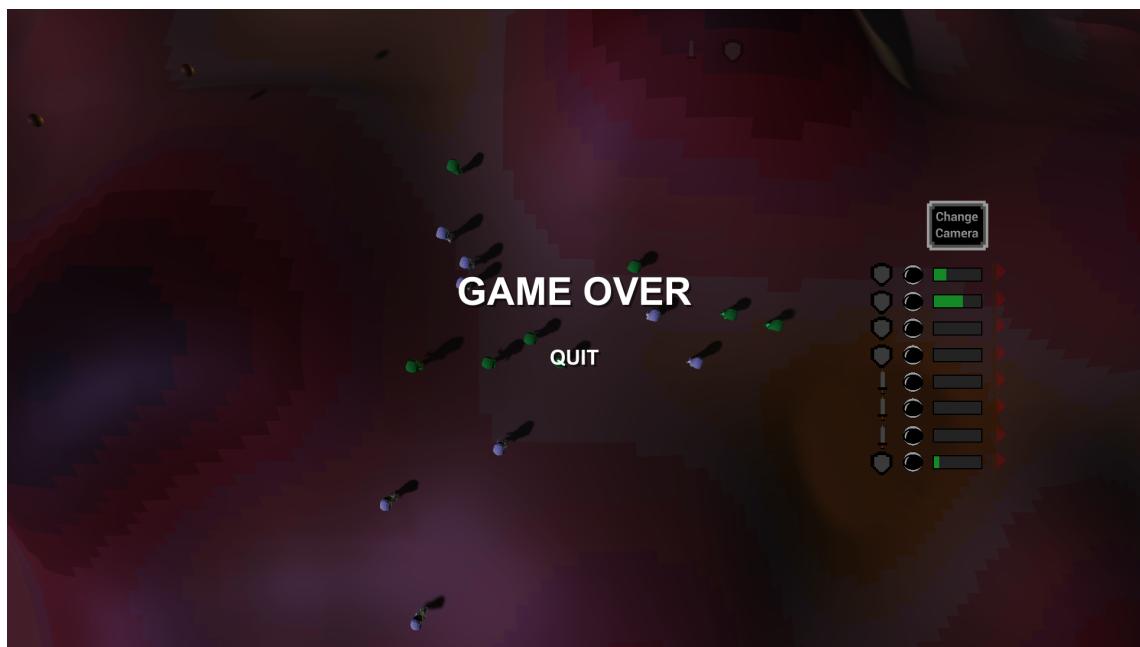


Figura A.11: Game Over

Bibliografía

- [1] John K Haas. A history of the unity game engine. 2014.
- [2] Jonathan Schaeffer and H Jaap Van den Herik. Games, computers, and artificial intelligence. *Artificial Intelligence*, 134(1-2):1–7, 2002.
- [3] F. van den Bergh and A.P. Engelbrecht. A study of particle swarm optimization particle trajectories. *Information Sciences*, 176(8):937 – 971, 2006.
- [4] Ioan Cristian Trelea. The particle swarm optimization algorithm: convergence analysis and parameter selection. *Information processing letters*, 85(6):317–325, 2003.
- [5] Jacob Olsen. Realtime procedural terrain generation. 2004.
- [6] Ian Parberry. Designer worlds: Procedural generation of infinite terrain from real-world elevation data. *Journal of Computer Graphics Techniques*, 3(1), 2014.
- [7] Henry Vuontisjärvi et al. Procedural planet generation in game development. 2014.
- [8] V. V. Sanzharov and V. A. Frolov. Level of detail for precomputed procedural textures. *Programming and Computer Software*, 45(4):187–195, Jul 2019.
- [9] Ken Perlin. Improving noise. In *ACM transactions on graphics (TOG)*, volume 21, pages 681–682. ACM, 2002.
- [10] Adi Botea, Bruno Bouzy, Michael Buro, Christian Bauckhage, and Dana Nau. Pathfinding in Games. In Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius, editors, *Artificial and Computational Intelligence in Games*, volume 6 of *Dagstuhl Follow-Ups*, pages 21–31. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2013.
- [11] David Silver. Cooperative pathfinding. *AIIDE*, 1:117–122, 2005.

Apéndice B

GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <<https://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History.”) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or non-commercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five

of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document’s license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled “History”, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <https://www.gnu.org/licenses/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.