Universidad de Zaragoza

VIDEOJUEGOS: REMAKE DEFENDER

Javier Miguel Sauco (650010)

Fernando Aliaga Ramón (610610)

José Ángel Caudevilla Casasús (649003)

Jaime Puig Ortega (647286)

TABLA DE CONTENIDO

1 INTRODUCCIÓN

Defender es un videojuego arcade de tipo side-scrolling desarrollado y publicado por Williams Electronics en 1980. Liderado por Eugene Jarvis, un programador de pinballs de Williams; *Defender* fue el primer proyecto de videojuego de Jarvis, inspirándose en *Space Invaders* y *Asteroids*. Este escrito tiene como objetivo principal plasmar los elementos que debe incluir Defender.

1.1 Concepto del juego

Defender es un videojuego arcade de tipo side-scrolling. Es un videojuego shooter de naves en un espacio bidimensional. Está situado en un planeta ficticio donde el jugador debe vencer oleadas de extraterrestres mientras protege a los humanoides situados en la superficie.

1.2 Características principales

El juego se basa en los siguientes pilares:

- **Planteamiento sencillo:** La historia mencionada es muy simple, una mera excusa para el desarrollo del juego pero lo suficientemente explícita para que el jugador sienta que tiene un objetivo.
- **Táctica**: detener a las oleadas de enemigos debe ser imposible si se comienza a atacar indiscriminadamente. La gestión de nuestras limitadas capacidades de forma inteligente será imprescindible.
- **Dinamismo**: al contrario que algunos juegos de shooter, Defender debe ser dinámico y provocar una sensación de tensión y dificultad en el jugador.
- **Ampliación**: Defender debe ser ampliable con nuevos niveles y enemigos de forma sencilla. El motor será todo lo independiente posible del contenido. De esta forma los artistas podrán generar nuevos niveles, habilidades o enemigos.

1.3 Género

Defender supone una unión de varios géneros. A continuación se listan los géneros de los que toma elementos y sus motivos:

- Arcade: A pesar de que arcade es más un calificativo que un género en

este contexto lo consideramos como un género. Defender se caracteriza por su jugabilidad simple, repetitiva y de acción rápida con jugabilidad, unos gráficos y/o un argumento simples.

- Shooter: El jugador hace un uso continuado de armas de fuego para abrirse paso en el juego.

1.3 Propósito y público objetivo

El objetivo de Defender es ofrecer a los jugadores un ejemplo de un videojuego arcade adaptado a las nueva era, se pretende demostrar que en los videojuegos lo mas importante no son los graficos sino la jugabilidad y que una historia simple puede ser igual de divertida que una más elaborada. Su interés no solo debe radicar en el código fuente y su proceso de desarrollo sino en el propio juego y sus mecánicas. Defender esta dirigido a aquellos jugadores amantes de la recreativas , a jugadores novatos que desean conocer cómo eran los juegos antiguamente.

Por ello, se apuesta por un sistema de partidas cortas en el cual la dificultad va en aumento conforme el juego avanza, se desea hacer que el juego sea lo más parecido a un juego arcade por lo tanto no existira ningun tipo de sistema de guardado automático ni comienzo de la partida en una fase posterior a la inicial en el caso de que el jugador muera o abandone la partida.

2 HISTORIA

Defender es un videojuego arcade de tipo side-scrolling desarrollado y publicado por Williams Electronics en 1980. Es un videojuego de naves en un espacio bidimensional. Está situado en un planeta ficticio donde el jugador debe vencer oleadas de extraterrestres mientras protege a los astronautas situados en la superficie.

El jugador controla una nave espacial que se desplaza por la superficie, pudiendo moverse a la izquierda o a la derecha.

El objetivo es destruir a los invasores extraterrestres, mientras que proteges a los astronautas en la superficie de ser abducidos por los extraterrestres. Los astronautas abducidos se convierten en mutantes que atacan la nave. Al vencer a los extraterrestres se avanza al siguiente nivel. Si todos los astronautas de un nivel son abducidos, el planeta explota y el nivel se llena de mutantes. Si se sobrevive a las oleadas de mutantes, el planeta vuelve a aparecer.

Figura 1. Pantalla de juego

3 DIAGRAMA DE FLUJO

El siguiente diagrama de estados muestra las pantallas presentes a lo largo de Defender y las transiciones entre ellas. En puntos posteriores nos centraremos en ellas de forma individual:

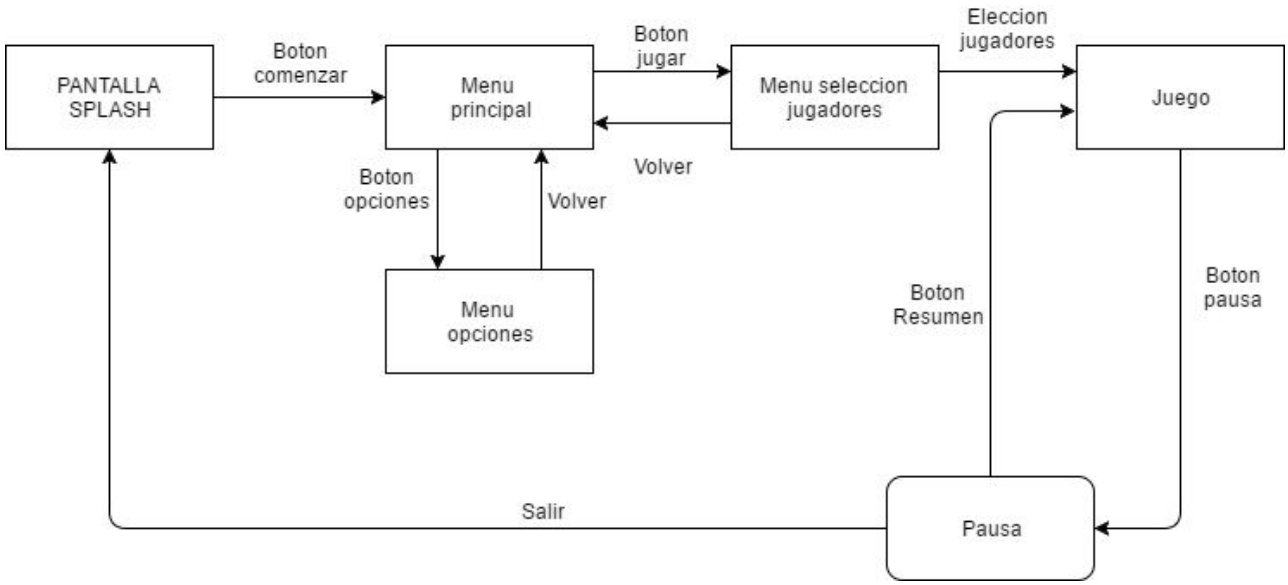

Figura 2. Mapa de navegación

3.1 Pantalla de Splash

A continuación, la pantalla de Splash:

Figura 3. Pantalla de splash

En esta pantalla aparecerá una animación del juego con el título del juego en la parte superior/central de la pantalla, en la parte de abajo aparecera un boton para iniciar el juego.

3.2 Pantalla de Inicio

A continuación el boceto de la pantalla de inicio:

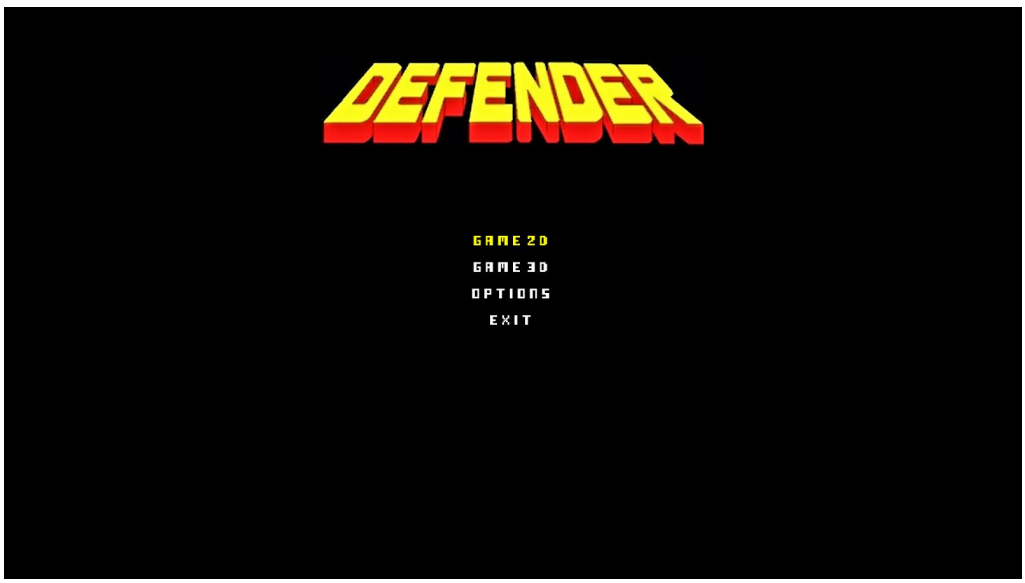

Figura 4. Pantalla de menú

En la pantalla de inicio aparecerá el título en la parte superior y en la parte inferior aparecerán 3 botones:

- Game 2Da: se inicia el juego 2D.
- Game 3D: se inicia el juego 3D.
- Opciones: se pasará a la pantalla de opciones
- Salir: se saldrá del juego.

3.3 Pantalla de opciones

A continuación el boceto de la pantalla de opciones:

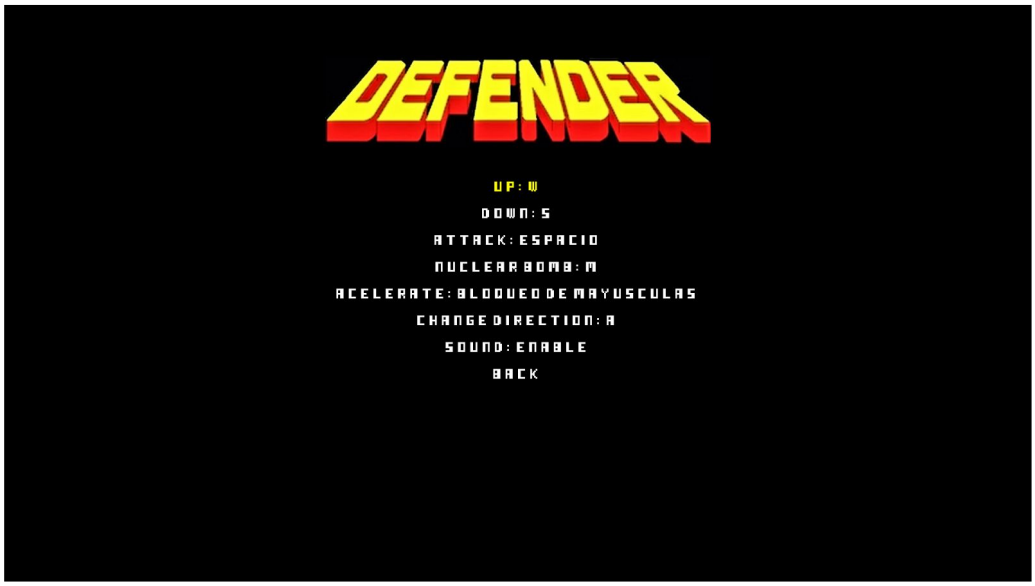

Figura 5. Pantalla de opciones

En el menú de opciones aparecerá en la parte central una serie de opciones del juego en la que el usuario elegirá la que más prefiera.

3.5 Pantalla de juego

A continuación el boceto de la pantalla de juego:

Figura 7. Pantalla de menú

Esta pantalla está dividida en 4 zonas:

- En la superior izquierda aparecerá la información del jugador: vidas, puntos, etc
- En la parte central superior aparecerá un minimapa de la partida
- En la parte superior derecha aparecerá el número de entidades terrestres que aún no hayan sido abducidas.
- En la parte inferior aparecerá la pantalla de juego.

4 Personajes e IA

El juego consta de tres tipos de personajes distintos:

- El primero es la **nave controlada por el jugador**. Es una pequeña nave espacial armada con una ametralladora y varias bombas que el jugador debe usar para acabar con los enemigos. Además tiene una opción de salto al hiperespacio para teletransportarse a cualquier punto del mapa instantáneamente.
- El segundo son los **civiles**. Son pequeños humanoides que habitan la superficie del planeta y que el jugador debe defender. Los aliens intentarán abducirlos, y si lo consiguen los convertirán en mutantes que intentarán matar al jugador. Si todos los civiles son abducidos el planeta explotara y se llenará de mutantes. El comportamiento de los civiles es un movimiento lento y aleatorio por la superficie del planeta.

- El tercero y último son los **aliens**. Son los enemigos del juego y hay de varios tipos, cada uno con armas y patrones de movimiento diferentes. El jugador debe eliminarlos para ganar la partida. El comportamiento de todos ellos está definido por una serie de reglas, en el caso del Bomber estas reglas están implementadas con una red neuronal para cumplir las especificaciones del trabajo.

    - **Lander**: el enemigo estándar. Intentará abducir a los humanoides y llevarlos al espacio exterior para transformarse en un mutante. Se mueven a velocidad media y disparan al jugador cuando lo ven.

      

      Su comportamiento está definido por una serie de reglas:
        - Los landers deben mantenerse a cierta altura, casi a ras de suelo
        - Los landers disparan cada poco tiempo hacia el jugador si lo tienen en pantalla
        - Los landers acudirán al aldeano más cercano para intentar abducirlo
        - Si los landers han conseguido atrapar un aldeano, subirán al espacio para transformarse en Mutantes.

- **Mutante**: el resultado de la abducción exitosa de un humanoide. Muy agresivo y rápido, intentará perseguir al jugador hasta matarlo o morir. Si todos los humanoides son abducidos aparecerán numerosos mutantes y el planeta explotará. Para salvar la partida habrá que eliminar a todos los mutantes del mapa para restaurar el planeta y poder continuar jugando.

Su comportamiento está definido por una serie de reglas:

- Si está en la línea de tiro del jugador lo intenta evitar acelerando.
- Intentará colocarse siempre justo encima o debajo del jugador.
- Si lo consigue, le intentará placar en un ataque suicida.
- De vez en cuando disparará.

- **Baiter**: aparecen cuando se acerca el final de nivel. Son pequeños y rápidos, intentarán chocar al jugador y le dispararán insistentemente.

Su comportamiento está definido por una serie de reglas:

- Si está en línea de tiro del jugador lo intenta evitar acelerando.
- Se mantendrá siempre a una distancia prudencial del jugador.
- Disparará insistentemente hacia el jugador para intentar matarlo.

- **Bomber**: aparecen en pequeños grupos. Se mueven de forma predecible ignorando la posición del jugador. Dejan minas indestructibles e inamovibles que matan al jugador al contacto.

Su comportamiento está definido por una serie de reglas:

- Si está en línea de tiro del jugador lo intenta evitar acelerando.
- Esporádicamente deja tras de sí minas estáticas.

Este comportamiento está regulado por una pequeña red neuronal de 2 entradas y una salida, con 4 nodos ocultos. Las entradas son la distancia relativa con el jugador, la salida es un binario, 1 acelerar, 0 mantener velocidad. La salida será uno si el jugador está demasiado cerca del bomber horizontal o verticalmente.

- **Pod**: enemigo grande y peligroso. Dispara al jugador en cuanto lo ve. Al morir liberará entre 5 y 7 swarmers para acabar con el jugador.

Su comportamiento es aleatorio, se moverá lentamente sin disparar hasta morir

- **Swarmer**: pequeños y difíciles de tratar. Persiguen al jugador de cerca y le disparan frecuentemente.

Su comportamiento está definido por una serie de reglas:

- Si está a más de media pantalla del jugador, se dará la vuelta
- Mientras esté dentro de la pantalla del jugador, seguirá en una misma dirección y le disparará.

5 Diseño de niveles

Al inicio de la partida el jugador dispone de 3 vidas, que se pierden al entrar en contacto con un enemigo o un proyectil de un enemigo. El juego termina cuando se agotan todas las vidas.

Para completar un nivel hay que destruir a los invasores extraterrestres, mientras se protege a los humanoides de ser abducidos. Una vez que todos los invasores han sido destruidos se avanza de nivel.

Si todos los humanoides de un nivel son abducidos, el planeta explota y el nivel se llena de mutantes. Si se sobrevive a las oleadas de mutantes, el planeta vuelve a aparecer.

Conforme se avanza de nivel, aparecen enemigos cada vez más poderosos, lo que aumenta la dificultad del juego de manera exponencial.

Se han implementado 4 niveles:

- Nivel 1: 15 landers.
- Nivel 2: 20 landers, 3 bombers y 1 pod.
- Nivel 3: 20 landers, 4 bombers y 3 pods.
- Nivel 4: 20 landers, 5 bombers y 4 pods

En cada nivel, los enemigos no aparecen todos de a la vez, sino que aparecen por oleadas conforme se destruyen los enemigos.

Se hace de esta forma para que cada nivel sea progresivo.

6 Arte

El apartado artístico gira en torno al estilo retro propio de las primeras máquinas arcade. Todos los elementos de la interfaz están diseñados con estilo pixelado como en la versión original del juego.

El juego carece de música salvo en los menús de inicio. Todo el sonido está compuesto en 8 bits y durante la experiencia de juego únicamente se van a oir los efectos de sonido producidos por los distintos personajes.

7 INTERFAZ DE USUARIO Y CONTROLES

La interfaz de usuario está formada por :

- En la **esquina superior izquierda**: el número de vidas del jugador y su puntuación.
- En la **parte superior central**: un escáner con la que se ve el nivel completo.
- En la **parte principal**: el juego, con el jugador y los distintos enemigos.

Figura 9. Interfaz del usuario

Los controles disponibles del juego son:

- **Joystick**: sirve para controlar la altura de la nave.
- **Acelerador:**: permite al jugador moverse hacia donde mire con la potencia que se desee.
- **Disparo**: el jugador dispara de manera ilimitada.
- **Reverso**: el personaje del jugador cambia el sentido de la nave.
- **Hiperespacio**: permite al jugador desaparecer y aparecer en otra parte del mapa.
- **Bombas especiales**: cada jugador tiene tres de estas bombas al principio del juego, aunque cada vez que se consiguen 10.000 puntos se gana una adicional. Estas bombas sirven para destruir todos los enemigos en pantalla.

8 Cronograma y reparto de tareas

El cronograma de la primera parte de nuestro proyecto ha sido:

- 10 de Marzo: primera reunión y reparto de tareas.
- 15 de Marzo: estructura de clases del proyecto.
- 17 de Marzo: mapa estático del jugador.
- 20 de Marzo: lógica de puntuación y de vida.
- 25 de Marzo: gestión de colisiones y movimiento del jugador.
- 27 de Marzo: mapa circular y minimapa.
- 30 de Marzo: inteligencia artificial.

En los días posteriores se realizaron diferentes pruebas de lo implementado.

Después de la demo y la presentacion del dia 4 de abril, el cronograma ha sido:

- 10 de Abril: reunión para marcar los pasos a dar tras la demo.
- 14 de Abril: primeros pasos 3d.
- 18 de Abril: primeras mejoras del 2d (parte gráfica)
- 20 de Abril: mejoras en cuanto a la lógica del juego.
- 21 de Abril: IA de los 3 primeros enemigos.
- 25 de Abril: 3d, colisiones y disparos.
- 28 de Abril: IA de los 3 enemigos restantes y su red neuronal.
- 29 de Abril: finalización de la parte grafica y lógica del juego
- 1 de Mayo: finalización del 3d
- 2 de Mayo: pruebas finales del juego.

Por otro lado, el reparto de tareas de la primera parte ha sido el siguiente:

- Como primer paso en todo proyecto escogimos nuestro jefe del proyecto. En este caso el elegido fue José, cuyas habilidades como dirección encajaban a la perfección con el rol. Una vez definida la cúspide de nuestra pirámide de equipo, dividimos las tareas principales para una primera subdivisión del problema:
  - Mapa circular, minimapa y gestión de colisiones : José
  - Lógica de puntuación, vida y movimiento del jugador: Javier
  - Inteligencia artificial : Fernando y Jaime
- A través de un entorno compartido de Git, hemos desarrollado independientemente, en la medida de lo posible, las distintas partes asignadas a cada uno. A medida que el proyecto aumentaba de tamaño y las diferentes partes se entrelazaban, hemos realizado varias reuniones de

equipo para la conexión del código y la lógica del juego.

En la segunda parte del desarrollo del videojuego el reparto de tareas ha sido el siguiente:

- Solución de errores del 2d y su finalización: Javier y Fernando.
- Solución de errores de la IA y su finalización: Jaime.
- 3D: Jose.

9 Problemas encontrados

Los mayores problemas de la primera fase encontrados han sido;

- En cuanto al mapa circular, la gestión de las posiciones absolutas del jugador fueron problemáticas.
- La gestión de colisiones nos retrasó, en gran medida por los cálculos de las mismas.
- En cuanto a la inteligencia artificial encontramos problemas por el retraso a la hora de encontrar un sistema fiable de coordenadas para los distintos objetos dinámicos de nuestro proyecto.

En cuanto a la segunda fase los principales problemas han sido:

- Las colisiones nos dieron más problemas y nos costó solucionarlos.
- Hubo dificultades a la hora de encontrar una configuración para las redes neuronales que fuera óptima para el resultado deseado y suficientemente fácil de entrenar.
- En cuanto al 3D, el mayor problema fueron las colisiones y la falta de experiencia en trabajo en 3D.

10 Bucle principal de juego

Cuando se inicializa el juego se crea un thread que permite actualizar y pintar los elementos en pantalla limitando el número de fps.El bucle principal está formado por una fase de inicialización, una de actualizacion, pintado y delay para pausar el juego.

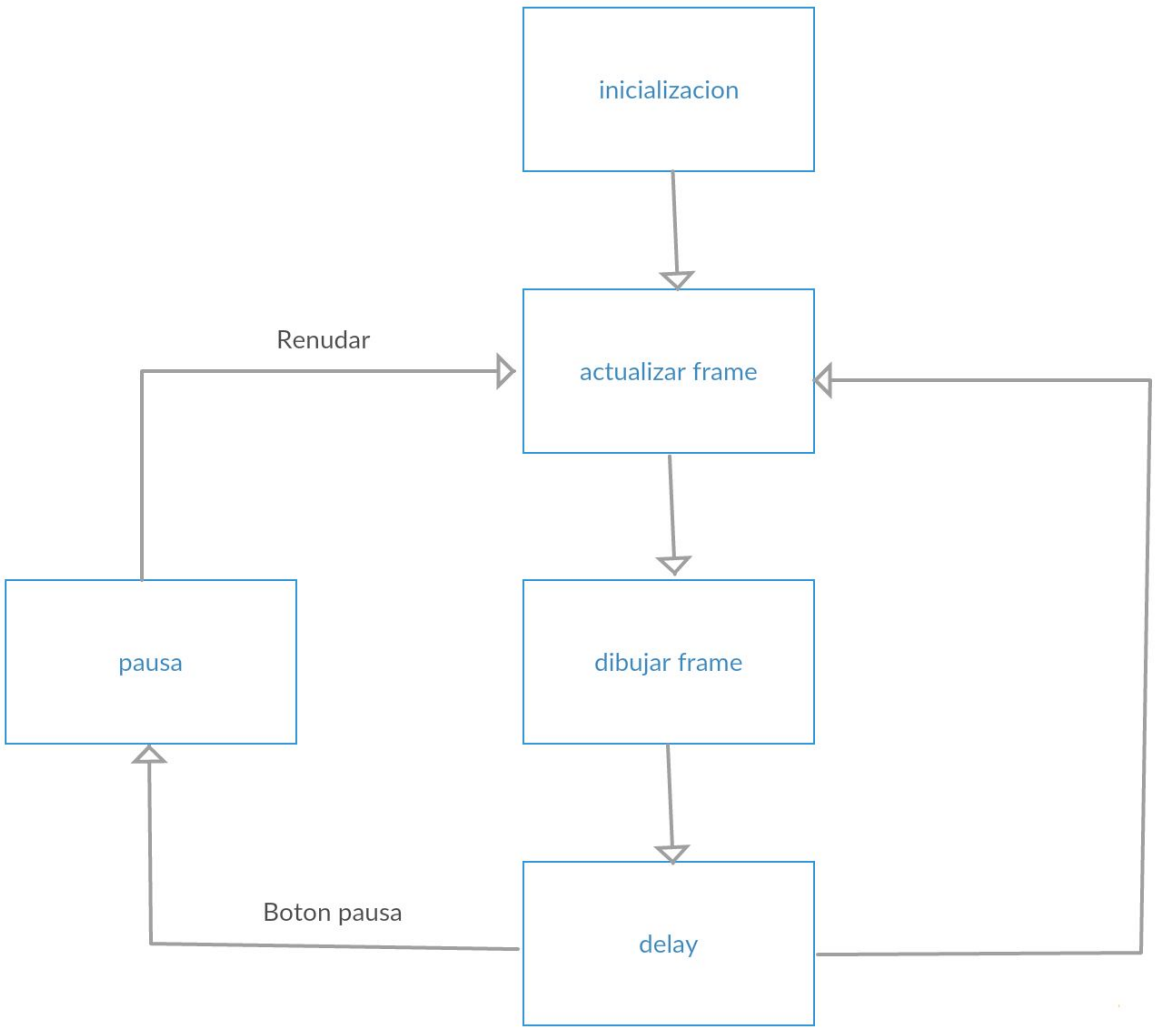

Figura 10. Bucle principal del juego

- Inicialización: Este bloque se encarga de inicializar todos los elementos necesarios para crear una pantalla de juego.
- Actualizar Frame: En este bloque se actualizan todas las entidades que tiene el juego en el frame actual, una entidad se considera cualquier elemento que se pueda pintar en el juego.
- Dibujar frame: Aquí es donde se dibujan en pantalla todas las entidades que tiene el frame actual, se utiliza un doble buffer , un buffer va almacenando todas las imágenes del frame actual y otro muestra por pantalla, en el momento en el que se han cargado todos los elementos el en buffer de almacenamiento se carga este en pantalla.
- Delay: Se duerme bucle principal un tiempo equivalente al limite de frames por segundo.

- Pausa: Cuando el jugador decide pausar, el juego deja de actualizar y pintar las entidades que hay actualmente a la espera de que se vuelva a reanudar el juego.

11 SIMULACIÓN DE PLANETA Y MINIMAPA

11.1 Simulación de planeta

Una de las características principales del videojuego es que se desarrolla en un planeta "circular". Para simular este efecto se han establecido unos limites de mapa en los ejes x e y, en el eje y el espacio está limitado pero en el eje x no hay restricciones.

Cuando una entidad atraviesa uno de los límites del mapa en el eje x se mueve al otro limite del mapa , por ejemplo si una entidad escapa del borde derecho del mapa se desplaza hasta el borde izquierdo del mapa, de esta manera conseguimos que se dé una sensación de "mapa circular". Pero hay un problema y es que cuando el jugador llega al límite del mapa no ve lo que hay al otro lado, para solucionar esto se ha calculado la distancia que no ve el jugador del límite contrario del mapa y se han dibujado las entidades que están dentro de esta distancia, los resultados obtenidos se muestran en la siguiente imagen donde la línea roja representa el límite del mapa:

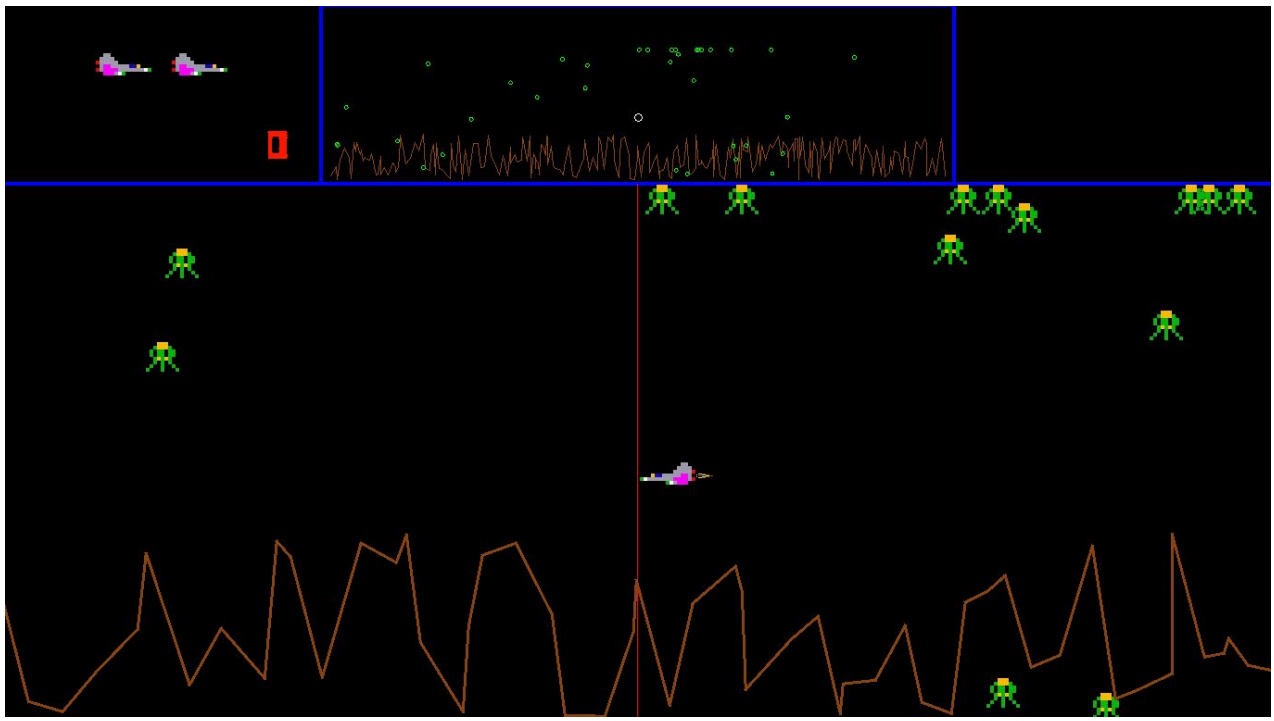

Figura 11. Mapa circular

A la hora de dibujar el mapa de juego se dibuja primero la GUI en posiciones fijas así como la posición del jugador (siempre está en el centro de la pantalla), cuando se desea pintar las entidades cercanas al jugador que se encuentran dentro de la pantalla se utiliza un matriz de transformación que lo que hace es convertir posiciones absolutas a posiciones relativas al jugador, de esta manera se consigue ver solo aquellas entidades que están al alcance del jugador, como el resto de entidades no nos interesa dibujarlas solo pintamos las entidades que son visibles.

11.2 Minimapa

Otra de las características del juego es que posee un minimapa donde se indica la posición de los enemigos y de los ciudadanos y se mueve en función de la posicion del jugador, el minimapa es considerado como una entidad estáticas ya que forma parte de la gui.

Para realizar el minimapa se pinta al jugador en centro del eje x del juego y minimapa y se “mueven” las entidades y el mapa en función de la distancia que ha recorrido el jugador con respecto a esa posición inicial, si la distancia de una entidad al jugador es mayor que la mitad de la longitud se pinta en el lado contrario del mapa para indicar que esa entidad esta mas cercana en la otra direccion, por ejemplo cuando hay una entidad a la izquierda y el jugador se mueve a la derecha se esta alejando por la izquierda pero se acerca por la derecha.

12 Gestión de colisiones

Para detectar la colisión entre entidades se han usado celdas de colisión, el mapa se ha dividido en una matriz de celdas de tamaño n cada una en la que cada una de ellas se almacenan las entidades que hay dentro de esa celda.

De esta manera no es necesario calcular la intersección de todas las entidades del mapa sino de aquellas que estén dentro de una determinada celda. Para en que celda colocar una entidad se calcula a partir de su posicion la celda a la que pertence, puede darse el

caso de que una entidad se encuentre en varias celdas, como en el ejemplo mostrado abajo, a parte de calcula la posicion se calcula la distancia que ocupa la entidad y en el caso de que este en mas de una celda se añade en todas las celdas en las que se encuentra.

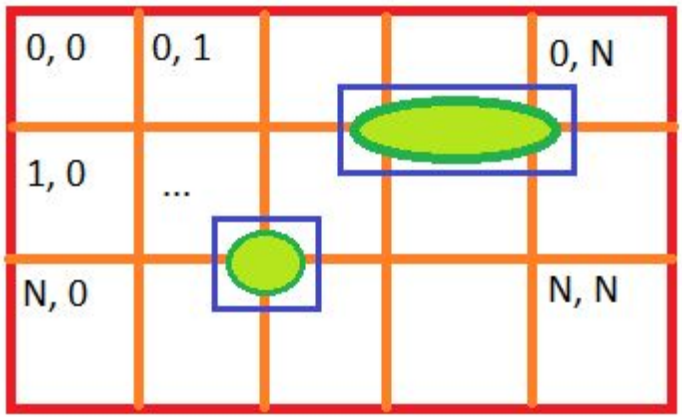

Figura 11. Gestión de colisiones

Cuando queremos actualizar la posición de una entidad, lo primero que se hace es borrar la entidad de todas las celdas que están ocupando, se actualiza su posición y se "inserta" la entidad dentro de la celda o celdas que ocupe.

13 Modo 3D

En el siguiente apartado muestra los pasos seguidos para la creación de un modo de juego 3d usando funciones nativas de openGL.

13.1 Tecnología usada

Para la creación de la versión 3d del juego se ha usado la librería LWJGL (Biblioteca Java Ligera para Juegos), esta librería ofrece bibliotecas como openGL y openAL, para el manejo de matrices y vectores se ha usado la librería matemática JOML.

13.2 Cámara

La cámara es un punto en el espacio que posee un vector de posición y otro de rotación, por lo tanto una cámara se puede mover y rotar en cada uno de los ejes.

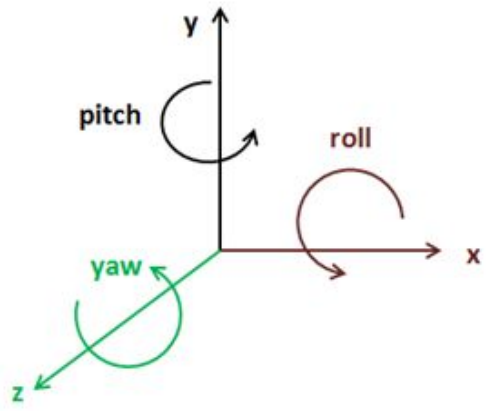

Figura 12. Movimientos de la cámara

Mediante una matriz de vista se trasladan todos los vértices de los objetos en la posición opuesta al movimiento de la camara y despues rotandolos de acuerdo con la rotación de la cámara.

13.3 Carga de modelos .obj

Para cargar modelos con formato .obj se ha creado un parser que leer un fichero en formato .obj el cual extrae todos los vertices(v) que posee el modelo asi como la normal(vn) y las caras(f). una vez almacenados todos los vertices en un array de flotantes se envian estas corrdenadas al VBO (Vertex Buffer Object) convirtiendo las coordenadas a un formato que sea reconocido por la tarjeta grafica, de esta manera se obtiene un mejor rendimiento. Para agrupar coordenadas , normales y color se usa un VAO(Vertex Array Object).

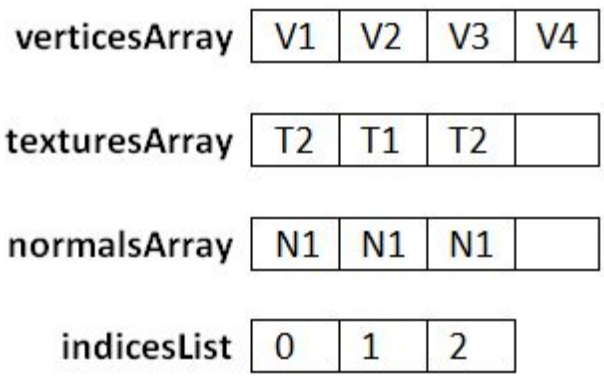

Figura 13. Estructura modelo .obj

Nota: no se han usado texturas en modelos complicados porque daban problemas

El resultado obtenido se muestra en las siguientes captura:

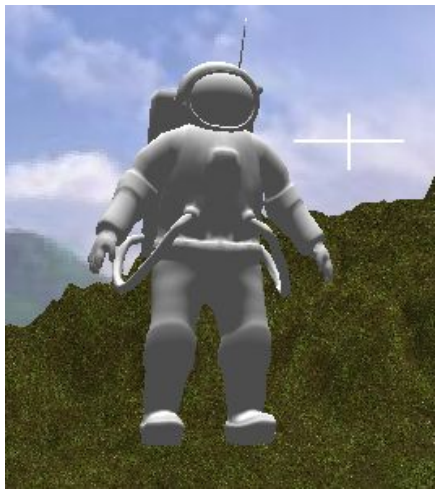

Figura 14.modelo .obj

13.4 Texturas

Para aplicar texturas a un objeto se ha asignado a cada vértices coordenadas de la textura , se ha usado el VBO donde se almacenan las coordenadas de la textura para cada pixel. Para descomponer la imagen en un formato RGBA y almacenarlo en el VBO se ha usado la librería PNG Decoder.

Figura 15. ejemplo de texturas

13.5 Mapa y skybox

Para generar un mapa de profundidad primero se ha generado una malla de vértices como se muestra en la siguiente figura:

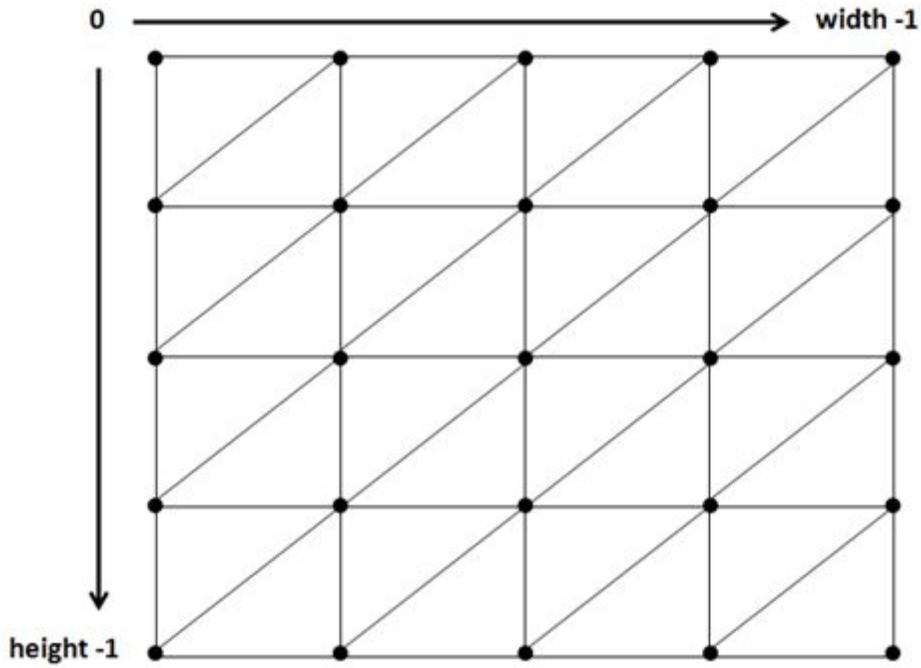

Figura 16. maya del terreno

Para generar la profundidad del mapa se ha usado una imagen en escala de grises

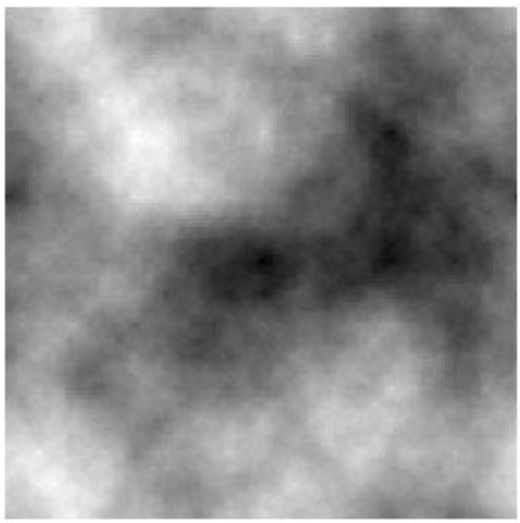

Figura 17. Altura del terreno

El proceso para crear un mapa 3d ha consistido en:

- Cargar la imagen que contiene el mapa de profundidad.
- Para cada pixel de la imagen se creado un vértice a una determinada altura dependiendo del color del pixel.
- Asignar una textura al pixel.
- Crear los triángulos asociados a cada vértice

Para poder iluminar el terreno correctamente se ha calculado la normal calculando primero los vectores tangentes a la superficie Po=(V1=P1-P0).

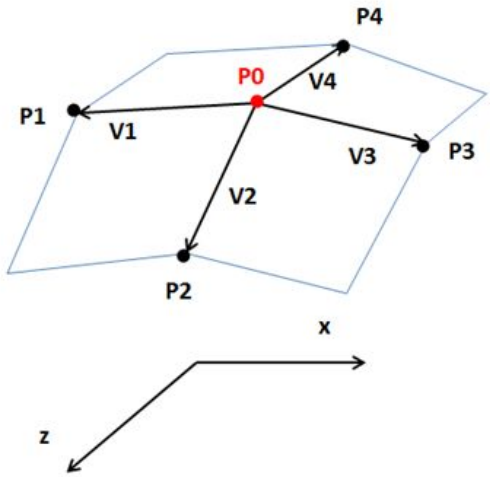

Figura 17. Calculo vectores tangentes

Una vez obtenidos los vectores tangentes a la superficie se ha realizado el producto vectorial de cada uno de ellos obteniendo asi la normal para cada uno de los planos del terreno:

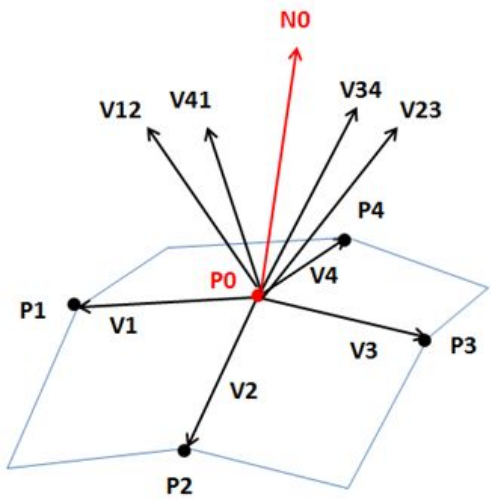

Figura 18. Cálculo normales mapa

Para la creación del sky-box se ha seguido el siguiente procedimiento:

- Se ha creado un cubo que ocupe todo el mundo
- Se le ha aplicado una textura para crear la ilusión de cielo.
- Se ha ajustado la textura para que concuerde con la posición de la camara y no de la sensación de que el mundo es un “cubo”.

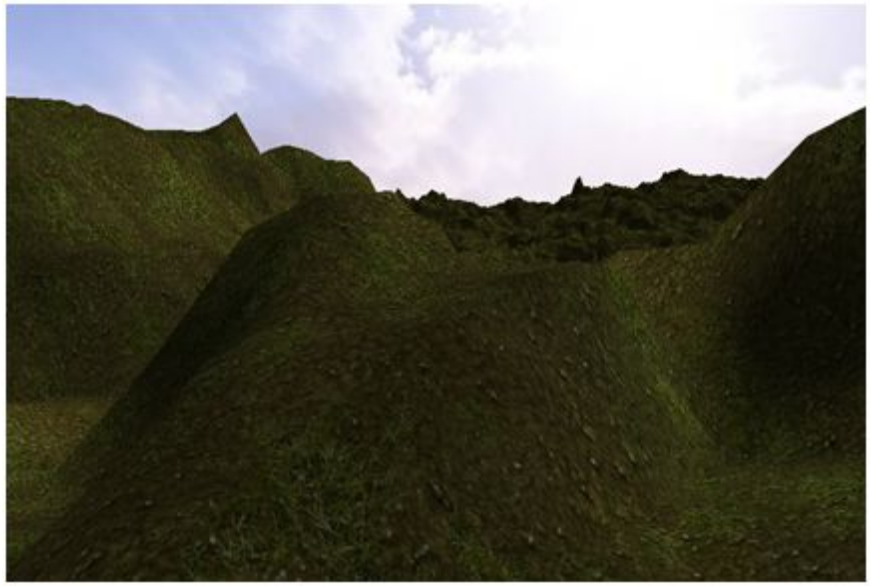

Figura 18. Ejemplo de terreno

13.6 Resultados

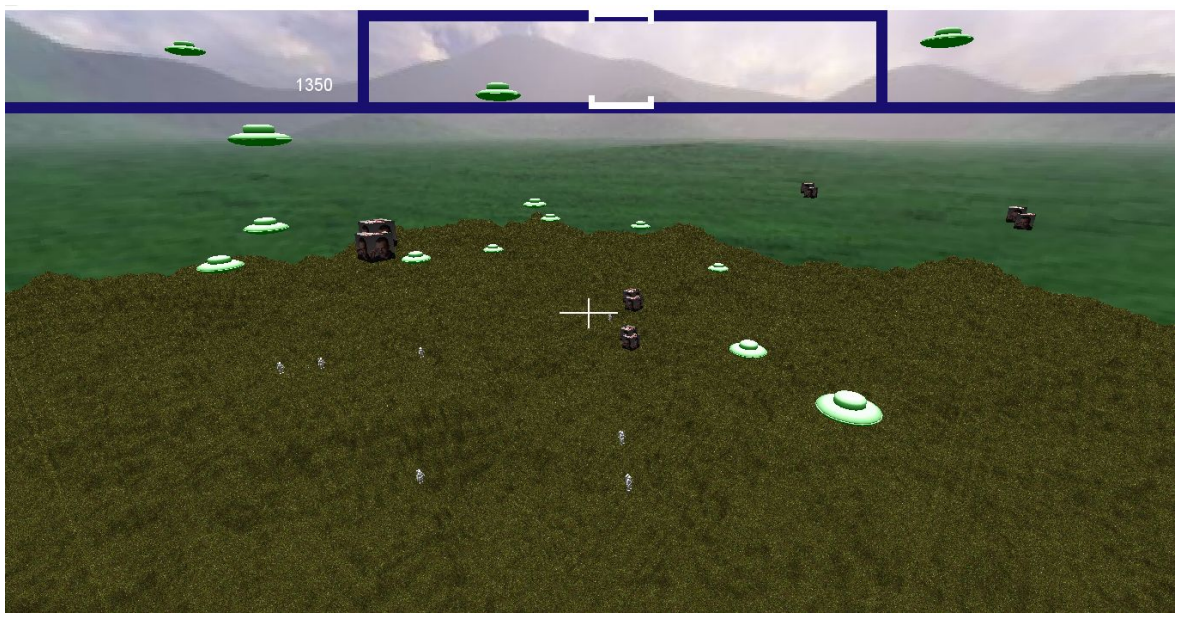

Figura 19. Resultados obtenidos

14 DIAGRAMAS UML

En el siguiente apartado se describe la estructura del videojuego por módulos así como las decisiones tomadas

13.1 Entrada de teclado

A continuación de detalla el esquema desarrollado para leer la entrada del teclado:

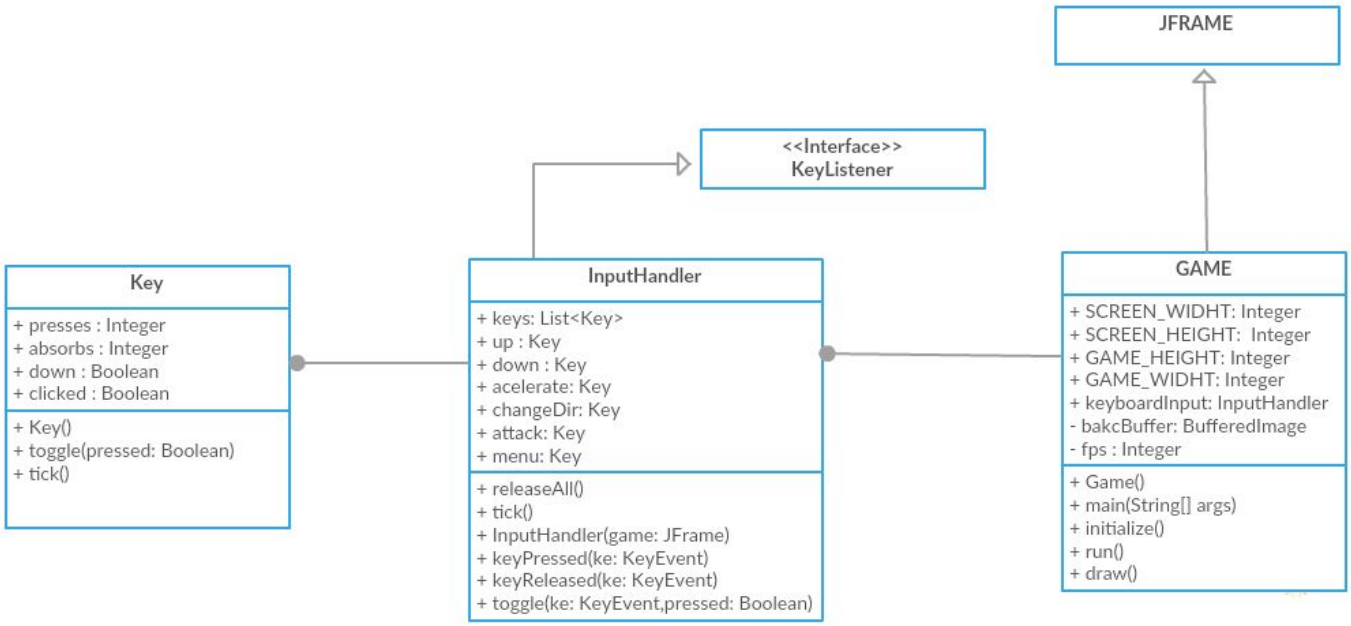

Figura 20. Diagrama UML entrada teclado

La entrada de teclado se compone de 3 clases , la clase Game, Key e InputHandler:

- Game: Clase principal que hereda de la clase JFrame y es la encargada de inicializar todo el juego, posee como atributos las dimensiones de la pantalla y del juego, un objeto de tipo InputHandler y un número máximo de fps a soportar(como defecto son 120 fps). Posee como métodos un constructor , la funcion principal main, el método initialize() se encarga de inicializar la

pantalla del juego, el método run es el encargado de inicializar el bucle principal del juego y el método draw() se encarga de cargar de dibujar por pantalla.

- Key: Clase que permite gestionar los botones pulsados, posee atributos para saber si un botón ha sido presionado,absorbido, está pulsado y ha sido clikado. Como métodos posee un constructor que inicializa el boton, una variable para contar el número de ticks y un método toggle() para cambiar el estado del botón.
- InputHandler: Clase que permite controlar la entrada y salida del teclado del ratón. Posee como atributos una lista de key y los botones usados en el juego. El método constructor tiene como atributos un objeto de tipo JFrame y posee métodos para controlar qué botón se pulsa, eventos de pulsado y soltado de botones y cambio de estado de los botones.

Al diferenciar cada uno de los botones pulsados desde el principio permite usar el mismo objeto durante toda la duración del juego sin necesidad de crear nuevas estancias / duplicados de la clase InputHandler.

13.2 Gestión de pantallas

A continuación de detalla el esquema desarrollado para gestionar la diferentes pantallas del videojuego :

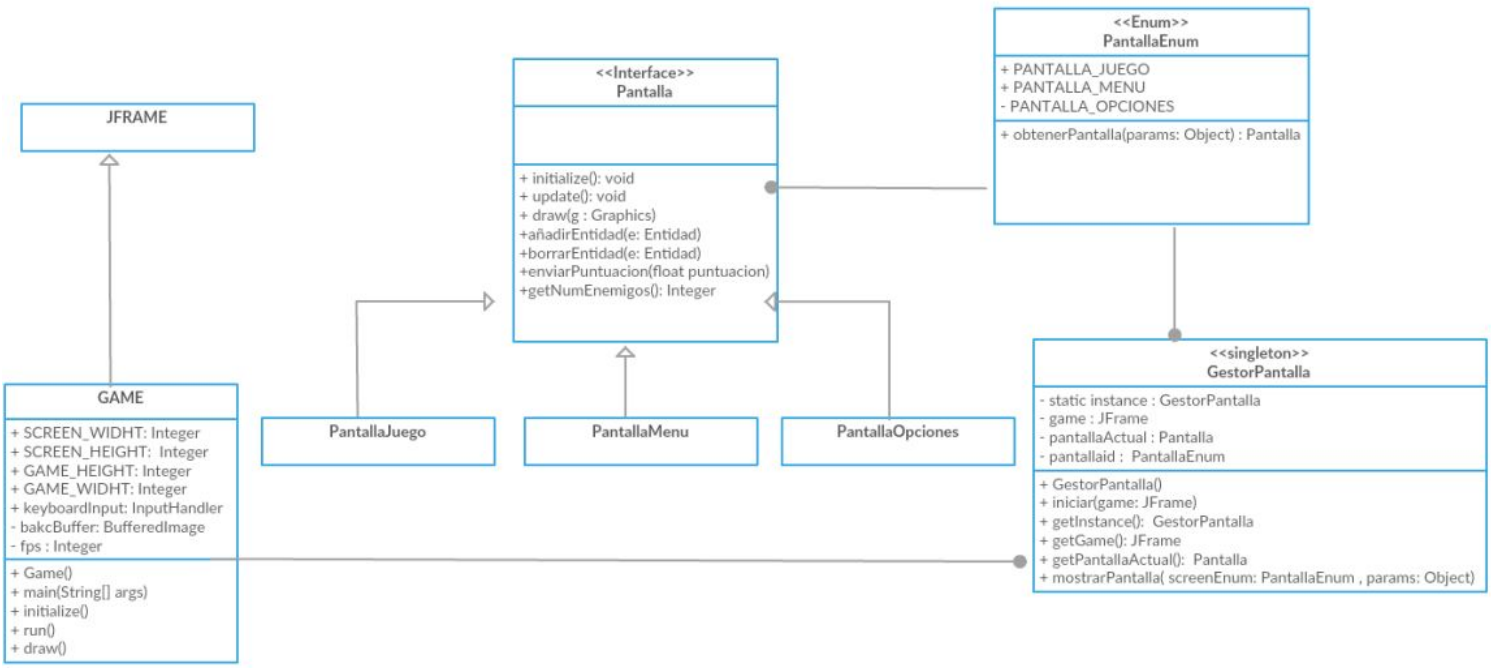

Figura 21. Diagrama UML gestión de pantallas

La gestion de pantallas se compone de 4 clases principales: Game, Pantalla, PantallaEnum, GestorPantalla y de 3 clases secundarias: PantallaJuego, PantallaMenu, PantallaOpciones:

- Pantalla: Interfaz que contiene los métodos necesarios para definir una pantalla, cualquier pantalla del juego debe implementar esta interfaz. Posee principalmente los métodos initialize() que sirve para inicializar la pantalla, update() que es la encargada de actualizar todos los elementos de la pantalla, draw() que se encarga de dibujar todos lo elementos en el videojuego, añadirEntidad() para añadir entidades en la pantalla actual,

borrarEntidad() para borrar entidades de la pantalla.

- PantallaEnum: Dato enumerado que identifica cada una de las pantallas del juego, posee el método obtenerPantalla() que devuelve una instancia de la clase a la que pertenece el identificador de la pantalla.
- GestorPantallas: Clase que utiliza el patrón de diseño singleton. Esta clase es la encargada de cargar las pantallas del videojuego. Posee como atributos una instancia de la clase GestorPantallas , el juego al que pertenece, la pantalla que se está cargando actualmente y su identificador.Posee el método getinsantiate() para obtener la instancia, un método iniciar() que tiene como parámetro un objeto de la clase JFrame que inicializa el gestor y el método mostrarPantalla() es el encargado de cargar una pantalla en el juego a partir de un dato enumerado.

Se ha decidido usar el patrón de diseño singleton para la gestión de pantallas del juego para evitar la creación de pantallas duplicadas así como la creación errónea de pantallas, y se ha usado una interfaz para definir las pantallas de manera que el el juego sea lo más escalable posible y facilite la implementación de nuevas pantallas.

13.3 Entidades

A continuación de detalla el esquema desarrollado para la creación de entidades, una entidad es considerada como cualquier elemento que se dibuja en el juego.

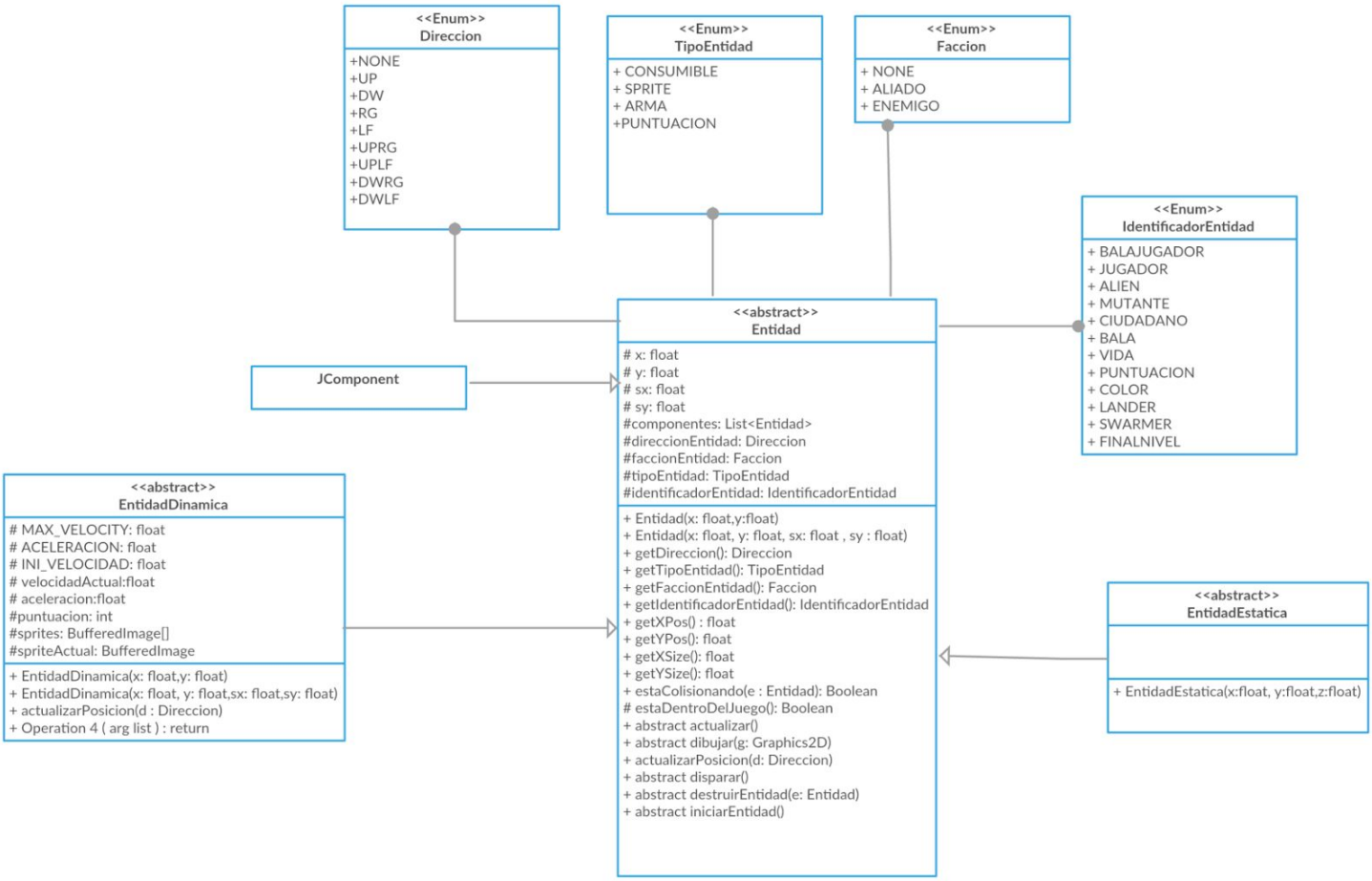

Figura 22. Diagrama UML entidades

La jerarquia de las entidades está formada por 4 datos enumerados: Direccion,Faccion, TipoEntidad, IdentificadorEntidad y 3 clases : Entidad, EntidadDinamica e EntidadEstatica.

- Direccion: Identifica la dirección de la entidad en un determinado frame , una entidad puede estar quieta, moverse arriba,abajo,izquierda,derecha o en diagonal.
- TipoEntidad: Identifica el tipo de entidad al que pertenece un objeto, una entidad puede ser un sprite, un arma, un consumible o una puntuación.
- Facción: Identifica la faccion a la que pertenece una entidad, una entidad puede ser un aliado, un enemigo o no pertenecer a ninguna entidad.

- IdentidicadorEntidad: Este dato enumerado identifica el tipo específico de entidad, como un jugador, un ciudadano, la vida del jugador…
- Entidad: Clase abstracta que incluye todos los atributos y métodos necesarios para dibujar “algo” en pantalla. Como atributos tiene una direccion,un tipo de entidad, una faccion, un identificador de entidad, la posicion en el juego asi como su tamaño y entidades hijo que posea. Los metodos que contiene esta clase son, un metodo constructor en el que se define una posición, un metodo constructor donde se define una posición y un tamaño de entidad, métodos getter y setter para obtener y modificar atributos, métodos para actualizar y dibujar la entidad y métodos para gestionar la colisión con otras entidades asi como si la ubicación de la entidad se encuentra dentro del mapa.
- EntidadDinamica: Clase abstracta que define cualquier entidad que posea velocidad y aceleración. Esta clase contiene atributos para almacenar la velocidad actual de una entidad, la aceleración que posee asi como su limite de velocidad.
- EntidadEstatica: Clase abstracta que define cualquier entidad que no cambia de posición durante el desarrollo del videojuego.

Se ha decidido crear esta jerarquía para que la creacion y programacion de entidades sea lo más breve posible y evitar el uso de atributos y variables repetidas, asi como la separación de entidades dinámicas y estáticas permite el ahorro de uso de variables y espacios innecesarios en memoria.

13.3.1 Entidades dinámicas

Anteriormente se ha definido una entidad dinámica como cualquier entidad que posea velocidad y aceleración, será el caso de un jugador , un enemigo, un disparo...

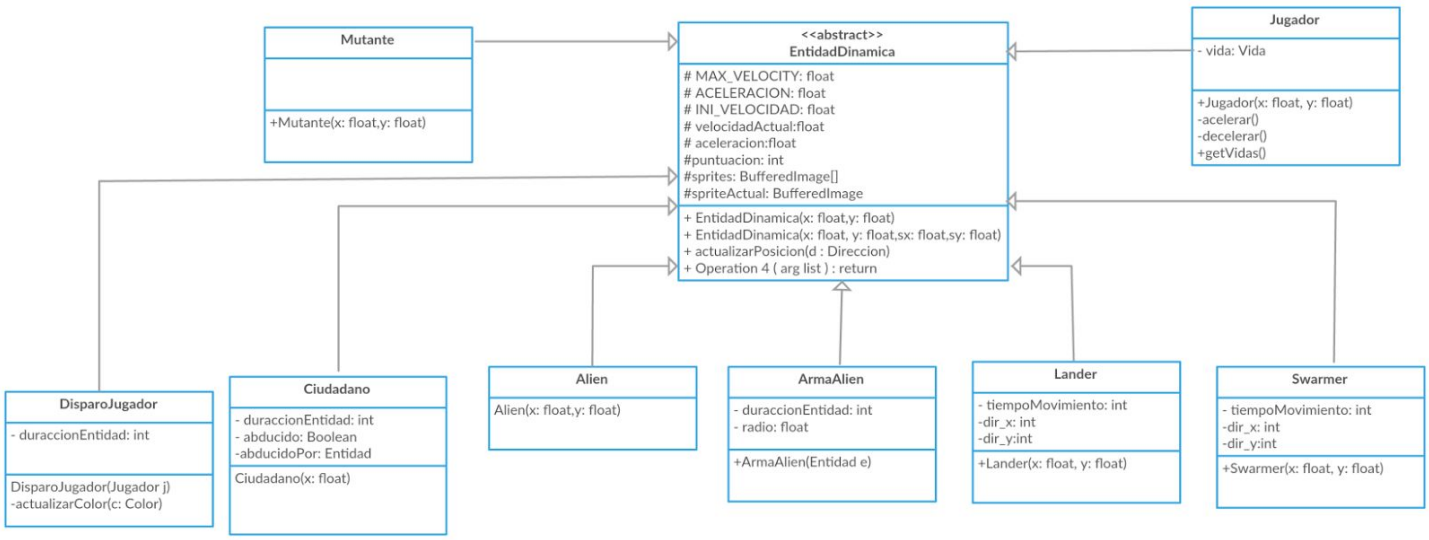

Figura 23. Diagrama UML entidades dinámicas

Cualquier cosa que posea movimiento en el juego debe heredad de la clase EntidadDinamica, de esta manera el comportamiento común de las entidades está definido y asi solo es necesario centrarse en el comportamiento específico de cada entidad como por ejemplo mover el jugador cuando el usuario toca una tecla de movimiento.

13.3.1 Entidades estáticas

cualquier entidad que no cambia de posición durante el desarrollo del videojuego debe heredar de la clase EntidadEstatica, a esta clase pertenecen las entidades mapa,vida, puntuación...

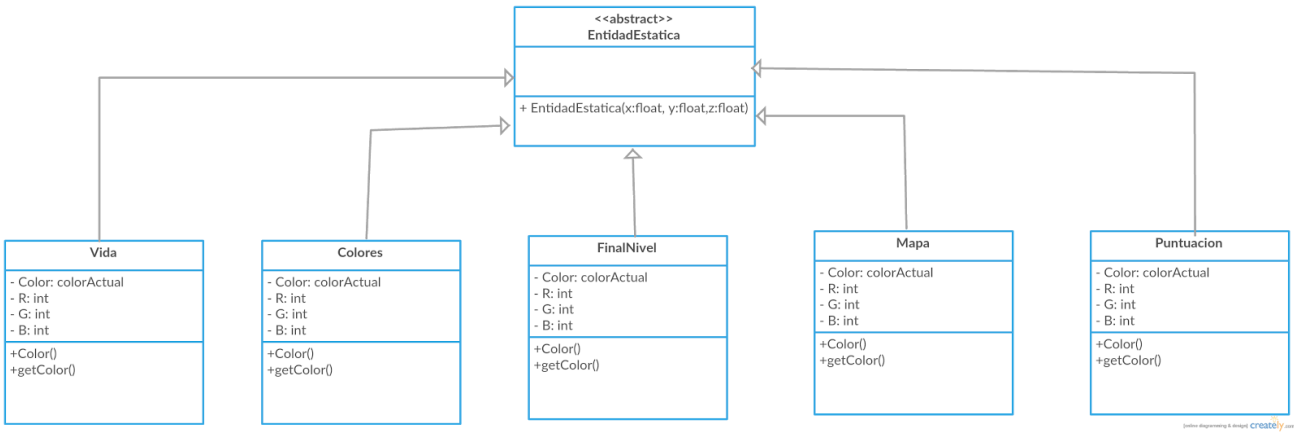

Figura 24. Diagrama UML entidades estaticas

13.4 Pantalla de Juego

A continuación de detalla la estructura de la pantalla de juego

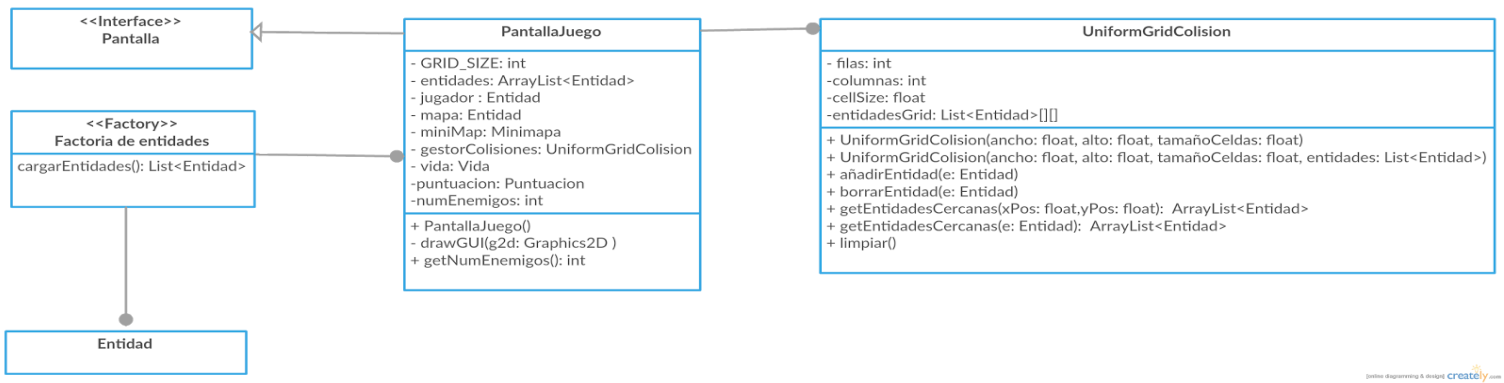

Figura 25. Diagrama UML juego

La pantalla de juego está formada por 4 elementos: La clase encargada de visualizar y actualizar los frames del juego, una factoría de entidades que se encarga de cargar las entidades que hay dentro del juego y un sistema de gestión de colisiones:

- PantallaJuego: Esta clase implementa la interfaz Pantalla , esta clase se encarga de actualizar todas la posiciones de la entidades asi como de gestionar todas las colisiones y dibujar todas las entidades.
- Factoría de entidades: Se encarga de inicializar todas las entidades del juego y enviarlas a la pantalla de juego.
- UniformGridColision: Clase que almacena las entidades que se encuentran dentro de un grid de una determinada posición, posee como atributos una matriz de lista de entidades que representa las entidades cercanas, métodos para añadir una entidad en un grid, borrar una entidad de un grid, obtener las entidades cercanas a un grid y limpiar todas las entidades.

15 Anexos

14.1 Diagrama de paquetes

En la siguiente imagen se muestra como está organizada a estructura del proyecto por paquetes:

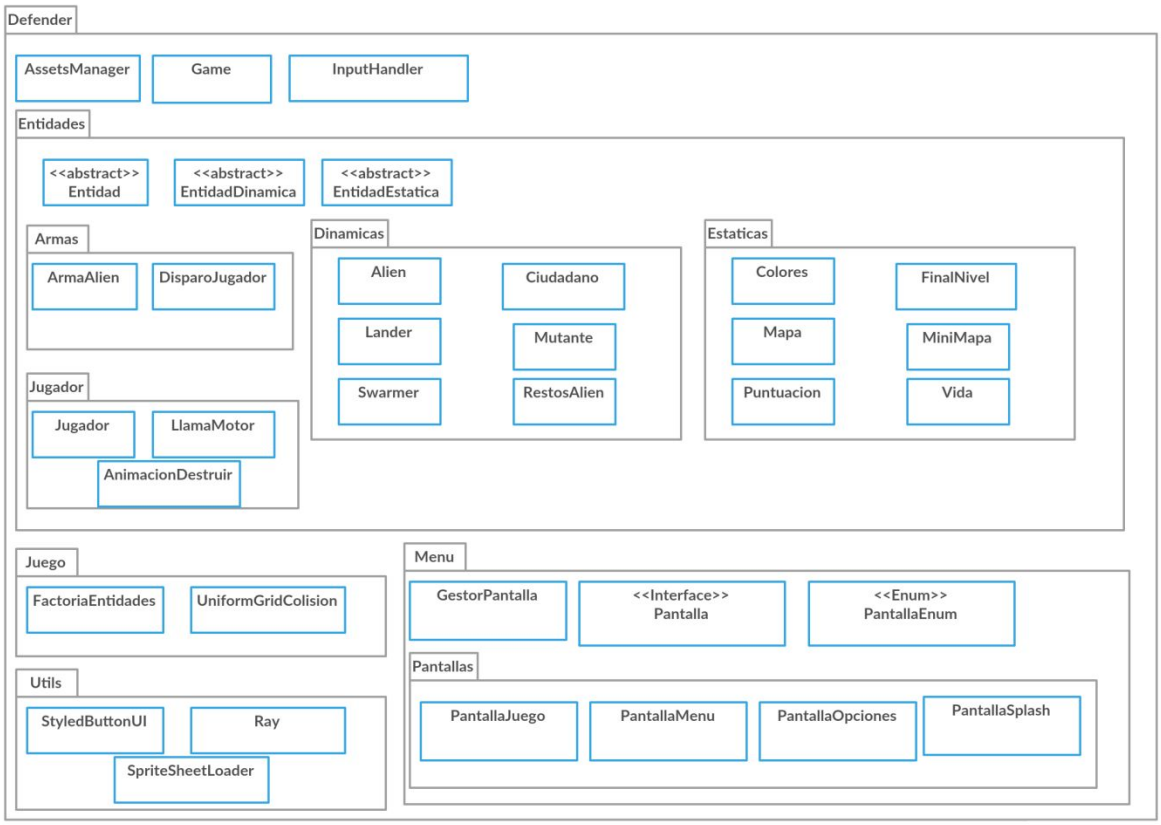

Figura 26. Diagrama de paquetes proyecto

14.2 Planificación

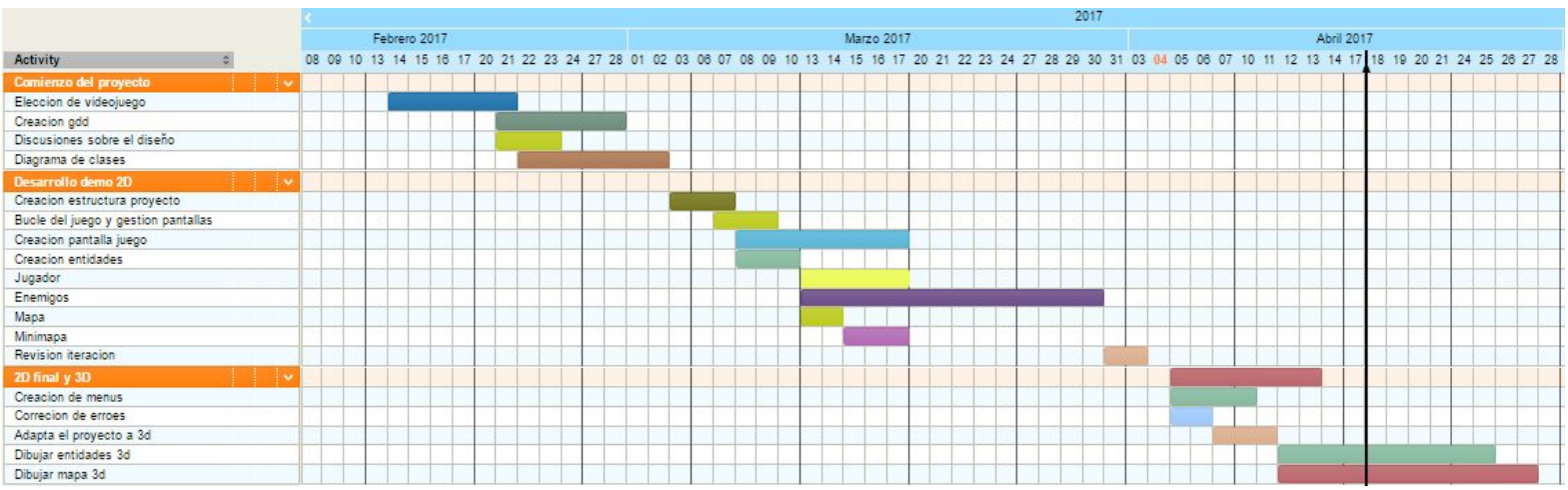

Figura 27. Planificación del proyecto

16 Referencias bibliográficas

[1] Defender - Wikipedia https://es.wikipedia.org/wiki/Defender_(videojuego) [02/03/2017]

[2] Defender (Red Label) 1980 Williams Mame Retro Arcade Games - Youtube [04/04/2017] https://www.youtube.com/watch?v=gss3lxeqCok&t=346s

[3] How to implement uniform grids - http://gamedev.stackexchange.com [03/04/2017] http://gamedev.stackexchange.com/questions/69310/how-to-implement-uniform-grids

[4] Basic game engine - http://codereview.stackexchange.com [03/04/2017] http://codereview.stackexchange.com/questions/122611/basic-game-engine-timer-fps-and-logic-loop

[4] 3d Game develope with LWJGL- Antonio Hernández Bejarano [01/05/2017] https://www.gitbook.com/book/lwjglgamedev/3d-game-development-with-lwjgl/details