Universitat Oberta de Catalunya
uoc.edu

Pirate King

Mikel Ayuso Rodríguez
Máster universitario de Diseño y Programación de Videojuegos

Helio Tejedor Navarro
Joan Arnedo Moreno

07 de junio de 2020

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Pirate King*
Nombre del autor:	*Mikel Ayuso Rodríguez*
Nombre del consultor/a:	*Helio Tejedor Navarro*
Nombre del PRA:	*Joan Arnedo Moreno*
Fecha de entrega:	06/2020
Titulación:	*Máster universitario de Diseño y Programación de Videojuegos*
Área del Trabajo Final:	*Trabajo Final de Máster*
Idioma del trabajo:	*Castellano*
Palabras clave	*Puzle, Estrategia, 3D*

Resumen del Trabajo:

Este documento explica el proceso de creación y la versión final del juego *Pirate King,* un juego 3D con turnos simultáneos que combina la estrategia y calma de un juego tablero con elementos *roguelike*.

Pirate King ha sido diseñado y desarrollado desde la idea inicial hasta el producto final durante el transcurso del proyecto, para lograrlo se han utilizado distintas herramientas como el motor de juegos Unity 3D, *Krita*, *Maya y Audacity*.

El producto final es una versión completa del juego, un modo aventura que se basa en partidas cortas con dos personajes jugables, seis tipos de enemigos, cada uno con su propia inteligencia artificial, cinco niveles generados de forma procedural con dos fases cada uno y nueve mejoras seleccionables por el jugador para desarrollar el barco durante cada partida.

Aunque ciertas ideas se hayan quedado sin implementar y estén como posibles ampliaciones en un trabajo futuro, el resultado es un juego completo, relajante y agradable, un juego que no mete prisa al jugador, pero tampoco le deja las cosas fáciles.

Abstract:

This document explains the creation process and final version of *Pirate King*, a 3D simultaneous turn-based 3D game that combines the strategy and calmness of a board game with roguelike elements.

Pirate King has been designed and developed from the initial idea all the way to the final product during the course of this project, different tools such as the Unity 3D engine, Krita, Maya and Audacity have been used in order to achieve this.

The final product is a complete version of the game, an adventure mode that is based on short games with two playable characters, six types of enemies, each one with its own artificial intelligence, five procedurally generated levels with two different phases each and nine enhancements for the player to choose from between levels to improve the ship during each game.

Although some ideas have not been developed and are now part of the possible future work, the final result is a complete game, relaxing and pleasant, a game that does not put the player in a rush, but that also does not allow the player to win without a fight.

Índice

Lista de figuras

1 Introducción

1.1 Contexto y justificación del Trabajo

El videojuego es un tipo de arte muy especial, ya que, a diferencia de muchos otros tipos de expresiones artísticas, es interactivo.

No es la única, por supuesto, existen obras musicales e incluso cuadros y esculturas que cambian según cómo y desde donde se miren, pero en un videojuego el jugador puede ser parte de la obra, meterse dentro, y lo mismo sucede con otro tipo de juegos como los juegos de mesa.

Pirate King es el intento de combinar una experiencia de juego de mesa relajada con la sensación de progreso, mejora y riesgo-recompensa del género *roguelike*.

En una industria que tiende a juegos cada vez más grandes, más espectaculares y más rápidos, este intenta hacer todo lo contrario, *Pirate King* pretende ser un juego relajante, crear una experiencia similar a una partida de ajedrez, partidas cortas pero con mecánicas de combate adaptadas a una sola pieza jugable (el barco pirata que controla el jugador), turnos simultáneos y progreso en forma de mejoras en el barco del jugador durante la partida, es decir, elementos del género *roguelike*.

1.2 Objetivos del Trabajo

El objetivo principal del proyecto es:

- Desarrollar un videojuego 3D multiplataforma (Escritorio, web y dispositivos móviles) desde cero hasta tener un producto terminado, jugable y pulido.

A título personal, hay otros objetivos secundarios que quería perseguir al empezar el proyecto:

- Mejorar en el uso del motor Unity 3D
- Mejorar como programador C#
- Gestión de un proyecto de tamaño considerable
- Diseñar y desarrollar elementos procedurales
- Diseñar y desarrollar inteligencias artificiales
- Explorar la idea de un juego con turnos simultáneos

1.3 Enfoque y método seguido

Aunque es cierto que el juego utiliza ideas o conceptos inspirados en otros títulos, el desarrollo se ha realizado desde cero, es un producto totalmente nuevo.

Como forma de trabajo se ha utilizado una variante personalizada de la metodología ágil *SCRUM*, que, si bien esta está diseñada para el trabajo en equipo, tiene ciertos elementos, como el trabajo por *features*, que son muy útiles para este tipo de proyectos cortos en los que se requiere mucha flexibilidad y cambios de alcance o rediseños rápidos.

Durante la planificación se dividió el trabajo en 3 etapas, y cada una de ellas tenía como objetivo una versión con ciertas funcionalidades implementadas (Más detalles en la sección Planificación del Trabajo):

- Versión Parcial
- Versión Jugable
- Versión Final

El trabajo a realizar para cada una de estas versiones se dividió en *features* concretas abarcables en rangos de tiempo de días o, como mucho, pocas semanas, para un total de 10 funcionalidades, al estar todas implementadas, el juego se podría dar por finalizado, o al menos como *feature complete*, a falta de corregir los últimos *bugs*, si los hubiera.

Así mismo, cada *bug* encontrado durante el desarrollo, nueva funcionalidad necesaria pero no planificada inicialmente o el *feedback* de los usuarios se ha tratado de la misma forma.

Para organizar todas las *features* a implementar se ha utilizado el sistema de *issues* integrado en el propio repositorio *GitHub*, asignándola al hito al que pertenecía (Versión Parcial, Jugable o Final) y etiquetándola según el origen de la tarea (Por ejemplo, *core* para aquellas que estaban planeadas desde el principio, *enhancement* para aquellas que surgieron durante el desarrollo o *bug* para los fallos encontrados).

1.4 Planificación del Trabajo

A continuación, se muestra la planificación inicial del proyecto y el resultado de cada tarea:

Tarea	Inicio	Final	Estado
Diseño			
Diseño	2020/02/19	2020/03/01	Completada
Versión Parcial			
Estructura general y sistema de turnos	2020/03/02	2020/03/08	Completada
Tablero, jugador y movimiento	2020/03/02	2020/03/22	Completada
Enemigos y resolución de turnos	2020/03/23	2020/03/29	Completada
Nivel 1	2020/03/30	2020/04/05	Completada
Versión Jugable			
Generación procedural del tablero	2020/04/06	2020/04/12	Completada
Nivel 2	2020/04/13	2020/04/19	Modificada
Objetos / Mejoras	2020/04/20	2020/05/10	Completada
IA Enemigos	2020/04/13	2020/05/17	Completada
Opciones	2020/05/18	2020/05/24	Completada
Versión Final			
Bugs y ajustes de dificultad	2020/05/25	2020/06/07	Completada

La tarea de crear el segundo nivel en la versión jugable fue la única que varió de alguna forma y no fue implementada según estaba diseñada en un primer momento. Al ser un juego *roguelike* con niveles generados aleatoriamente y con un barco que empieza cada partida sin ninguna mejora, lo ideal es que el jugador pueda avanzar por distintos niveles de forma rápida para poder mejorar el barco y tener una sensación de progreso.

Pirate King es un proyecto con un alcance muy limitado, tanto en recursos humanos como en tiempo, por lo que se decidió cambiar esta tarea de la realización de un segundo nivel más complejo que el primero por una serie de 5 niveles con dos fases cada uno, aprovechando el sistema de generación de niveles procedurales implementado al comienzo del desarrollo de esa misma versión jugable.

1.5 Breve sumario de productos obtenidos

El producto final obtenido es un juego 3D por turnos multiplataforma disponible como descargable para PC, a través de un navegador web alojado en un servidor y en dispositivos móviles, el proyecto se compone de los siguientes elementos:

- 5 niveles con 2 fases cada uno, todo ello generado de forma procedural
- 2 personajes jugables
- 3 tipos de enemigos, cada uno con 2 variantes (normal y mejorada)
    - Inteligencia artificial que logra un comportamiento único para cada tipo de enemigo
- Interfaz de usuario completa con temática pirata
- Opciones de sonido y de pantalla editables
- Sistema de guardado y cargado de opciones y progreso
- Modo de juego *Versus* (1 contra 1), aunque no es accesible por el jugador (Más información en el apartado Modo Versus)

1.6 Breve descripción de los otros capítulos de la memoria

- 2) Estado del arte: En este capítulo se aborda el género *roguelike*, haciendo especial hincapié en los que títulos integran las mecánicas de ese género con sistemas por turnos.
- 3) Definición del juego: El capítulo detalla aspectos clave del juego como la ambientación, las mecánicas principales o los personajes jugables, sin entrar en análisis técnicos a bajo nivel.
- 4) Diseño técnico: Análisis técnico del producto final, desde las herramientas utilizadas, hasta el comportamiento de la IA enemiga o la arquitectura del sistema *software*.
- 5) Diseño de niveles: Este capítulo detalla el funcionamiento del sistema de niveles y la generación procedural del tablero de juego.
- 6) Interfaz de usuario: Información sobre la interfaz de usuario, menús, interfaz física, interfaz durante la partida y cómo el usuario introduce o recibe información del juego.
- 7) Conclusiones: Mi propio *feedback* sobre el trabajo realizado, el producto final y las posibles líneas de trabajo futuras para el proyecto.
- 8) Glosario: Definición de los términos y acrónimos más importantes utilizados en cada contexto concreto.

2 Estado del arte

Un juego del subgénero *roguelike* es un juego de rol que se caracteriza por tener niveles generados de forma procedural y muerte permanente, es decir, cuando el personaje del jugador muere, la partida vuelve a empezar desde el principio. El primer juego con estas características fue *Rogue* (1980), de ahí el nombre del género.

Durante la década pasada vimos un *boom* de este género en la escena *indie.* Tal vez porque es un tipo de juego que no requiere de grandes equipos técnicos, ya que se trata de jugar una y otra vez los mismos niveles con los mismos personajes, pero con ciertos elementos aleatorizados, que hacen que cada partida sea totalmente diferente a la anterior, aunque ocurran con el mismo personaje y en el mismo escenario.

Ilustración 1: The Binding of Isaac: Afterbirth+

El resurgir del género a la popularidad empezó con *The Binding of Isaac* (2011), y ha abarcado infinidad de ambientaciones como mundos postapocalípticos en *Nuclear Throne* (2015) o aventuras épicas en *Dead Cells* (2017).

Además de las características del género, todos ellos tienen mucho en común, el ritmo frenético de atacar y esquivar golpes y proyectiles hace que sean juegos que no te dan ningún respiro más allá de las transiciones entre niveles y las áreas de descanso.

Ilustración 2: Selección de mejoras en *Nuclear Throne*

Por otro lado, todos ellos tienen también dan la opción de ir mejorando al personaje durante la partida, por ejemplo, en *Nuclear Throne* se recoge experiencia de los enemigos y cada vez que el personaje sube un nivel, el jugador puede seleccionar una mejora de entre 4 posibilidades aleatorias, otros juegos tienen sistemas diferentes como tiendas y dinero para comprar mejoras o enemigos que directamente dejan caer mejoras para el personaje al ser derrotados.

Paralelamente a estos, y más notoriamente en la última mitad de la década hemos podido ver juegos que aplicaban las mecánicas de los *roguelikes* a sistemas basados en turnos, dos buenos ejemplos son *Slay the Spire* (2017) e *Into the Breach* (2018), siendo el primero un juego de cartas y el segundo un juego de tablero.

Ambos, entre otros tantos que siguieron la misma corriente, mantienen las mecánicas base de los *roguelike* (niveles generados dinámicamente y muerte permanente), pero a diferencia de los mencionados anteriormente (*The Binding of Isaac, Nuclear Throne,* etc.) estos se pueden jugar de forma mucho más relajada ya que al ser por turnos, no tienen límites de tiempo o si los tienen, son amplios y dejan tiempo para pensar sin tener que estar esquivando enemigos y/o sus proyectiles constantemente.

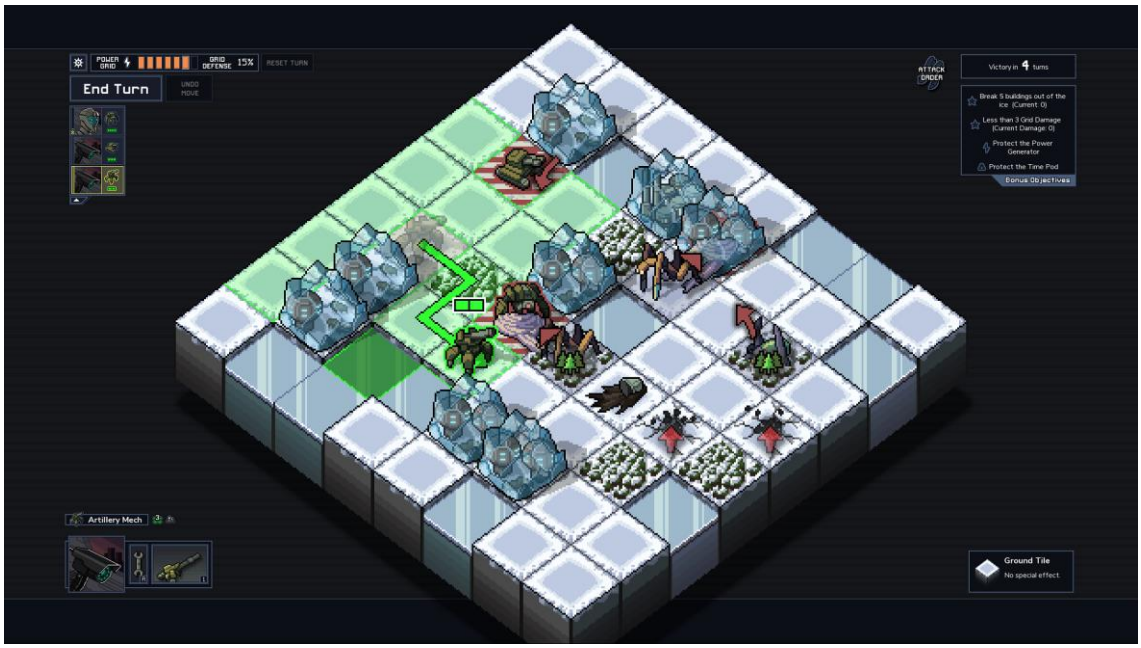

Ilustración 3: *Into the Breach*

No hay una tecnología única ni una plataforma única para este subgénero, se pueden encontrar *roguelikes* para cualquier plataforma y desarrollados con diferentes tecnologías.

Podemos fijarnos en *Archero* (2019), juego parecido a *Nuclear Throne* disponible tanto para Android como para iOS, mientras que todos los juegos mencionados hasta el momento son para PC y/o Linux.

En cuanto a la tecnología para desarrollarlos, Deck *Hunter* (2019) es un *roguelike* de cartas por turnos desarrollado en Unity, mientras que el ya mencionado *Slay the Spire*, también de cartas y por turnos está hecho en LibGDX (*framework* en Java para el desarrollo de videojuegos), y *The Binding of Isaac* y *Nuclear Throne* se hicieron con flash y *GameMake Studio* respectivamente.

(Solamente la primera versión de *The Binding of Isaac* se desarrolló en flash, la versión R*ebirth* y posteriores expansiones y ampliaciones se desarrollaron con un motor *C++* propio de los creadores)

3 Definición del Juego

Pirate King es juego de combate por turnos que integra mecánicas *roguelike* con la peculiaridad de que los turnos son simultáneos (Las acciones de todos los barcos se ejecutan a la vez, primero el movimiento y luego los disparos), el jugador toma el control de un barco pirata con el objetivo de hundir al resto y ser el último barco en pie.

Ilustración 4: Menú principal del juego

3.1 Ambientación e historia

La intención del juego es que el jugador pueda jugar partidas sueltas y tomarse su tiempo para pensar movimientos, se busca una ambientación distendida, casi fiestera y un tono muy relajado.

Todos los menús están diseñados para parecer pergaminos sobre una mesa de madera una taberna pirata con música de fondo incluida, y durante las partidas, aunque hay riesgo de que un enemigo hunda el barco del jugador casi todos los turnos, este puede tomarse todo el tiempo que quiera para decidir sus próximos movimientos.

El jugador encarna a uno de los piratas seleccionables en cada nueva aventura, desde la taberna (el menú principal) elige con qué pirata jugar (*Doug* o *Lenina*), y al terminar la aventura, ya sea porque ha conseguido el objetivo o por que ha perdido, vuelve a la taberna. En ningún momento hay puzles que requieran de rapidez o reacciones rápidas.

3.2 Mecánicas jugables

Al principio el jugador tenía 3 huecos de movimiento (más dos de disparo por cada movimiento) y 4 posibles movimientos:

Avanzar, avanzar y girar a la izquierda, avanzar y girar a la derecha y quedarse quieto.

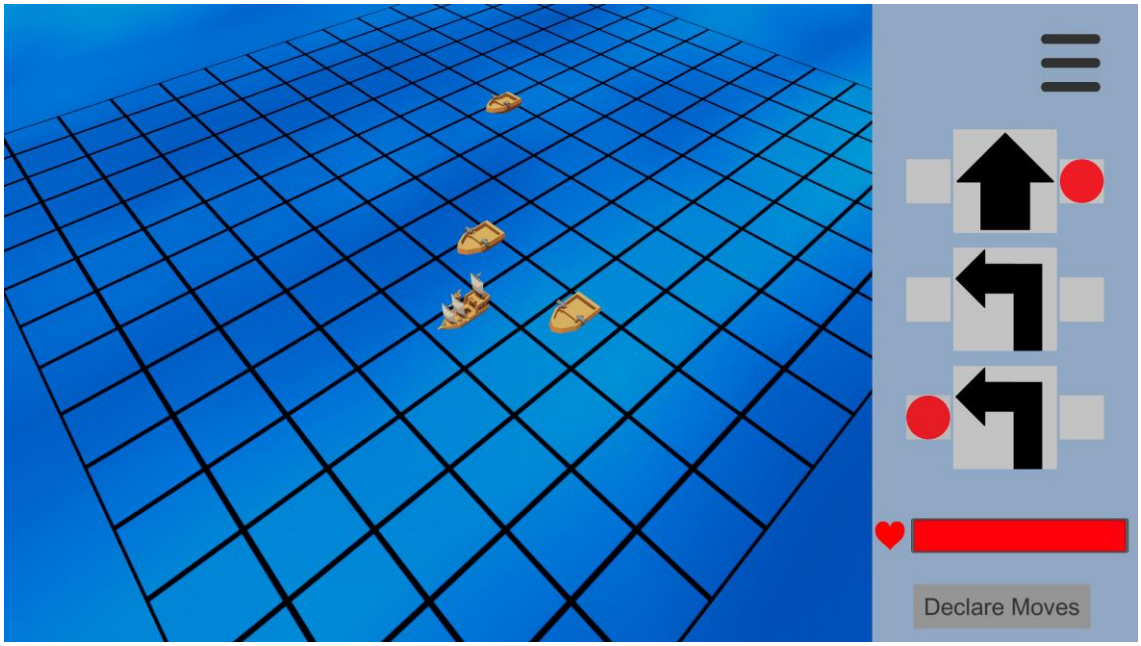

Ilustración 5: Primer prototipo que permitía movimiento del jugador

Cada turno el jugador elegía 3 movimientos (Ejecutados de arriba hacia abajo) y disparar o no después de cada uno.

Por ejemplo, en el caso de la Ilustración 5, el jugador se movería hacia adelante, dispararía hacia la derecha, después giraría dos veces hacia la izquierda y finalmente dispararía hacia la izquierda.

Esta mecánica se ha mantenido, la idea base de cómo funcionan los disparos y movimientos es la misma, pero ciertos detalles han cambiado desde ese primer prototipo a la versión final del juego.

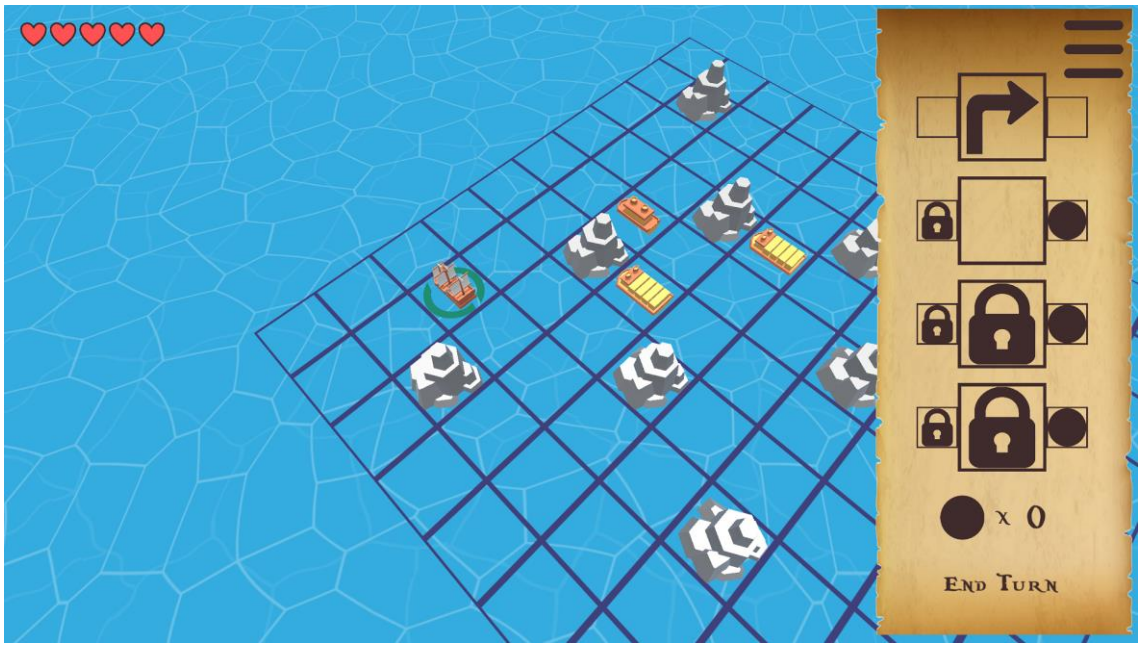

Ilustración 6: Introducción de movimientos en la versión final

En el ejemplo de la Ilustración 6, el jugador avanzaría y giraría hacia la derecha, después permanecería en la misma posición, y finalmente haría tres disparos seguidos a la derecha.

Durante el desarrollo se vio que utilizar como máximo 3 movimientos por turno era una limitación demasiado grande, por lo que se añadió un movimiento extra, y se limitó la cantidad de movimientos iniciales de cada personaje, añadiendo una mejora que, al recogerla, desbloquea las casillas de movimiento, dando opción a añadir más movimientos por turno.

Además, esta última versión tiene mejoras como la visualización de acciones bloqueadas, entre otras, la información sobre estos cambios se puede encontrar en la sección Input y output durante la partida.

El juego final además añade dos tipos de movimiento extra, girar a la derecha y girar a la izquierda, es decir, rotar sin moverse de celda.

La única forma de dañar o recibir daños en el barco es disparando los cañones, excepto para el caso de los enemigos de tipo Kamikaze y *Charger,* estos hacen daño por contacto, aunque no pueden disparar.

3.3 Personajes jugables

Los personajes jugables son uno de los pocos apartados que no han sufrido cambios desde la idea inicial hasta el final del proyecto, se han hecho mejoras al menú de selección de personaje, y se ha limitado los movimientos por turno que pueden hacer, pero la idea inicial de cada personaje se ha mantenido intacta.

- *Doug*

El barco de *Doug* puede realizar movimientos ágiles: avanzar, avanzar y girar a la izquierda, avanzar y girar a la derecha y quedarse quieto. Al comienzo de cada partida puede realizar 2 movimientos por turno, y puede desbloquear las otras dos casillas de movimiento recogiendo mejoras.

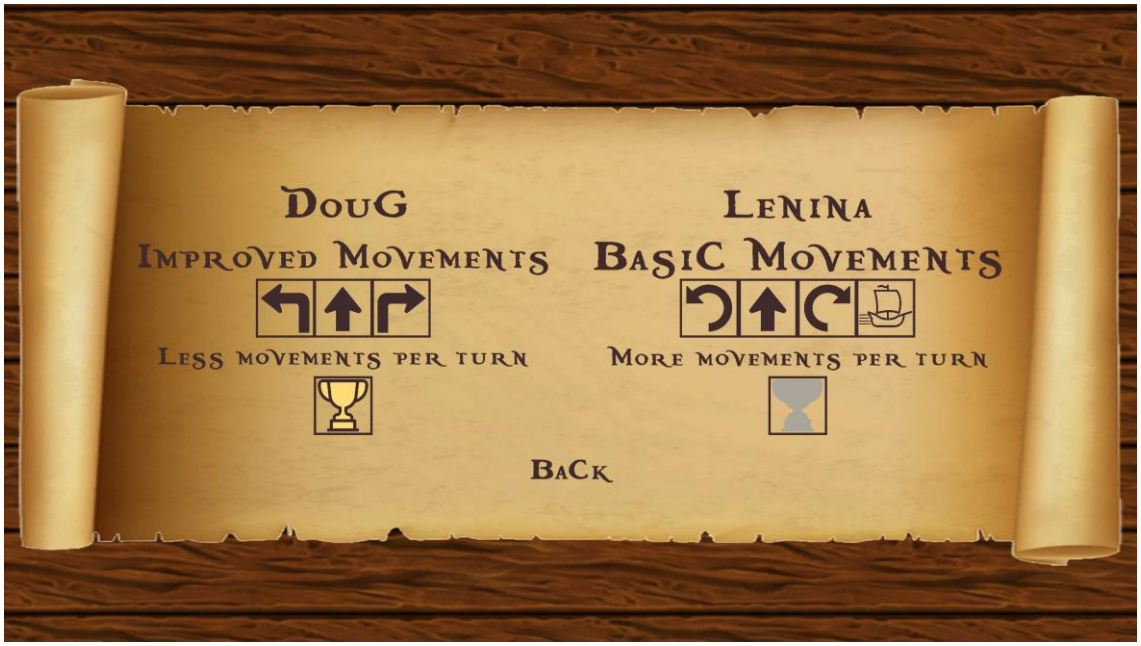

Ilustración 7: Pantalla de selección de personaje

- *Lenina*:

Lenina en cambio tiene un barco con movimientos más “lentos”. Puede avanzar en línea recta de la misma manera, pero solamente puede girar en la celda, es decir, girar a la derecha o a la izquierda sin moverse del sitio, y al igual que *Doug*, también puede optar por no realizar ningún movimiento. A diferencia del primero, si el jugador escoge el barco de *Lenina*, podrá realizar 3 movimientos por turno en un principio, hasta que escoja la mejora de desbloquear el cuarto movimiento extra. El número máximo de movimientos por turno para ambos es de 4.

3.4 Objetivos

El objetivo de cada partida es el mismo, derrotar (hundir) a todos los barcos enemigos. Al acabar con todos los enemigos en cada una de las 2 fases de los 5 niveles, el jugador consigue un trofeo visible en el menú de selección de personaje para el personaje con el que haya completado la hazaña.

4 Diseño técnico

4.1 Entorno de desarrollo

Durante todo el desarrollo se ha utilizado la versión 2019.3.0f6 del motor de videojuegos 3D Unity.

La elección de herramienta principal para el desarrollo se hizo entre *Godot*, *LibGDX* y *Unity*. Los motivos principales son que los tres son gratuitos (Unity hasta cierto punto, para un proyecto de esta escala lo podemos considerar totalmente gratuito) y son herramientas que ya he utilizado anteriormente.

LibGDX quedó descartada por que solamente había trabajado en proyectos en 2 dimensiones con ese *framework* y el tiempo de desarrollo era muy limitado, no quería empezar con una herramienta que no sabía si iba a servir. La decisión entre *Godot* y *Unity* fue más complicada, las dos herramientas podrían servir perfectamente para este proyecto, y *Godot* es *open source*, lo cual es siempre un punto a favor, aunque finalmente opté por Unity por comodidad, los últimos años he trabajado más con *Unity* que con *Godot*, y teniendo tan poco tiempo para crear un producto final, ese tiempo que podía ahorrar utilizando la herramienta que más “fresca” tenía podía venir muy bien.

4.2 Otras herramientas empleadas

A parte del motor *Unity*, se han utilizado otras herramientas para las distintas necesidades del proyecto:

- *Visual Studio Code*: IDE (*Integrated Development Environment)* para escribir el código C#.
- *Krita*: Software de dibujo y edición 2D para los *sprites*.
- *Maya*: Software de modelado para los modelos 3D.
- *Audacity*: Software de audio para la edición de música y efectos de sonido.
- *Git*: Sistema de control de versiones para llevar un histórico del proyecto y una trazabilidad de versiones y cambios.

Se ha intentado priorizar el uso de herramientas de software libre, excepto en el caso del modelado 3D, me hubiera encantado utilizar *Blender*, pero de nuevo, la limitación de tiempo era demasiado grande y nunca lo he utilizado, por lo que opté por usar *Maya*, que lo conocía mejor gracias a asignaturas anteriores y contaba con la licencia necesaria.

4.3 *Assets* y recursos utilizados

Se ha utilizado una combinación de *assets* creados desde cero para este proyecto, *assets* comprados y *assets* gratuitos de distintas fuentes:

- Modelos 3D

Los modelos 3D con una combinación de *assets* de *Kenney*:

  - Barco y rocas del Pirate Kit
  - Barcos enemigos del Watercraft Pack

- Música y efectos de sonido

Todos los efectos de sonido y música del juego son de un *Music & SoundFX bundle* de noviembre de 2019 de HumbleBundle, por desgracia ese directorio ya no está disponible para listarlo.

- Sprites e iconos 2D

Todos los dibujos, iconos e imágenes en 2D con creadas desde cero a excepción de los pergaminos, que son de Freepik.

Se utilizó ese pergamino como base para todos los demás, utilizando Krita se crearon también todos los iconos con la idea de que parecieran estar dibujados sobre los pergaminos.

- Colores

La paleta de colores utilizada para todos los elementos del juego es la siguiente: Spanish Palette

Todos los modelos, imágenes y efectos de sonido fueron editados con *Maya*, *Krita* y *Audacity* respectivamente para encajar con el resto de componentes del juego.

4.4 Arquitectura del proyecto

4.4.1 Organización del proyecto

La organización del proyecto intenta basarse lo máximo posible en *prefabs*, añadiendo todo lo que necesita cada uno de ellos en su propio directorio. El sistema de ficheros dentro del directorio *Assets* del proyecto está organizada de la siguiente forma (este listado solamente incluye los elementos más importantes, hay otros muchos ficheros y directorios que no se listan aquí para no alargar innecesariamente la lista, ya que la idea es que se entienda la estructura general de organización):

- *Images*
- *Materials*
- *Prefabs*
  - *Board*
  - *Camera*
  - *Common*
  - *CommonScripts*
  - *CommonSounds*
  - *Enemies*
  - *PauseScreen*
  - *Player*
  - *Projectile*
- *Scenes*
- *Scripts*
- *Sounds*

Cada directorio dentro de *prefabs* contiene los scripts, imágenes, sonidos, etc. Necesarios y únicos de ese *prefab* en concreto, los directorios *Common, CommonScripts* y *CommonSounds* contienen recursos utilizados en varios prefabs, un ejemplo son los sonidos de movimiento o de disparo de los barcos, que se utilizan tanto para enemigos como para el jugador, y por último las carpetas externas como *Images, Sounds, Scripts* y *Materials* contienen *Assets* que se utilizan de forma puntual sin ser parte de ningún *prefab*, por ejemplo, la música de menú principal del juego.

4.4.2 Navegación del usuario entre menús

La navegación entre menús toma como base el menú principal. Al iniciar el juego se abre este menú por primera vez y el jugador escoge qué quiere hacer.

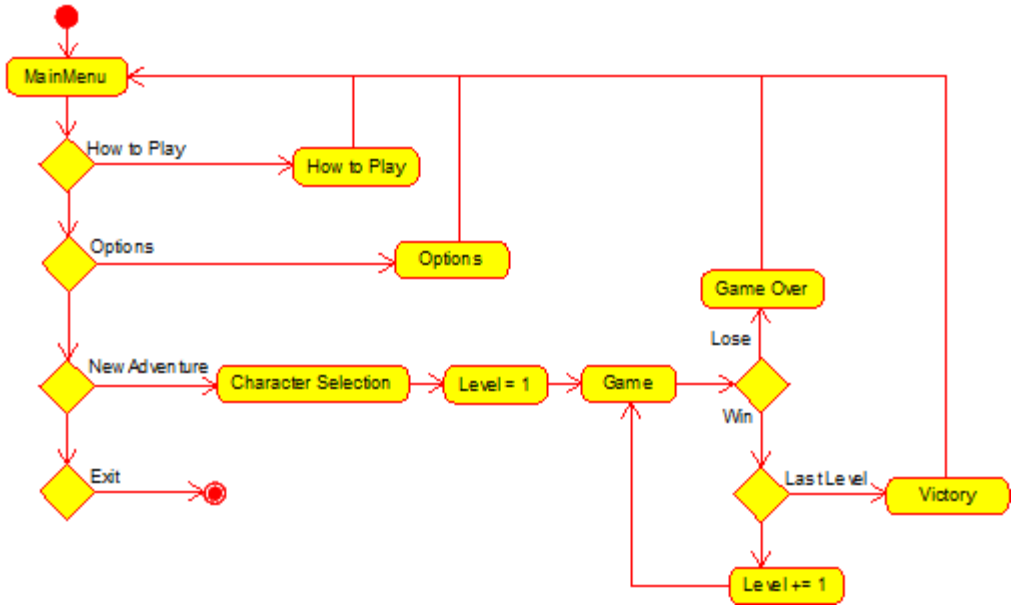

Ilustración 8: Diagrama de flujo entre menús

Como se puede ver en el diagrama de flujo de la Ilustración 8, todo el sistema de menús está pensado para que cada acción que haga el jugador acabe siempre llevándolo de vuelta al punto de partida, el menú principal.

El caso más completo es, obviamente, el de jugar una partida, si el jugador selecciona la opción *New Adventure*, accede al menú de selección de personaje, después a la propia partida, y va avanzando niveles hasta superar el último y ver la pantalla de *Victory* o quedar eliminado y ver la pantalla de *GameOver*, en cualquiera de los dos casos el último paso siempre será volver al menú principal.

4.4.3 Arquitectura de clases

La arquitectura de clases es compleja, pero el núcleo lo forman las clases *Board*, *BaseController* y GameManager.

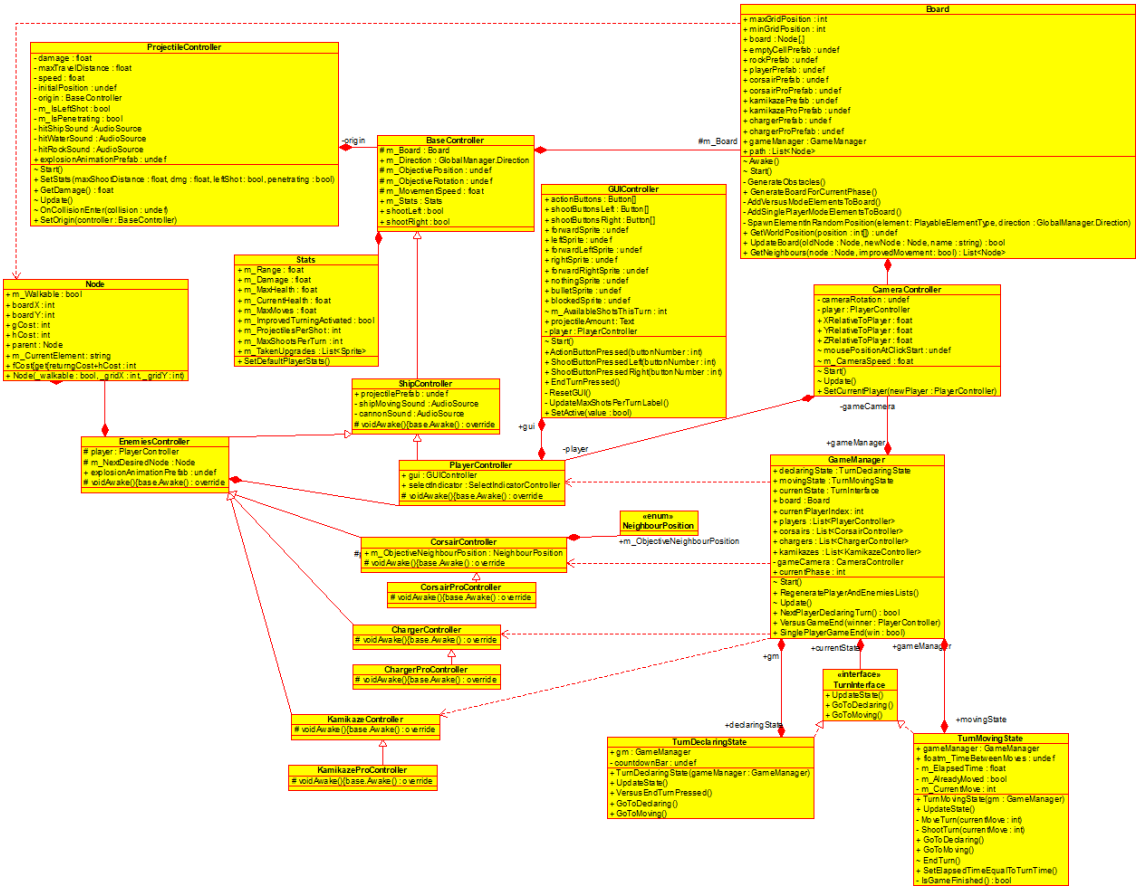

Ilustración 9: Diagrama de las principales clases

La Ilustración 9 muestra solamente las clases principales, hay muchos otros scripts más que funcionan de forma más o menos individual, pero no son relevantes para entender la estructura general. El funcionamiento básico se basa en las 3 clases ya mencionadas:

- *Board*: Durante un nivel solamente se instancia un objeto de este tipo, ese objeto contiene una lista de *Nodos* (celdas del tablero) y contiene la información del estado del tablero, el tamaño, qué celdas están libres y cuales están ocupadas por enemigos, por el jugador, por rocas, etc.

  Esta información se almacena en un único lugar para que todos los barcos puedan saber dónde está el resto, por ejemplo, cuando un enemigo quiere saber dónde está el jugador, accede al objeto *Board* y lo consulta en la lista de celdas.

- *BaseController*: La clase base de todos los elementos dinámicos del tablero (barcos enemigos y barco del jugador). Esta clase contiene la lógica común de todos los elementos como por ejemplo la lógica para moverse por el tablero y otras funciones virtuales que se pueden sobrescribir para añadir o sustituir partes de código y lograr comportamientos puntualmente diferentes.

  Por ejemplo, si cogemos el caso de un *Charger* común: Este utiliza una clase llamada *ChargerController*, esta hereda de *EnemiesController,* la cual hereda los métodos y variables de *ShipController,* y esta a su vez hereda finalmente *de BaseController*.

  El comportamiento base de todos los barcos al colisionar en movimiento con otro barco es volver a una casilla válida, esto se hace en la función virtual *OnCollisionEnter(Collision collision)* de la clase *BaseController*, tanto el *Charger* común como el mejorado tienen el efecto añadido de hacer daño al jugador si colisionan con él, para lograrlo se hace un *override* de la función de colisión ya mencionada, se añade la lógica de hacer puntos de daño al barco del jugador y finalmente se llama a base.*OnCollisionEnter(collision)*, que ejecuta la lógica implementada en la función virtual de *BaseController*, haciendo que el barco vuelva a posicionarse en una celda vacía.

  El caso del jugador sería parecido, aunque hereda de *ShipController* directamente en lugar de *EnemiesController*, y se le añade un objeto tipo *GUIController*, ya que este necesita del *input* del jugador para hacer los movimientos en vez de los algoritmos de IA y búsqueda de caminos que contiene *EnemiesController.*

- *GameManager*: Esta clase, al igual que *Board* solamente se instancia una vez por nivel, este objeto es el encargado de controlar el sistema de turnos (Fase de declaración, fase de ejecución y ejecutar cada acción en orden), también tiene una lista de todos los enemigos y jugadores activos en partida y comprueba cuándo el jugador ha perdido (El barco del jugador se ha hundido) o ha ganado (No quedan enemigos).

4.5 Enemigos

Pirate King cuenta con 3 tipos básicos de enemigos, Corsarios, Kamikazes y *Chargers*.

Cada uno de estos tipos tiene la variación normal (de color amarillo) y la mejorada (de color rojo, más poderosa y difícil de combatir), para un total de 6 variantes de enemigos, todos ellos con estadísticas y comportamiento propio.

Los enemigos solamente aparecen en los márgenes superior, derecho e inferior del tablero de juego, mientras que el jugador aparece siempre en la parte izquierda al comienzo de cada nivel, con esto se consigue que el jugador nunca esté rodeado al comienzo de un nivel sin haber cometido ningún error. La posición exacta de aparición es aleatoria dentro de las filas y columnas exteriores del tablero.

Por otro lado, el tipo de enemigos que aparecerán es también aleatorio dentro de unos límites, las segundas fases de cada nivel (excepto el primero) tienen siempre al menos un corsario, mientras que en la primera fase nunca aparecen. La aparición de *Chargers* y Kamikazes es aleatoria, en una fase concreta se puede dar el caso de que todos los enemigos sean *Chargers* o todos sean Kamikazes, aunque lo más normal es que aparezca una cantidad combinada de ambos.

4.5.1 Corsario

El corsario es el enemigo más poderoso del juego, su barco es muy parecido al del jugador, tanto en aspecto como en movimientos y en la capacidad de disparar proyectiles.

Ilustración 10: Corsario Común (Amarillo)

El corsario común se puede encontrar en la segunda fase de los niveles 2 y 3 (en el primer nivel nunca aparecen corsarios) y tiene las siguientes estadísticas:

- Salud: 1 (equivale a un corazón del jugador)
- Rango de disparo: 2 celdas
- Movimientos por turno: 3
- Daño por proyectil: 1

Ilustración 11: Corsario Mejorado (Rojo)

El corsario mejorado es idéntico al común en cuanto a los movimientos que puede realizar, se puede encontrar en la segunda fase de los niveles 4 y 5 y tiene estadísticas mejores:

- Salud: 4
- Rango de disparo: 4 celdas
- Movimientos por turno: 4
- Daño por proyectil: 1

4.5.2 Charger

El *Charger* es un enemigo robusto que busca embestir al jugador, aunque ninguna de sus variantes tiene la capacidad de disparar proyectiles.

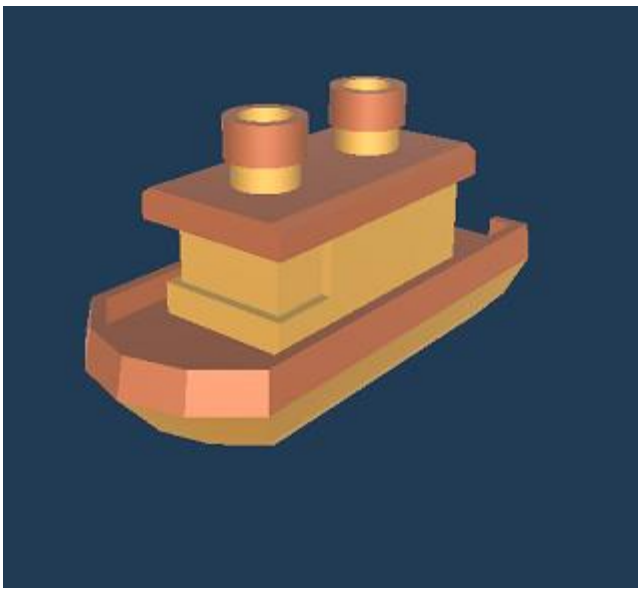

Ilustración 12: *Charger* Común (Amarillo)

El *Charger* común solo puede realizar los movimientos de *Lenina*, es decir, solo puede girar rotando sin moverse de celda, no puede avanzar y girar al mismo tiempo, aparecen en los tres primeros niveles y estas son sus estadísticas:

- Salud: 2
- Movimientos por turno: 3
- Daño por contacto: 1

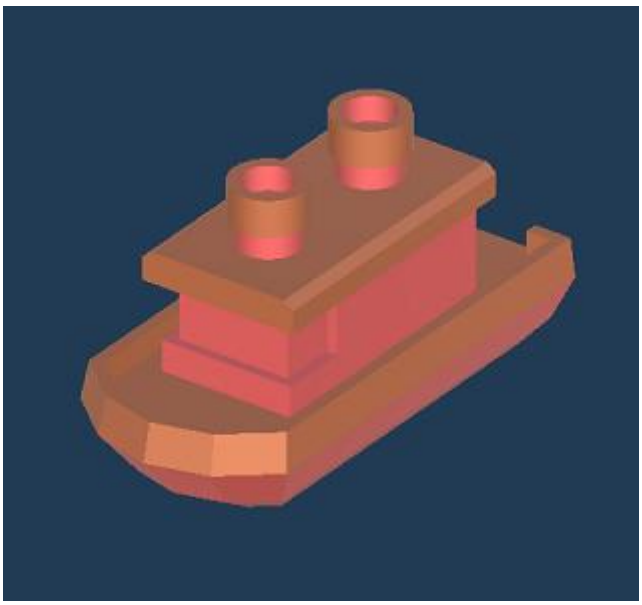

Ilustración 13: *Charger* Mejorado (Rojo)

Los *Charger* mejorados pueden realizar todos los movimientos disponibles, al igual que los Corsarios, aparecen en los niveles 4 y 5 y sus estadísticas son solamente ligeramente superiores a las de los *Charger* comunes:

- Salud: 3
- Movimientos por turno: 3
- Daño por contacto: 1

4.5.3 Kamikaze

Los Kamikazes son enemigos especialmente peligrosos, ya que, aunque no tienen mucha salud, causan una gran cantidad de daño en comparación al resto de enemigos al tocar el barco del jugador, aunque si llegan a hacerlo se inmolan y pierden toda la salud.

Ilustración 14: Kamikaze Común (Amarillo)

Los Kamikazes comunes tienen los mismos movimientos que los *Charger* comunes, los de *Lenina*, aparecen en los primeros 3 niveles y sus estadísticas son las siguientes:

- Salud: 1
- Movimientos por turno: 3
- Daño por contacto: 2

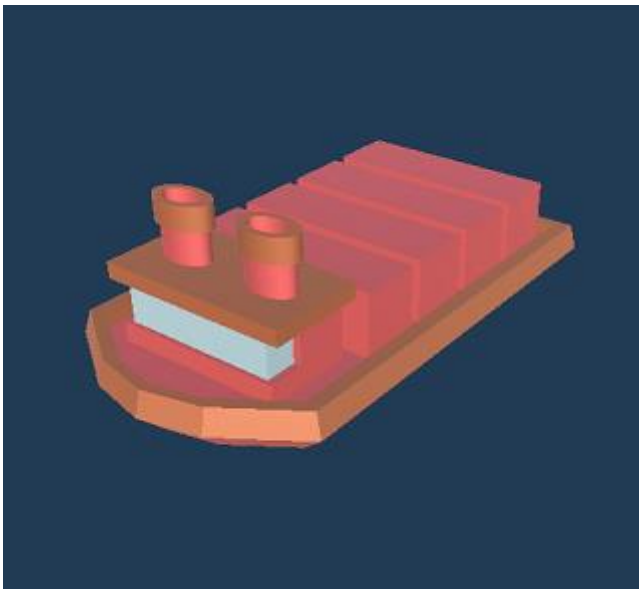

Ilustración 15: Kamikaze Mejorado (Rojo)

Los Kamikazes mejorados pueden utilizar todos los movimientos disponibles, de la misma forma que los *Charger* mejorados y las dos variaciones de Corsarios y solamente pueden aparecer en los niveles 4 y 5. Sus estadísticas son:

- Salud: 2
- Movimientos por turno: 3
- Daño por contacto: 2

4.6 Inteligencia Artificial

Para el movimiento de los enemigos se utiliza una implementación del algoritmo *A** (*A-Star*) ajustada al sistema de celdas y nodos del tablero.

Para encontrar la información que el algoritmo necesita para buscar un camino hace falta que los barcos enemigos ejecuten ciertos pasos previos, la ejecución del algoritmo cada turno se basa en estos tres pasos:

1) Buscar un objetivo

El primer paso es seleccionar la posición objetivo a la que el barco quiere llegar, los Kamikazes y los *Charger* tienen como objetivo chocar contra el jugador, por lo que su celda objetivo es siempre la posición del jugador.

Los corsarios en cambio quieren tener al jugador a tiro de sus cañones, por lo que su celda objetivo será una colindante a la del jugador, si es la celda inmediatamente delantera, cualquiera de las dos laterales o la trasera se decide al instanciar el Corsario y no varía durante la vida del mismo, es decir, un Corsario que haya elegido como objetivo inicial colocarse delante del jugador para dispararle intentará cada turno colocarse delante, mientras que

uno que haya elegido la parte trasera del jugador intentará llegar a estar detrás del jugador.

Esta información el jugador no la sabe, de esta forma se consigue que los Kamikazes y los *Chargers* sean relativamente predecibles (Enemigos más sencillos) mientras que el comportamiento de los Corsarios es más difícil de leer.

2) Buscar posibles celdas a ocupar

Los enemigos con movimiento mejorado evalúan todas las celdas a su alrededor (8 celdas) para encontrar a cuál de ellas moverse. Por otro lado, los enemigos con el movimiento básico evalúan solamente las celdas verticales y horizontales a su alrededor, es decir, la celda delantera, las laterales y la trasera respecto a su posición actual, sin tener en cuenta las diagonales.

Todos ellos evitan los obstáculos del tablero a la hora de buscar qué celdas son válidas como posible camino.

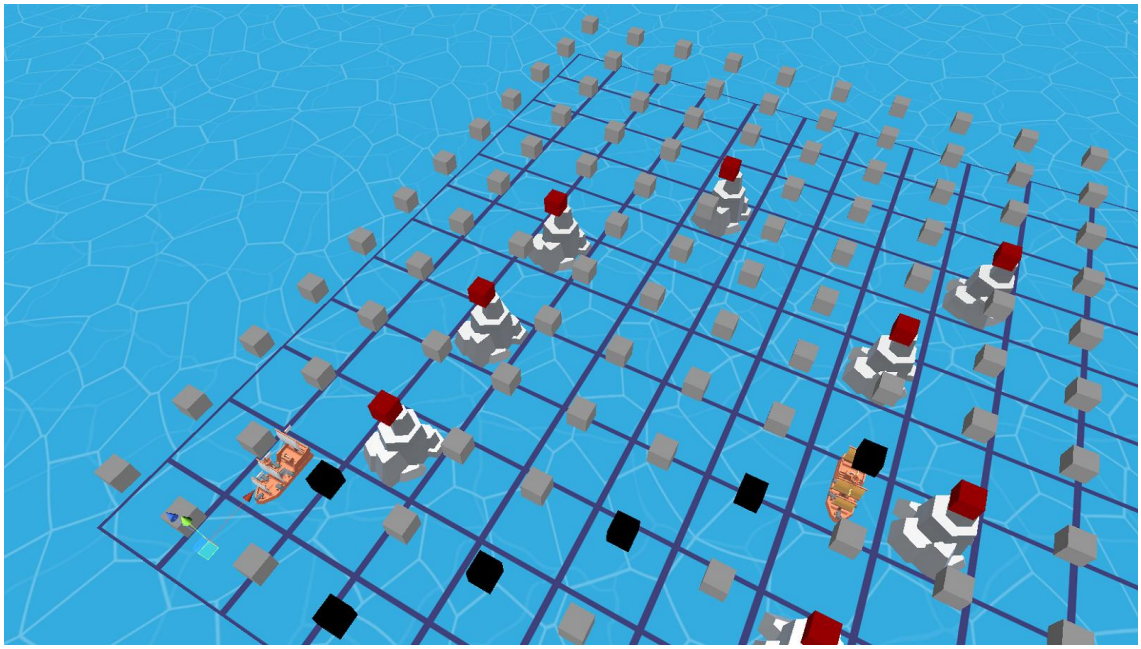

Ilustración 16: Corsario (Derecha) evaluando un posible camino

En la Ilustración 16 se puede ver una representación gráfica de las celdas que evalúa (Gris), descarta (Rojo) y finalmente selecciona (Negro) un enemigo de tipo corsario para colocarse en la celda inmediatamente a la izquierda de la posición del jugador.

3) Seleccionar un movimiento

Utilizando el objetivo y las posibles celdas a las que moverse como *input*, el algoritmo *A** se encarga de evaluar todas las opciones y encontrar el camino optimo, y, por tanto, el siguiente movimiento a realizar este turno.

*A** funciona por costes, asigna un coste a cada celda y elige moverse a la celda que menos coste suponga, en el caso de los enemigos con movimiento básico se varía un poco este algoritmo incrementando muy ligeramente el coste de un movimiento si no es avanzar en línea recta. El incremento es minúsculo y solamente sirve como “desempate”, si el algoritmo encuentra dos posibles movimientos igual de buenos, priorizará moverse hacia adelante en lugar de girar. Este cambio solo se aplica a los enemigos con el movimiento básico para evitar que giren cuando podrían seguir avanzando, los enemigos con movimiento mejorado no necesitan este ajuste ya que al girar ya están avanzando, por lo que no pierden tiempo girando.

4.7 Objetos / Mejoras

Al acabar con todos los enemigos de la segunda fase de cada nivel (excepto en el último) se abre el menú de selección de mejoras, en él aparecen 4 iconos, al hacer clic sobre cualquiera de ellos se aplica la mejora seleccionada al barco del jugador y se continúa al siguiente nivel.

Ilustración 17: Pantalla de selección de mejoras

Hay un total de 9 mejoras disponibles, siempre aparecen en pantalla 3 elegidas aleatoriamente a escoger y la última (la primera por la derecha) es siempre una cura / reparación del barco.

Algunas de estas mejoras se pueden coger varias veces, mientras que otras están limitadas a uno o pocos usos. La lista completa de mejoras es la siguiente:

- Mejoras que se pueden coger ilimitadamente
    - Aumentar vida máxima
    - Aumentar daño
    - Aumentar rango del disparo
    - Recuperar salud / Curar
    - Más disparos / proyectiles disponibles por turno

- Mejoras limitadas

    - Un movimiento extra

    La mejora que otorga un movimiento extra al barco está limitada a un total de 4 movimientos. En el caso de *Doug*, que empieza con la mitad de los movimientos disponibles y la otra mitad bloqueados, la mejora podría cogerse 2 veces, en el caso de *Lenina*, empieza con 3 por lo que solo podría recoger esta mejora una vez.

- Mejoras únicas

    Las mejoras únicas solamente pueden ser seleccionadas una vez.

    - Disparo doble (Cada disparo lanza dos proyectiles)
    - Poder disparar a los dos lados a la vez
    - Disparo penetrante (el proyectil sigue su trayectoria hasta alcanzar el rango máximo, impactar contra un barco enemigo no hace que desaparezca)

Las mejoras limitadas y únicas dejan de aparecer entre las opciones seleccionables del menú de selección cuando ya no están disponibles.

4.8 Opciones y sistema de guardado

El juego pone a disposición del jugador una serie de opciones de sonido y pantalla para que pueda personalizar su experiencia.

Ilustración 18: Menú de opciones

Tanto estas opciones como el progreso del juego se guardan en disco y se cargan cada vez que inicia el juego, los parámetros que se guardan son:

- *VSync: True/False* (*bool*)
- *TargetFramerate*: Valor numérico entero (*int*)
- *MusicVolume*: Valor numérico (*float*)
- *SoundFXVolume:* Valor numérico (*float*)
- *DougComplete: True/False* (*bool*)
- *Lenina Complete: True/False* (*bool*)

Estas opciones se cargan únicamente al inicio de la partida y se guardan cada vez que el usuario hace algún cambio a cualquiera de las opciones o completa el juego con uno de los personajes jugables.

En el ejemplo de la pantalla de selección de personaje de la Ilustración 7, se ha cargado un fichero guardado, en él ya se habían completado todos los niveles con *Doug*, pero no con *Lenina*, de ahí que aparezca el trofeo en el lado izquierdo, pero no en el derecho.

4.9 Modo *Versus*

Aunque este modo de juego no haya llegado a ser parte de la versión final como modo jugable, es una parte del proyecto destacable a mencionar.

En un comienzo el juego se planteó con dos modos, el modo aventura, del que se ha estado hablando en todo momento hasta ahora, y el modo *Versus*, un modo que coloca a un jugador frente al otro, en igualdad de condiciones y con un sistema de turnos, movimientos y mecánicas jugables igual que el modo aventura. El primer barco en reducir la vida de su rival a 0, gana.

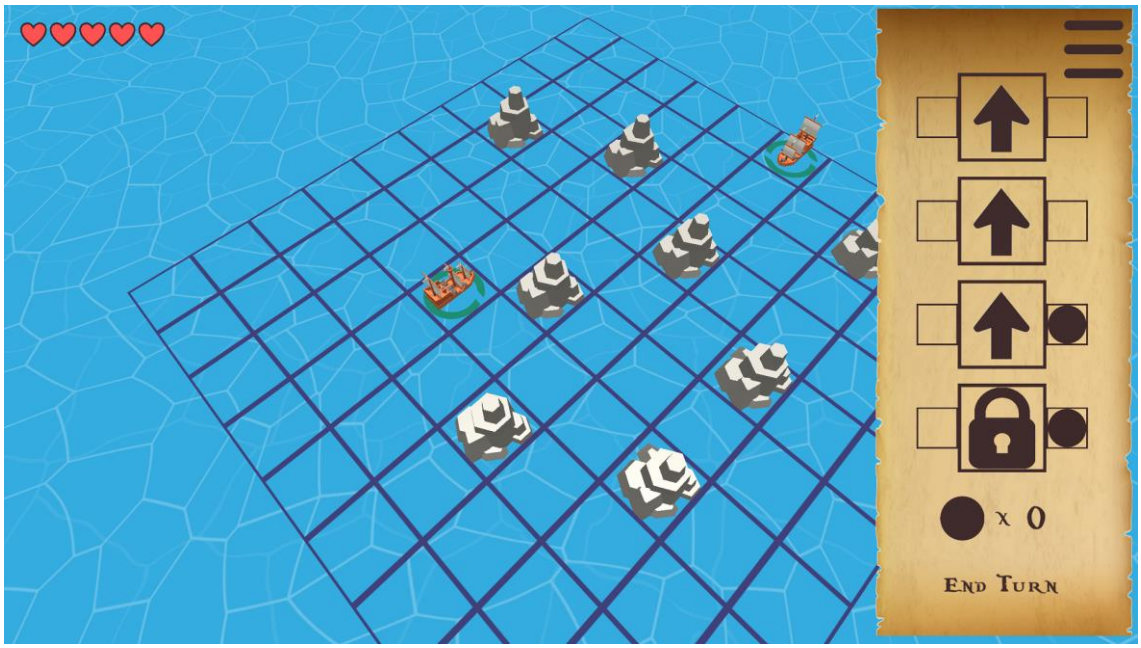

Ilustración 19: Modo *Versus* (1 contra 1)

Este modo de juego está implementado, es jugable y es parte del proyecto, pero en mitad del proyecto se decidió retirar la opción del menú principal para que no sea accesible para los jugadores ya que, al ser un modo *versus* local, lo que un jugador hacía podía verse en pantalla y el otro jugador podía contrarrestarlo muy fácilmente. Es un modo que tiene sentido para partidas en red local entre varios dispositivos o por internet, pero no como modo de juego de varios jugadores en el mismo dispositivo.

Aun así, el código y sistema está implementado y podría ser interesante cogerlo en un futuro para convertirlo en un modo de juego *online*.

5 Diseño de niveles

Los niveles en *Pirate King* se generan de forma procedural. No se han diseñado niveles como tal, sino que se ha desarrollado un sistema de generación de niveles para que se creen siendo lo más aleatorios posibles, pero siempre dentro de unos límites que garanticen que el nivel es jugable, superable y no va a ser una experiencia injusta y frustrante para el jugador.

5.1 Tablero de juego

El tablero de juego es distinto en cada fase. Al comienzo de cada nivel, en la primera fase, el tablero tiene un tamaño de 10 celdas por 10 celdas.

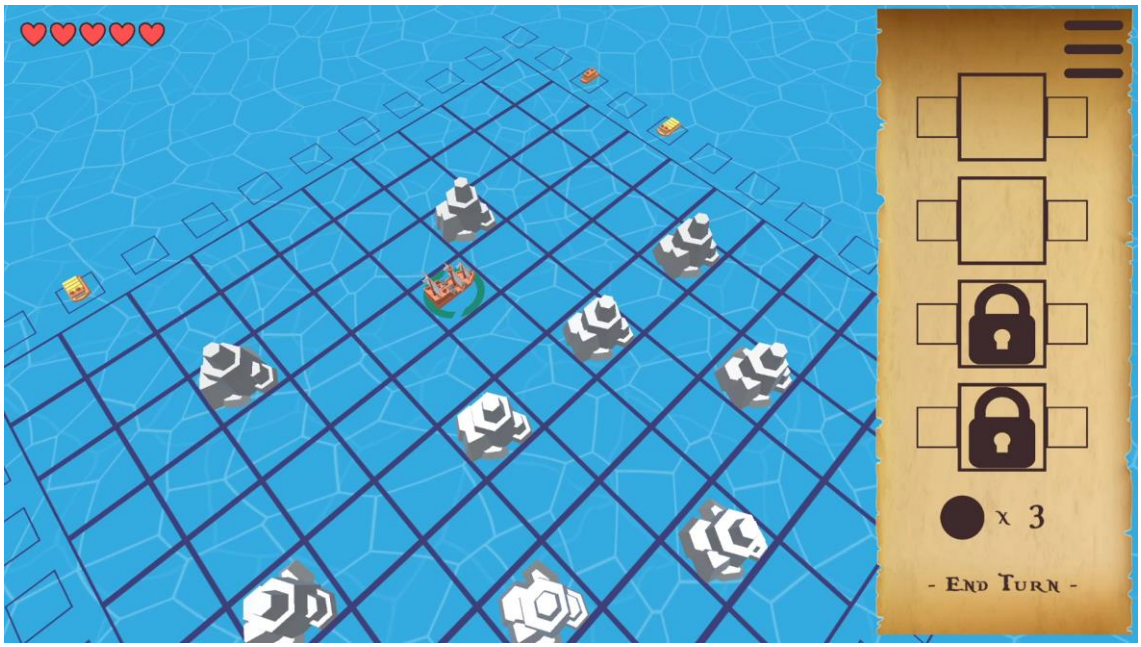

Ilustración 20: Tablero expandiéndose y generando enemigos para la fase 2

Al derrotar a todos los enemigos de la fase y avanzar a la siguiente, no se cambia de tablero, se expande el actual y se generan enemigos nuevos. Se añade una fila o columna de celdas en cada extremo del tablero para acabar siendo un tablero de 12 por 12.

Los límites del tablero funcionan como paredes invisibles, ningún barco puede salir y al chocarse contra ellos el comportamiento es el mismo que al chocar contra una roca. En la segunda fase, al expandir el tablero de juego, estos límites invisibles también se expanden para que coincidan con el límite de las nuevas celdas.

5.2 Obstáculos

Los obstáculos tienen forma de roca, son estáticos y se generan siempre al inicio de cada nivel. En la segunda fase no se generan más obstáculos.

La posición de estos es aleatoria, aunque el algoritmo de generación se asegura de que las rocas no estén muy juntas entre ellas para no generar pasillos sin salida o zonas inaccesibles. Tampoco se generan en los extremos del tablero, esa zona se reserva para generar enemigos.

Durante la partida los obstáculos juegan un papel importante, no se pueden atravesar, ni con el barco ni con proyectiles. El jugador puede utilizarlos a su favor para colocarse en posiciones favorables (Al no poder atravesarse, son buena protección), aunque también pueden jugarle una mala pasada si no domina la situación y puede quedar atrapado o entorpecido y que los enemigos embistan su barco.

5.3 Niveles

El juego consta de 5 niveles, el tablero y los obstáculos son generados de forma procedural al comienzo de cada nivel. Los enemigos también son generados de la misma forma al comienzo de cada fase.

Cada nivel se compone de dos fases, y al superar la segunda fase del último nivel, el juego se da por finalizado y se otorga el correspondiente trofeo para el personaje con el que se ha logrado completar.

6 Interfaz de usuario

6.1 Interfaz física

Para que el juego pueda ser jugado en cualquier plataforma, se decidió que toda acción debería poder ser realizada haciendo uso del ratón o clics en la pantalla en el caso de dispositivos móviles.

Tanto durante el *gameplay* como a la hora de navegar entre menús el ratón o los clics en pantalla son el único método de input que tiene el jugador.

Aunque no todo es bueno, y tener una interfaz física tan estricta, aunque es bueno para la compatibilidad entre plataformas, también es muy limitante, al no tener un mando de consola, por ejemplo, se pierde un canal muy valioso de información hacia el jugador, la vibración.

En el caso del ratón poco se puede hacer, pero en el caso de los dispositivos móviles se podría plantear el añadir una pequeña vibración cada vez que el barco del jugador colisiona o recibe daño de cualquier tipo. Es una idea interesante que podría aportar *feedback* físico al jugador, pero por el momento esta opción está descartada para que la experiencia en todas las plataformas sea idéntica y no está previsto añadir esta funcionalidad en el futuro.

6.2 Menús

Para crear la ilusión de que todos los menús son pergaminos sobre una única mesa, se han implementado todos los menús en la misma escena, a excepción del de selección de mejoras, que tiene su propia escena (Ilustración 17).

Ilustración 21: Vista de la "mesa de pergaminos" en el editor

Todos ellos, desde estos menús de navegación (Menú principal, menú de opciones, menú de cómo jugar y menú de selección de personaje), hasta los menús de introducción de ordenes al barco, menú de pausa y por supuesto el menú de selección de mejoras son pergaminos, y el jugador puede interactuar con ellos utilizando únicamente el ratón.

6.3 *Input* y *output* durante la partida

Como ya se ha explicado en el apartado Mecánicas jugables, los movimientos del barco dependen del panel de input en la parte derecha de la pantalla.

Hay 4 casillas de movimiento y 8 de disparo en total y dependiendo del icono que el usuario elija en cada una de las casillas el barco realizará una acción u otra. Al comienzo de cada turno todas las casillas aparecen vacías (Excepto las que por un motivo u otro están bloqueadas).

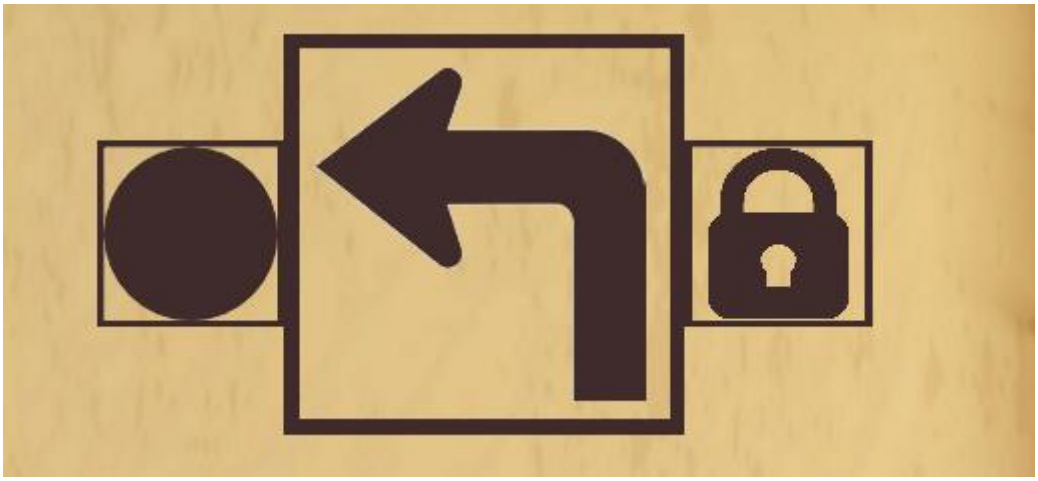

Ilustración 22: Introducción de movimientos

El jugador puede hacer clic en cada una de las casillas no bloqueadas para cambiar el icono que aparece, por ejemplo, en el caso de las casillas de disparo, hacer un clic hará que se muestre un círculo, eso indica que se va a realizar un disparo (En la Ilustración 22, por ejemplo, hacia el lado izquierdo del barco).

Las casillas de movimiento funcionan de la misma forma, cada clic cambia el icono, el primero pondría el icono de avanzar en línea recta, el segundo el de girar a la izquierda (o avanzar y girar, si se está jugando con *Doug*, como es el caso de la ilustración), el siguiente lo mismo, pero hacia la derecha y el cuarto y último volvería a dejar la casilla en blanco, haciendo que para esa fase del turno el barco permanezca inmóvil.

La Ilustración 23 muestra un ejemplo de la interfaz de usuario completa durante la fase de declaración de movimientos de un turno.

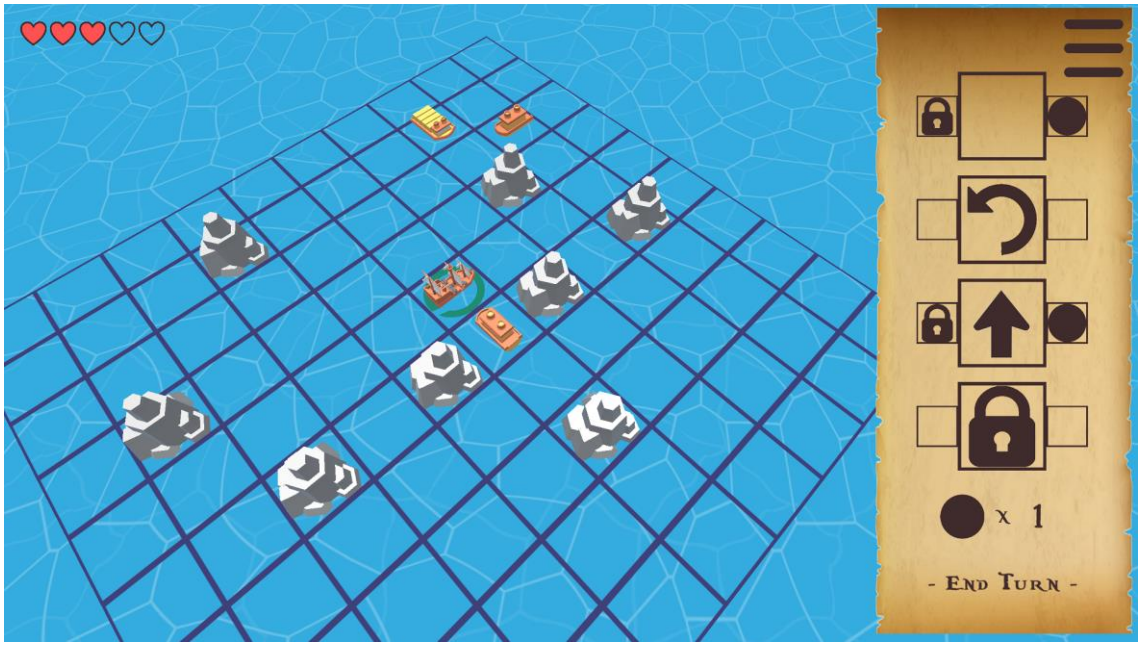

Ilustración 23: Interfaz de usuario durante una partida

En el lado superior izquierdo se ve la salud del barco, los corazones rojos indican la salud actual y los corazones vacíos la salud que se ha perdido, la suma de los dos es la salud total, en este caso 5.

La base del barco del jugador tiene un elemento espacial en forma de círculo verde giratorio que indica qué barco es el del jugador. Este elemento se introdujo gracias al *feedback* del tutor.

El panel derecho tiene varias funciones muy importantes, por un lado, como ya se ha explicado anteriormente, el *input de movimientos* y el botón de *End Turn* en la parte inferior, al pulsarlo comienza la fase de movimiento, ejecutando los movimientos seleccionados en las casillas superiores, en orden descendente.

En este caso, primero el barco se quedaría quieto, después dispararía hacia la derecha (El disparo hacia la izquierda está bloqueado por que el jugador aún no ha obtenido la mejora de disparar hacia los dos lados a la vez), después rotaría hacia la izquierda (Está jugando con *Lenina*, no puede avanzar y girar al mismo tiempo), no dispararía, avanzaría en línea recta y volvería a disparar a la derecha.

Entre las casillas de introducción de movimientos y el botón de finalizar turno el jugador también puede ver cuántos disparos tiene disponibles cada turno. Al comienzo de cada partida, sin haber recogido ninguna mejora, se pueden realizar hasta 3 disparos por turno. En este ejemplo, el jugador ya ha añadido dos disparos a la derecha en las casillas 1 y 3 (de nuevo, orden descendente), y le queda un disparo más, que puede elegir utilizar o no.

Durante la fase de movimiento (después de declarar movimientos, cuando los barcos ejecutan las acciones del turno), el panel derecho de *input* de movimientos se esconde para que el jugador pueda ver mejor qué está sucediendo en el tablero. Originalmente la salud del barco también se escondía, pero gracias al *feedback* de varios usuarios diciendo que, en ocasiones, no sabían cuánto daño habían recibido por qué motivos en concreto durante un turno, se decidió dejar la salud visible en todo momento y actualizarla en el mismo instante que se recibía el daño, sin esperar a la resolución del turno.

7 Conclusiones

7.1 Producto final

Estoy muy contento con el producto final, creo que es un juego entretenido, diferente y el tono relajado y ambiente tranquilo que pretendía conseguir creo que está ahí.

Los cambios que realicé durante el desarrollo (quitar límite de tiempo para declarar movimientos, quitar el modo *versus*, etc.) hacen que el juego sea más pequeño y más puro, y esto lo veo como algo positivo, tiene una idea, ser un juego de puzles sin ningún tipo de estrés y lo consigue.

Destacar también el *feedback* tanto del tutor, como de algunos de los compañeros de clase y finalmente de gente que jugó a *Pirate King* por internet sin conocerme ni a mí ni al juego. Todos estos comentarios fueron de gran ayuda para rematar detalles y lograr una versión final del juego mucho más robusta y atractiva.

Tampoco creo que sea perfecto, me hubiera gustado tener más tiempo para implementar más tipos de enemigos, más mejoras, más niveles, etc. Pero esto queda como trabajo futuro.

7.2 Trabajo realizado, planificación y metodología

Cuando hice la primera planificación en el primer entregable fui precavido, lo admito. Añadí pocas cosas, por que imaginaba que no iba a poder desarrollar mi juego ideal en el tiempo que duraba este proyecto, y creo que acerté.

Desde el principio la planificación y la ejecución han ido casi a la par, incluso tareas como la IA enemiga que pensaba que iban a llevar mucho más tiempo fueron mejor de lo esperado.

Aunque no todo es perfecto. Al principio planeé tener dos modos de juego (el modo aventura, incluido en el juego, y el modo *versus*, no incluido), todo el tiempo extra que gané haciendo algunas de las tareas más rápido de lo esperado creo que se me fue en este modo 1 contra 1. Incluso a estas alturas creo que no es mala idea añadir este modo como un modo online o en LAN, pero como *versus* local no funciona (los jugadores pueden ver la pantalla por lo que saben cómo se va a mover su rival), y no vi que no era viable hasta que ya estaba desarrollado e hice pruebas de juego contra otros jugadores.

Por un lado, me vino bien que tuviera ese pequeño tiempo extra para permitirme probar y descartar un modo de juego entero, por el otro, si hubiera sabido ver que no iba a funcionar como modo de juego, habría invertido más tiempo en desarrollar parte de las ideas que presento a continuación como trabajo futuro.

En cuanto a la metodología, creo que adaptar *SRUM* a un trabajo individual ha sido un acierto. No solo me ha ayudado a llevar una trazabilidad clara de tareas completadas, en curso y pendientes, si no que me ha dado la flexibilidad necesaria para poder adelantar o aplazar ligeramente la previsión de una tarea cuando era necesario.

7.3 Trabajo Futuro

Por la limitación de tiempo de este proyecto, tuve que resignarme y dejar anotadas sin poder planificar y desarrollar muchas ideas que, a la larga, podrían hacer de *Pirate King* un juego mucho más interesante y completo.

De haber tenido más tiempo, estas son las características principales que hubiera añadido a *Pirate King*:

- Pantalla de título:

  El juego no tiene una pantalla de título, nada más arrancar, abre directamente el menú principal. Una pantalla de título atractiva, además de añadir contenido de calidad al juego, ayudaría a crear ese ambiente fiestero y relajado de taberna que se pretende conseguir nada más entrar.

- Enemigos

  Los enemigos incluidos en la versión final del juego son interesantes, pero no son los únicos que estaban diseñados para ser parte de *Pirate King*, de haber tenido tiempo, el juego tendría dos tipos de enemigos más:

  - Enemigo estático (torre de vigilancia):

    Un enemigo que no puede moverse, pero tiene cañones y dispara proyectiles siguiendo un patrón: Atacar hacia delante y hacia atrás, después atacar hacia la izquierda y la derecha, después de nuevo hacia delante y hacia atrás, y así sucesivamente.

    Este enemigo añadiría la necesidad de que el jugador tuviera en cuenta en qué momento pasar por ciertas celdas o no, dependiendo de si la torre va a disparar en una dirección u otra.

  - Enemigo que huye del jugador

    Este enemigo sería otro barco, pero en vez de perseguir al jugador, huiría de él, dejando minas o barriles explosivos en ciertas celdas por las que pasa.

    Con este enemigo, el jugador tendría que, además de esquivar a Corsarios, Kamikazes y *Chargers*, perseguir a un nuevo enemigo con ciertas celdas bloqueadas por las trampas que deja al huir.

- Obstáculos bajos

  Todos los obstáculos que se añaden al tablero tienen la misma altura, y bloquean tanto a los barcos que chocan contra ellos como a los proyectiles.

  Este tipo de obstáculos bajos bloquearían a los barcos, pero los proyectiles podrían volar sobre ellos, por lo que podrían usarse como protección mientras disparas por encima de ellos a los enemigos.

- Modo de juego *versus online*

  Esta última puede que fuera más a largo plazo, pero un sistema *online* de *matchmaking,* de partidas 1 contra 1, con un sistema de puntuación tipo *ELO* (la puntuación que se utiliza para clasificar a los jugadores de ajedrez) sería un modo de juego muy interesante para un juego de estrategia como este.

  Probablemente habría que cambiar ciertas decisiones de diseño como añadir un límite de tiempo por turno exclusivamente en este modo de juego, por lo que la experiencia sería diferente a jugar en el modo aventura. Mientras en el modo para un solo jugador sería una experiencia como hacer un puzle de forma relajada y sin límite de tiempo, este modo de juego *online* con rango sería una experiencia de partida puramente estratégica y competitiva.

8 Glosario

- A* (A-Star): Algoritmo de búsqueda de caminos basado en nodos y coste del recorrido entre los nodos.

- *Asset*: Activo, recurso. En este contexto se llama *Asset* a todo elemento que forma parte del proyecto dentro del motor de juego (scripts, sonidos, modelos, dibujos, etc.)

- *Bug:* Fallo o error dentro de un proyecto *software* que hace que el producto no se comporte o no haga lo que está diseñado para hacer.

- *Features (SCRUM):* Aunque en español se pueda traducir como característica, dentro del desarrollo de software y especialmente dentro de la metodología *SCRUM*, *feature* hace referencia a una nueva funcionalidad.

- *Feature Complete*: Concepto del área de ingeniería del software, cuando un proyecto llega a una fase donde todas las funcionalidades están implementadas y solo queda buscar y solucionar errores, se dice que el producto está *feature complete.*

- *Issues (GitHub)*: Una *issue* es un problema, pero en el contexto de *GitHub* no siempre es así, en este caso sería más acertado describirlo como un *ticket* para los desarrolladores, que puede ser un *bug*, una propuesta o un debate de cualquier tipo.

- *Prefabs:* En el contexto de *Unity, un prefab* es un objeto prefabricado y preconfigurado guardado en disco que está listo para ser instanciado como parte de una escena.

- *Roguelike*: Subgénero del género RPG (*Role-playing game*) de videojuegos.

- *SCRUM*: Metodología de trabajo ágil basada en iteraciones cortas (normalmente una o dos semanas) y constante revisión y rediseño de objetivos y acciones.

- *VCS*: Sistema de control de versiones, en el caso de este documento, siempre que se habla de este sistema se habla de *Git.*