Shata

Oscar Alonso Bustos
Diseño y desarrollo de videojuegos
Área de Posgrado Informática, Multimedia y Telecomunicación

Joaquin Campos Colmenareio
Jordi Duch Gavaldà

2020-12-06

Esta obra está sujeta a una licencia de Reconocimiento-Compartir Igual 3.0 España de Creative Commons

FICHA DEL TRABAJO FINAL

Título del trabajo:	Shata
Nombre del autor:	Oscar Alonso Bustos
Nombre del consultor/a:	Joaquin Campos Colmenareio
Nombre del PRA:	Jordi Duch Gavaldà
Fecha de entrega (mm/aaaa):	01/2021
Titulación:	Diseño y desarrollo de videojuegos
Área del Trabajo Final:	Área de Posgrado Informática, Multimedia y Telecomunicación
Idioma del trabajo:	Español
Palabras clave	Desarrollo de Videojuego

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

El diseño y desarrollo de un videojuego no es algo que se pueda hacer a la ligera, sino que se necesita de tiempo y recursos para poder alcanzar a realizar aquello que se ha pensado.

Llevar una idea simple de un papel a un videojuego completo requiere de mucho tiempo y de saber cómo y que se debe hacer. Requiere planificación y establecer todos los objetivos que se quieren alcanzar, obviando aquellos que esta fuera del alcance, ya sea por presupuesto o por capacidad de desarrollo.

En este proyecto, se pretende establecer una idea y, en un tiempo establecido, llevarla a cabo. No hay otro limite que el tiempo y mi propia capacidad. Todo lo aprendido a lo largo del master y todo lo que pueda aprender durante el desarrollo se debe poner en práctica aquí para desarrollar el mejor producto que sea capaz de hacer por mi propia cuenta.

La idea de la que se parte es la de desarrollar un videojuego de construcción y gestión de ciudades, donde el jugador deberá expandirse y sobrevivir.

Abstract (in English, 250 words or less):

Creating a video game is not an easy task that can be done in a few days. It takes a lot of time and resources to create what was planned.

When someone wants to create a video game, no matter how simple the original idea is, he will have to know how and what to do. He needs to plan and establish the goals that he wants to be implemented in the game, he needs to put on the table what he is able to do and what he is not, the time and resources that he has and so on.

Everything I have learnt over these two years of the master's must be put into practice to develop the best product I can do on my own. The time I was granted and my ability are the only two limitation I have to create the game.

The game I am creating is a city building game, where the player will have to expand and build to create a large and sustainable city.

Índice

Tabla de contenido

1. Introducción 1
1.1 Contexto y justificación del Trabajo 1
1.2 Objetivos del Trabajo 1
1.3 Enfoque y método seguido 1
1.4 Planificación del Trabajo 2
1.4.1 Versión parcial 3
1.4.2 Versión jugable 4
1.4.3 Entrega final 5
1.5 Breve sumario de productos obtenidos 5
1.6 Breve descripción de los otros capítulos de la memoria 5

2. Estado del arte 7
2.1 City-Building como género 7
2.2 Principales características 7

3. Definición del juego 9
3.1 Conceptualización 9
3.1.1 Breve descripción del juego 9
3.1.2 Tipo de interacción juego-jugador 9
3.2 Ambientación 10
3.2.1 Ambientación e historia 10
3.2.2 Personajes y elementos 12
3.2.3 Interacción entre los actores del juego 12
3.3 Objetivos planteados al jugador 13

4. Diseño técnico 14
4.1 Entorno 14
4.2 Requerimientos del entorno 15
4.3 Herramientas 16
4.4 Assets 16
4.4.1 Casillas 16
4.4.2 Edificios 19
4.4.3 Interfaz de usuario 22
4.4.4 Sonido 26
4.5 Arquitectura del juego 26
4.5.1 Mapa 29
4.5.1.1 Generación del mapa 30
4.5.1.2 Parámetros 33
4.5.2 Casillas 33
4.5.4 Tipo de casillas 34
4.5.5 Tipos de edificio 34
4.5.6 Almacén y recursos 34
4.5.7 Eventos 34
4.6 Mecánicas implementadas 35

5. Diseño de niveles 37

5.1 Testeo y balanceo **42**
5.1.1 Testo 43
5.1.2 Balanceo 43
6. Manual de usuario **44**
6.1 Controles **44**
7. Conclusiones **45**
7.1 Lecciones aprendidas **45**
7.2 Logros alcanzados **45**
7.3 Seguimiento y análisis **46**
7.4 Mejoras **46**
7.5 Links y recursos **47**
8. Glosario **48**
Bibliografía **50**

Lista de figuras

Ilustración 1 Diagrama Gantt planificación total 2
Ilustración 2 Diagrama Gantt planificación versión parcial 3
Ilustración 3 Diagrama Gantt planificación versión jugable 4
Ilustración 4 Casilla básica (grass) 17
Ilustración 5 Casilla fertile (fertile) 17
Ilustración 6 Casilla bosque (forest) 18
Ilustración 7 Casilla montaña baja (low mountain) 18
Ilustración 8 Casilla montaña (mountain) 18
Ilustración 9 Casilla montaña alta (high mountain) 19
Ilustración 10 Casilla agua (water) 19
Ilustración 11 Casilla agua profunda o mar (deep water) 19
Ilustración 12 Edificio granja (farm) 20
Ilustración 13 Edificio corrall (corral) 20
Ilustración 14 Casa de pescadores (fisher hutt) 20
Ilustración 15 Edificio casa (house) 21
Ilustración 16 Edificio molino (mill) 21
Ilustración 17 Edificio mina (mine) 21
Ilustración 18 Edificio serreria (sawmill) 22
Ilustración 19 Edificio almacén (storage) 22
Ilustración 20 Edificio ayuntamiento (townhall) 22
Ilustración 21 Botón UI 23
Ilustración 22 Menu de la UI 23
Ilustración 23 Combinación de 3 tipos distintos de menú 23
Ilustración 24 Imagen contenedor de indicador recursos 24
Ilustración 25 *Background* recurrente 24
Ilustración 26 Botón apertura menú *In-Game* 24
Ilustración 27 Botón de construcción de edificios y titulo 24
Ilustración 28 Iconos de los recursos 24
Ilustración 29 Menu superior con elementos y recursos 24
Ilustración 30 Recursos popOver para mas información 24
Ilustración 31 Panel con información casilla 25
Ilustración 32 Panel información edificios 25
Ilustración 33 Panel información edificio subir nivel 26
Ilustración 34 GameManager arquitectura primeriza 27
Ilustración 35 Arquitectura Shata [4] 28
Ilustración 36 Arquitectura mapa 29
Ilustración 37 Esquema generación mapa 30
Ilustración 38 Arquitectura casilla 33
Ilustración 39 Pantalla de inicio 37
Ilustración 40 Menú de créditos 37
Ilustración 41 Mapa de Shata al iniciar partida 38
Ilustración 42 Construcción del ayuntamiento 39
Ilustración 44 Construcción del aserradero 39
Ilustración 45 Localización de la producción 40
Ilustración 46 Construcción casa 40
Ilustración 47 Construcción granja 41

Ilustración 48 Post construcción granja 41
Ilustración 49 Construcción segundo aserradero 42

1. Introducción

1.1 Contexto y justificación del Trabajo

Desarrollar un videojuego es una tarea que requiere un esfuerzo titánico y más cuando no se tiene experiencia en dicho proceso. Hay miles de cosas a tener en cuenta que pueden ser pasadas por alto y luego volver con más fuerza, haciendo que perdamos más tiempo y recursos de los que hubiesen sido necesarios en un principio. La finalidad de este trabajo es hacernos pensar en todos estos puntos mientras desarrollamos el videojuego que nosotros queramos.

No hay mejor forma que desarrollar un videojuego de principio a fin para que se ponga en práctica todo lo que se ha estudiado durante estos dos años. Esto significa, hacernos trabajar desde el punto en el cual se nos ocurre la idea del videojuego, hasta que lo hemos desarrollado. Pasando por todos aquellos puntos intermedios, que incluye no solo el diseño de los personajes y los niveles y la programación del mismo videojuego, sino también la planificación de los recursos y del tiempo, del trabajo que se ha de realizar, de establecer que podemos hacer y que no dentro del tiempo del que disponemos. Todos estos puntos mencionados y muchos más se van a explicar en este documente que nos invita a conocer el videojuego en profundidad.

1.2 Objetivos del Trabajo

El objetivo de este trabajo está claro: desarrollar un videojuego con todo lo que ello conlleva, es decir, el videojuego final y su correspondiente Documento de Diseño de Videojuego.

Este es el objetivo final que tenemos nosotros, sin embargo, el verdadero objetivo, como ya se ha comentado en el punto anterior, no es solo que el producto final, es decir, el videojuego, este completo y bien, si no entender y realizar todo lo que nos permite llegar a ese punto: la planificación y el diseño previo del videojuego, así como el desarrollo como tal.

Intentar llegar y desarrollar todo lo que se ha “prometido” en las primeras entregas significa una buena gestión y saber aplicar los recursos de los que se ha dispuesto. Este es el objetivo de este trabajo de final de máster.

1.3 Enfoque y método seguido

Como ya se ha comentado, no hay mejor forma que aprender a desarrollar un videojuego que ponerlo en práctica.

A partir de una idea, se debe realizar todo el proceso de desarrollo, sin saltarse ningún punto. Como ya veremos, estableceremos y planificaremos las tareas, se creará un documento de diseño del videojuego y se programará el propio juego,

haciendo una pequeña demo que tendrá las principales mecánicas del juego que se ha pensado crear.

En este capítulo se van a presentar los hitos y las entregas que se deben cumplir, que debemos haber implementado en nuestro videojuego para llevar los tiempos correctamente. No se tiene que cumplir el 100% de lo establecido, ya que el desarrollo es un proceso vivo que se modifica a lo largo del tiempo y pueden salir imprevistos, aunque si debemos intentar acoplarnos a él en la medido de lo posible.

1.4 Planificación del Trabajo

En este proyecto hay tres días de entrega marcados, en los cuales hay que cumplir unos mínimos.

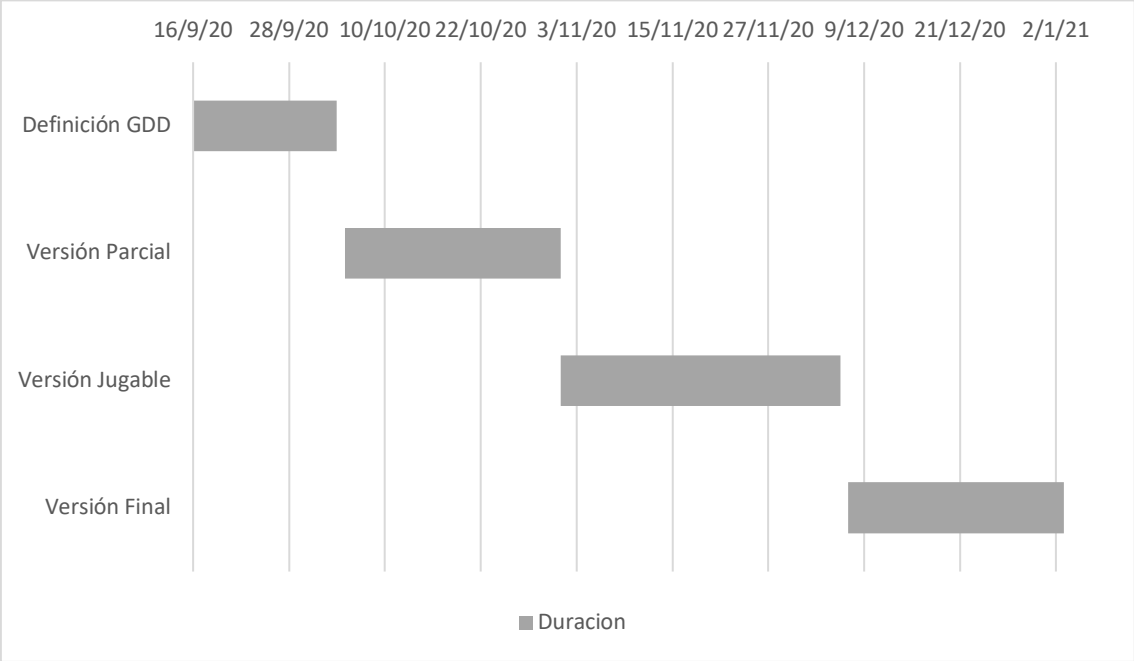

Ilustración 1 Diagrama Gantt planificación total

Es la creación de este documento inicial, que se entregó en la PEC1. Este servirá como guía para el desarrollo del juego. No solamente es una presentación, si no que intenta establecer los limites, así como el tiempo disponible para su creación y los pasos o *timings* que se deberían seguir para garantizar su éxito. Es el inicio del desarrollo y representa el alcance, en el se propone que se va a hacer y que no.

En las dos siguientes entregas se intentará desarrollar todos los puntos que se han establecido en este documento llamado GDD[1]. A continuación, se explican y se presenta otros Diagramas de Gantt donde se pueden ver, por encima, las tareas que se quieren realizar.

[1] Game desing document

1.4.1 Versión parcial

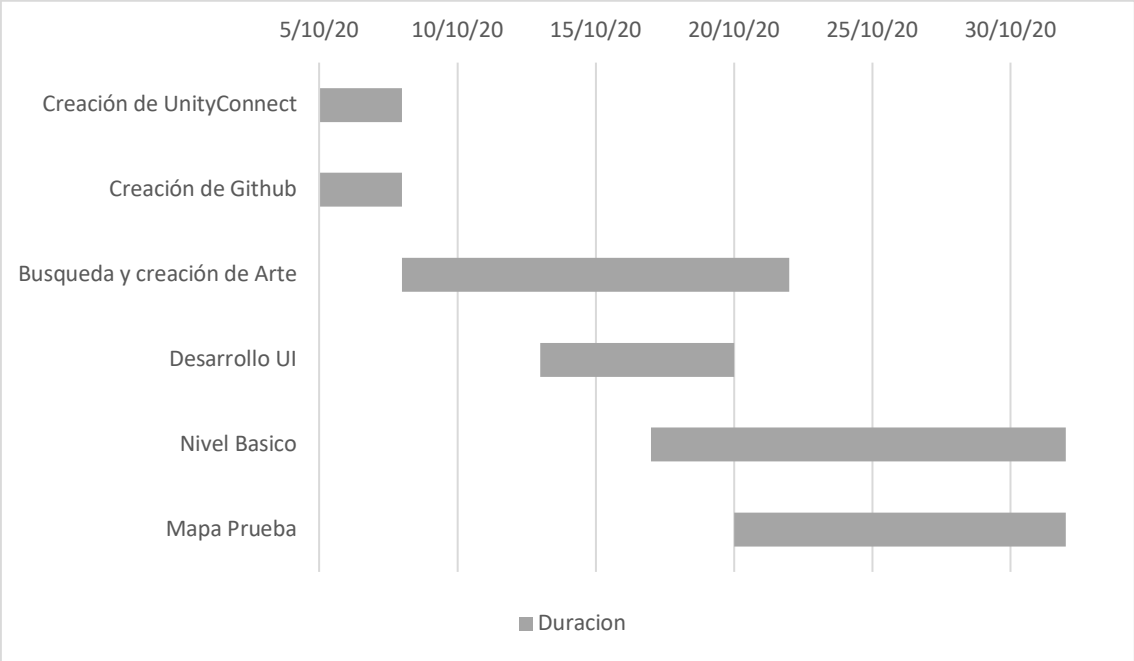

Ilustración 2 Diagrama Gantt planificación versión parcial

En esta primera entrega, se debe entregar una primera versión del juego, donde ya exista el principal flujo del juego. Esto significa que deben existir las pantallas de carga, menú principal, créditos, *game over*, pantalla de selección de niveles, menú de pausa y una pantalla de nivel[2].

Por otra parte, también se debe preparar otros elementos para poder trabajar en el videojuego. Esto significa que, como se puede ver en el diagrama de *Gantt*, se dedicarán unas horas a la creación de un repositorio de *Github* y de *UnityConnect*, y asegurarse que todo funciona correctamente[3], ya que esto son unos requisitos indispensables para la primera entrega o PEC. También se dedicará tiempo a crear y definir el arte que se usará en el posterior desarrollo. Una vez realizado eso, se empezará con la creación del videojuego como tal.

Aunque un juego puede realizarse con elementos visuales tan simples como cuadrados, círculos y triángulos, se va a dedicar un buen porcentaje de los recursos a la creación del arte que se usará en el videojuego. Así pues, durante dos semanas se dedicará a crear todos los elementos 3D o de la interfaz gráfica que vamos a necesitar. Sin embargo, esto es un poco mentira, ya que la búsqueda y creación de Arte no acabará realmente hasta bastante mas tarde, ya que siempre se puede actualizar o crear nuevos elementos, pero el groso de la tarea debería haber acabado en ese entonces.

---

[2] No será un nivel jugable completo, si no una pequeña demo donde se pueda acceder al menú de pausa, tal vez el planteamiento y posicionamiento del mapa y la cámara… Lo mas importante es el flujo completo de la interfaz de usuario

[3] Quiero especificar que la creación de estas cuentas es algo que no requiere realmente tanto tiempo como se le han asignado, pero debido a que no dispongo de tanto tiempo, he intentado reflejar el tiempo que tengo disponible debido al trabajo y otros estudios.

El desarrollo de la interfaz de usuario, del mapa de prueba y de las mecánicas básicas se realizarán casi casi al mismo tiempo, y se irán modificando mientras se trabaja en la “siguiente” tarea. Sin embargo, si que se puede visualizar un pequeño escalonamiento debido a la pequeña dependencia que hay entre ellos. Las mecánicas básicas se refieren a moverse por el mapa y poder seleccionar las casillas, así como los paneles básicos de la UI in-game, pero sin indagar demasiado en la construcción de edificios o la gestión de recursos. Como puedes suponer, la mecánica del moverse por el mapa no se puede empezar hasta que el mapa de prueba no este, como mínimo, empezado[4].

1.4.2 Versión jugable

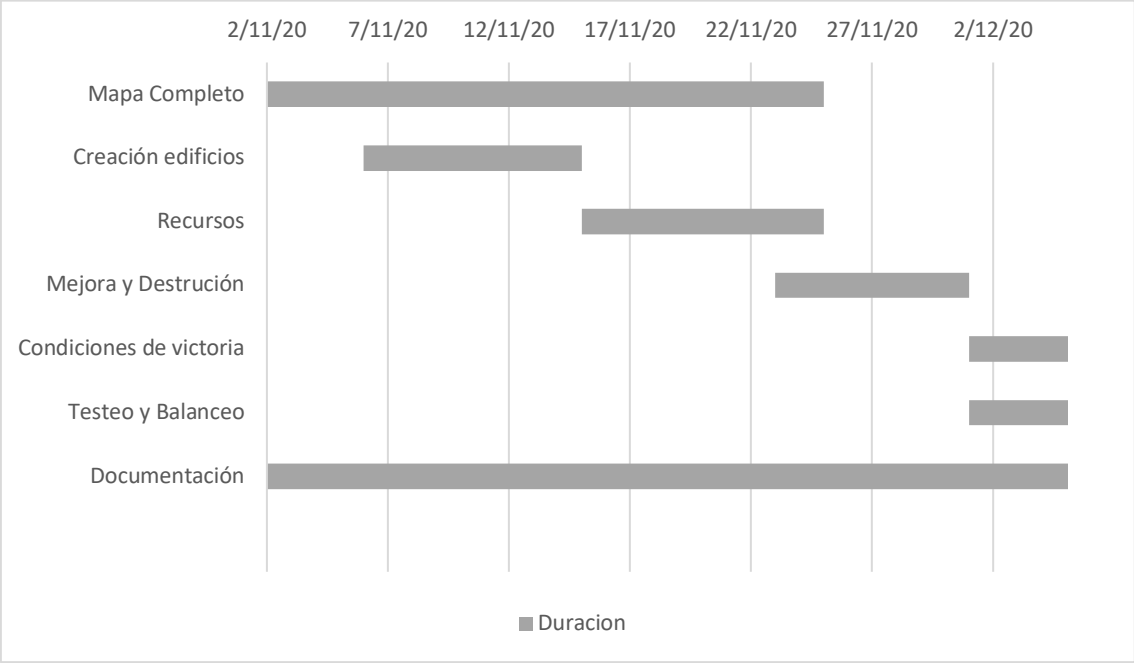

Ilustración 3 Diagrama Gantt planificación versión jugable

En esta entrega se terminarán las mecánicas que se han propuesto y que se han empezado en la entrega anterior, así como un primer mapa que pretende mostrar el alcance del videojuego.

En este proyecto, el punto más importante del juego o, donde se ha decidido invertir más recursos y tiempo, es en la creación de un mapa aleatorio[5], que sea distinto en cada partida pero que a su vez pueda ser recuperado para ser jugado de nuevo si así lo de sea el usuario. Un total de 3 semanas, que corresponde casi a un 40% del tiempo de las dos entregas es de lo que se dispone para

[4] De hecho, al final esta mecánica se acabo en la siguiente entrega, ya que fue cuando realmente se acabo el mapa y por lo tanto era necesario en este punto y no antes, dejando de ser una de las “mecánicas básicas”

[5] Cabe decir que esto no era así cuando se realizo la PEC1, el mapa iba a ser uno estático, pero se decidió cambiar ya que quedo claro que era una de las cosas mas importantes a la hora de este tipo de juego.

realizarlo. Obviamente, dado que soy el único programador intercalaré esta tarea con otras para no saturarme.

Quitado de este punto, las demás mecánicas como los edificios o manejos de recursos deberían hacerse en relativamente poco tiempo.

Para finalizar, solo quedará testear y documentar el videojuego, que se esta planeado realizar en esta entrega y en la siguiente.

1.4.3 Entrega final

En esta última entrega se realizarán mejoras en este documento, se preparará la defensa y se realizará el video y los elementos necesarios para la entrega.

En el apartado técnico, se realizarán pruebas y balanceos para dejar un juego más pulido, que, aunque no tenga muchas mecánicas desarrolladas, aquellas planteadas y alcanzadas estén acabadas y funcionando a la perfección.

Uno de los elementos que se ha tenido que aplazar para esta fase es la mejora de edificios, ya que, aunque se ha dejado preparada en la entrega correspondiente, no se ha podido finalizar. Por otro lado, se desea implementar una pantalla de carga, donde se ejecute en *background* el algoritmo de la creación del mapa[6]

1.5 Breve sumario de productos obtenidos

El objetivo de este proyecto es obtener una pequeña versión jugable de un videojuego. Se obtendrá un fichero ejecutable para cualquier ordenador que cumpla los requisitos mínimos. Debe ser una versión que muestre las mecánicas que se quiera implementar en la versión final, aunque ya que es una demo, no tienen por qué estar todas las mecánicas que se requieren para sacar un juego completo al mercado.

1.6 Breve descripción de los otros capítulos de la memoria

La estructura que se va a seguir para enseñar el trabajo realizado va a ser de la siguiente manera.

Primero se explicará un poco el género del videojuego, así como la situación actual de este tipo de videojuegos, para poner en contexto al lector y dar a entender el tipo de videojuego que se quiere desarrollar.

Posteriormente se dará una pequeña introducción al mundo del videojuego, la historia del mundo en el que nos encontramos y como se ha llegado a este

---

[6] Si no hay tiempo no se realizará ya que esto y otras cosas no estaban en el *planning*, pero son elementos interesantes.

punto… También se plantearán los objetivos que tiene el jugador a lo largo de su aventura.

En este punto, empezamos a ver más afondo el videojuego en sí. Se mostrarán los *assets* que se han usado, las herramientas, el entorno de desarrollo, etc. Además, se presentará la estructura del videojuego, como se han creado los componentes y la relación entre ellos.

Una vez acabado con la parte técnica, se pasará a un ámbito más de diseño, donde se mostrará un nivel, en el cual se hará un *walkthrough* completo, indicando que es lo esperado por parte del jugador para avanzar.

Para finalizar se mostrarán los controles básicos y las conclusiones que se han sacado al desarrollar este proyecto.

2. Estado del arte

2.1 City-Building como género

Como ya se ha comentado, *Shata* es un juego que pertenece al género de construcción y gestión de recursos, específicamente, construcción de ciudades no violento. Como en todos los juegos de este género, el jugador debe saber gestionar los recursos que se le ofrecen para poder lograr su objetivo, en este caso construir una ciudad. No violento significa que no hay enemigos en el mapa contra los que luchar, ni tampoco amenazas ajenas que supongan un conflicto del que defenderse.

La saga por excelencia de este genero y, seguramente, creadora del genero, es *Simcity*, publicado el 1989. Este juego eliminó la condición de victoria que existía en todos los juegos. El único objetivo del jugador era hacer sobrevivir a su creciente ciudad, enfocándose en la gestión de recursos. Posteriormente, llegaron juegos que iban enfocándose en otros aspectos más concretos (como la saga *Anno*, 1998, que se focalizaba en la gestión puramente económica) o añadiendo más jugabilidad y opciones al juego (por ejemplo *Stronghold*, 1993, que añadía soldados y batallas).

Hoy en día existen infinidad de juegos, cada uno con su propia complejidad y focalidad en aspectos distintos, pero todos ellos “simulando” la realidad. Algunos ejemplos, descartado los ya mencionados, son las sagas *Caesar*, *Settler* o juegos mas *indie* como *Banished* y *Before we leave*, siendo este ultimo mas parecido a *Shata* debido a su sistema de hexágonos (similar a *Civilization*). Aunque todos ellos son muy distintos, comparte un grupo de características entre ellos que los define y encaja en este género.

2.2 Principales características

Todos los juegos de este tipo comparten, como mínimo las siguientes características:

- Mapa de tamaño considerable, normalmente con casillas o con algún tipo de rejilla que facilita la construcción y emplazamiento de los edificios.
- Recursos esparcidos de forma aleatoria por el mapa que puede ser extraídos por el jugador y que se usarán para mejorar y avanzar en la partida.
- Recursos son limitados, ya sea en el tiempo (en velocidad de recolección, por ejemplo) o de cantidad máxima disponible.
- Edificios o similares que permiten al jugador avanzar y usar los recursos que ha obtenido para, en esencia, conseguir de nuevos.
- Casillas que permiten la localización de los edificios en el mapa. Esto puede ser de varios tipos, alguno tan simple como el de *Shata*, en el cual una casilla permite un solo edificio, o algo tan complicado como una cuadricula en la cual múltiples casillas forman un edificio completo, cada uno de ellos de diferente tamaño y, por lo tanto, necesitando un numero de casillas diferente preestablecido.

- Necesidad de manejar los recursos correctamente, pudiendo llegar a un punto muerto donde no puedes avanzar más o, vuelve a empezar.

Todas estas características están de una forma u otra implementadas en Shata. Pero hay muchas otras mecánicas que son muy habituales y recurrentes en este tipo de juego como pueden ser:

- Existencia de enemigos o adversidades externas a la gestión de tu ciudad a las que te debes enfrentar.
- Mercado con entidades externas que favorecen el desarrollo.
- Recursos más avanzados fuera de los típicos. Hay un sin limite a la hora de añadirlos. El juego *Kingdom Reborns*, actualmente en desarrollo, tiene elementos que entran tan en profundidad, simulando la realidad, como puede ser el ejemplo del trigo, que no solo se come, sino que sirve para hacer harina, cebada, cerveza, pan y para ello se tiene que usar carbón y madera, dando como resultado comida y/o felicidad (que es a su vez otros recursos).
- Asignación de los pobladores en los puestos de trabajo
- Enfermedades
- Elementos históricos
- Cartas y mejoras
- Eventos aleatorios que provoca que las partidas sean diferentes entre ellas
- Modo historia
- Etc, etc..

3. Definición del juego

3.1 Conceptualización

3.1.1 Breve descripción del juego

Shata es un juego de construcción de ciudades donde el jugador debe expandir su ciudad, haciendo de crezca y sobreviva, usando de forma adecuada los recursos que existen en el mapa. El único final posible es que se agoten los recursos disponibles, debido a una mala gestión de estos, provocando que se deba volver a empezar, al haber alcanzado un punto muerto.

¿Cuánto tiempo será capaz de sobrevivir el jugador hasta que esto ocurra?

3.1.2 Tipo de interacción juego-jugador

En todos los juegos de este genero, la interacción que existe entre juego y jugador está bastante limitada y es bastante parecida en todos los juegos. Sin embargo, hay distintas formas de aplicar lo mismo, pudiendo llegar a ser única e innovadora en cada juego. Aquí se explica como se ha decidido implementar estas mecánicas en el juego final de *Shata*.

La partida se desarrolla sobre una especie de tablero con casillas hexagonales que representa el mapa. Cada casilla tendrá unas características y recursos específicos, siendo estos condicionantes para que ciertos edificios puedan ser o no construidos. Estos edificios tienen, a su vez, ciertas características que los hace ser distintos unos de otros. Por ejemplo, algunos permiten la extracción y recolección del recurso existente en la casilla, otros dan beneficios al jugador, otros aumentan el máximo de recursos almacenables disponibles… Además, los edificios tienen un coste asociado, una cantidad de recursos que necesitas tener previamente a la construcción del mismo.

Cada una de las casillas es *clicable*. Al hacerlo, un pequeño panel aparecerá en el lateral izquierdo, explicando las propiedades, recursos disponibles y mostrando una pequeña descripción de la casilla seleccionada. Por otra parte, en el lado derecho aparecerá, si se requiere, otro panel con todos los edificios que se pueden construir o, en el caso de que ya este construido un edificio, mejorar[7]. En ambos casos también aparecerá el coste y una pequeña descripción sobre los beneficios de construir/mejorar el edificio. En este mismo panel, existirá la opción de demoler el edificio. Por ultimo, la casilla seleccionada tendrá un pequeño borde para poder reconocerla fácilmente, ayudando así a situarse en el mapa.

El mundo se ve con una vista isométrica con un ángulo de unos 45 grados. El jugador solo podrá mover la cámara hacia los lados o de arriba abajo, siempre en los ejes X y Z, nunca en el Y. Lo que si que puede hacer el jugador es *zoom*.

---

[7] Los edificios se pueden mejorar dos veces, hasta que alcanzan el nivel 3

Algo común en estos juegos a la hora de hacer *zoom*, no es simplemente que la cámara se mueve en el eje Y, sino que además hace una pequeña rotación, permitiendo ver el juego desde una perspectiva próxima a ser paralela al terreno. Obviamente, tanto el zoom como la rotación tiene un máximo controlada, impidiendo que se traspase el mapa o se den *glitchs* similares.

La llamada niebla de guerra es otro elemento que el jugador deberá tener en cuenta, ya que le impide ver el mapa y por lo tanto no puede saber con seguridad que acciones le beneficiarán más a largo plazo. Esta niebla desaparece al construir edificios cerca de ella.

Otra mecánica relacionada con las limitaciones de construcción que hay que tener en cuenta es que no se puede construir en cualquier casilla cuando se desee, sino que todos los edificios deberán tener un camino que permita comunicarlo con el resto.

En una barra superior aparecerá toda la información necesaria para comprender el estado actual del juego, como puede ser el dinero disponible, la comida, cantidad de habitantes… También se puede ver la cantidad de consumo y la producción actual de cada uno de los recursos de los que disponemos, así como una opción de "Menú" y otra de "Pausa". Mientras el juego esta en pausa, no se consumirán ni generarán recursos, pero se podrá interactuar con el mundo.

Por otra parte, se podrá jugar dos tipos de partida[8]. En las del primer tipo, no se acabará nunca la partida, el jugador simplemente decidirá cuando desea parar, ya sea por que se ha quedado atascado y no dispone de mas recursos, o por que todas las casillas han sido construidas y mejoradas. En la otra, se establecerá un objetivo a alcanzar, como puede ser tener un numero de pobladores, casillas construidas, dinero…

3.2 Ambientación

3.2.1 Ambientación e historia

Shata es un mundo de fantasía, ambientado en una época parecida a la medieval de nuestro mundo. Sin embargo, la historia sucede cientos de años en el futuro, en un mundo lejano al cual se le llamó *Shata*.

Todo empezó en el año 2019, cuando apareció el primer caso de un nuevo virus que fue llamado COVID-19. El virus, que se cree originario de china, se expandió de forma rápida por todo el mundo, haciendo que la economía de los todos los países se viese sumamente afectada, debido a las medidas de contención que se tuvieron que aplicar. El año 2020 pasó, con la inmensa mayoría de los ciudadanos encerrados en sus respectivas casas, para evitar el contagio del virus a mayor escala. Un año entero sin poder salir de casa, donde la gente se

[8] En la versión completa del juego habría estos dos tipos de juego. Pero no entra en el planning del desarrollo del demo, asi como otras mecánicas y/o personajes de la historia.

quedaba sin trabajo y centeneras de empresas se vieron muy afectas minaron la moral de la población. Sin embargo, estaba a punto de pasar algo que nadie nunca podría haber predicho.

En el año 2022 nada había cambiado y el virus se esparcía cada vez más, sin que pudiese ser contenido. La población de los países más ricos salió a la calle para manifestarse y pedir más control y responsabilidades. Esto tuvo un efecto contrario, en el cual miles de personas se contagiaron. Los hospitales estaban colapsados y no daban abasto con la cantidad de gente que acudía. Durante 15 años la situación no mejoraba. La población mundial empezó a decaer, siendo la primera vez que esto pasaba debido a una razón no relacionada con las guerras, si se excluye la peste negra.

La población empezó a emigrar a zonas rurales, intentando encontrar focos de población reducidos, para minimizar el contagio. La producción de materias de primera necesidad empezó a escasear. Fue entonces cuando, en escala global empezaron los robos y los pillajes, que acabaron evolucionando a saqueos. Algunos países entraron en una guerra civil y sus países vecinos aprovecharon para intentar conquistarlos y quedarse con sus recursos.

En este punto, año 2037, todo el mundo enloqueció. Los países se culpaban los unos a los otros, las tensiones crecían y la población decrecía de una manera escandalosa. En el año 2040 ya no existía el concepto de país. No había gobiernos. Las ciudades estaban vacías. Grupos de ciudadanos se juntaban en lugares recónditos y se aislaban del resto, sin pensar ni un solo segundo en disparar a cualquier persona que viniese de fuera.

Mientras todo esto sucedía, un grupo de ingenieros liderado por el programador Oscar Alonso lideró una campaña loca y peligrosa. La creación de una nave que permitiese a la humanidad llegar y reproducirse en otro planeta. No fue hasta el año 2053 que las naves con tripulantes que buscaban otros planetas salieron de la tierra. Un total de 24 naves, con una media de 35 y 40 expertos en diferentes materias que se consideraban necesarias para empezar la nueva vida y controlar la nave.

No se sabe nada de las 23 naves restantes que, hace tiempo, desaparecieron del radar. Pero 1 nave, llego a un planeta después de 12 años a la deriva. La nave se llamaba *Shata,* y ese fue el nombre que le dieron al planeta. El planeta era muy similar al planeta tierra, pero mucho mas grande y con mucha mas agua y, por lo tanto, con muchas mas islas. En una de estas, que llamaron *Origen* es donde aterrizó la nave. Durante 3 generaciones, la humanidad se dedicó a crear un pequeño pueblo donde pudiesen vivir tranquilamente, pero la humanidad crece muy rápido y necesita de muchos recursos, los cuales no estaban presentes en la isla o se habían agotado. Así pues, en el año 140 de *Shata,* que corresponde al año 2195, la humanidad que sobrevivió al éxodo de la tierra, se aventuro a otra colonización.

3.2.2 Personajes y elementos

Diversos personajes salen de la isla en la que la humanidad ha vivido durante 140 años para encontrar otras islas ricas en recursos, donde se pueda desarrollar otra colonia que sirva como residencia a los habitantes de la isla *Origen,* así como de proveedor de recursos.

La gobernanta de *Origen*, *Jenna*, es la encargada de asegurar que los colonizadores tienen todos los recursos necesarios y que reciben la ayuda apropiada para poder realizar la misión. Además, también fue la encargada de elegir y montar los equipos. Alguno de los personajes que aparecen son:

- *Jenna*: Gobernanta de la primera ciudad de *Shata*. Es también la jefa del Consejo de Expedición o, mejor dicho, la representante y mediadora ya que, a pesar de no tener voto, es la que rige las reuniones y la que hace saber a los afectados las decisiones tomadas. Mujer bastante alta y con un pelo negro como la noche, siempre con una coleta alta. Sus ojos azules casi plateados contrastan con el oscuro de su pelo. Es bastante estricta cuando el tema en cuestión es el trabajo, pero bastante amable y siempre dispuesta a ayudar. Todo el mundo confía en ella ciegamente y acata sus ordenes sin cuestionar el porqué. Como todos los actuales habitantes, nació en la isla y ha sido criada escuchando las historias de la pasada. Sueña con ver otras partes del nuevo mundo.
- *Ben*: Es el primer explorador que aparece en el juego. Con solo 17 años, es bastante inexperimentado así que *Jenna* le ayuda constantemente a moverse por el mapa y saber que tiene que hacer. Rubio y de ojos azules, de estatura mas bien baja. Sus padres están bastante en contra debido a que es bastante joven como para salir de la isla, pero el no aguantaba más el estar encerrado. Es el personaje que aparecerá en el tutorial. Una vez superadas diversas misiones y se desbloquee el modo juego libre, es también el personaje con el que jugamos.
- *Liss*: Segunda exploradora que aparece en el juego. *Jenna* confía mas en su experiencia así que tiene un poco mas de libertad. Es de estatura baja y bastante tímida. Rubia con el pelo muy largo que le llega casi hasta la cintura. Ojos marrones casi negros y con una sonrisa que, a pesar de ser difícil de sacar, enamora a todo el que la ve. Su padre pertenece al Consejo de Expedición y su madre es la mano derecha de *Jenna*, así que ambos esperan que sea ella quien cree la primera colonia habitable, cosa que ella no tiene tan claro. Su hermano no ha querido ir a ninguna expedición y prefiere quedarse en la isla ayudando a construir edificios, lo que ha creado una dura discusión dentro de la familia. Desea acabar rápido su misión para volver a casa, solucionar el problema y vivir junto a su familia de nueva. Eso si, habiendo cumplido su misión.

3.2.3 Interacción entre los actores del juego

A lo largo del desarrollo del juego, *Jenna* hablará con nuestro colonizador para darle consejos o misiones que debe cumplir. También puede, de forma ocasional, dar regalos y recursos para facilitar el crecimiento del pueblo.

Estas interacciones suceden mediante un cuadro de texto semitranslúcido que aparece en la parte superior de la pantalla, cubriendo casi el total de esta. El texto aparecerá completamente, y habrá un botón para pasar al siguiente y otro para cerrar por completo el dialogo. Estas interacciones son más bien monólogos, donde solo intervienen *Jenna* u otros personajes, como puede ser algún ciudadano descontento que nos informa de algo, pero nunca nuestro héroe

Así pues, el tutorial avanzaría de la siguiente forma, iniciando con un pequeño dialogo que podemos ver aqui:

- *Jeena: Ben,* por fin habéis llegado a tierra. Es enorme. Seguro que es mucho mas rica en recursos que *Origen*. Me estoy emocionando, pero todavía queda mucho trabajo por hacer. Lo primero que tienes que hacer es construir el ayuntamiento (*townhall).* Habras visto que no tienes recursos, pero no te preocupes que es gratis y una vez construido te enviaremos recursos y aldeanos para empezar tu pequeño pueblo. ¡Vamos! Constrúyelo donde mas te guste, aunque te recomiendo que tengas al alcance los recursos necesarios si quieres expandirte de forma adecuada.

Vemos que esta conversación sirve para poner un poco de contesto al jugador, pero también para indicarle como moverse por el mundo. Posteriormente, en el mismo tutorial se usan este tipo de conversaciones para guiar al jugador en el desarrollo del nivel.

3.3 Objetivos planteados al jugador

Como ya se ha comentado previamente, habrá dos tipos de partida. En el primer tipo, el jugador será guiado por *Jenna*, que le dará diversas misiones parecidas a la que hemos visto en el punto anterior.

Otros ejemplos de estos objetivos pueden ser:

- Mejora el centro de la ciudad a nivel X (siendo X > 1 y <= 3)
- Consigue X unidades de madera y Y unidades de comida para superar el invierno (siendo X e Y un número entre 500 y 2500, dependiendo del nivel)
- Consigue una población superior a X (siendo x > 25)

En el segundo tipo de partidas, el caso en el que no hay misiones, el objetivo es simplemente sobrevivir y ser capaz de llevar el crecimiento de la ciudad a nuevos limites que el jugador no haya podido alcanzar previamente, siendo el propio jugador el que se propone los objetivos, con el fin de superarse y aprender las combinaciones para que le permitan superar los niveles con misiones.

4. Diseño técnico

4.1 Entorno

Los dos principales motores de juegos que compiten por la supremacía del mercado son *Unity* y *Unreal Engine*, sin embargo, hay otros motores y kits de desarrollo a tener en cuenta, sobre todo cuando se trata de un estudio independiente que no dispone de los recursos para usar uno de estos dos grandes *engines*. *Game Maker Studio* 2 y *cocos2d* son dos motores de videojuegos a tener muy en cuenta, debido sobre todo a su facilidad de uso y su reciente popularidad. Obviamente hay muchos otros, pero la evaluación se va a hacer entre estos 4.

Tanto *Game Maker Studio 2* como *Cocos2d*, están especializados en realizar juegos en 2D y, aunque nuestro juego se podría haber desarrollado perfectamente en solo dos dimensiones, he decido jugar un poco con la tercera dimensión por primera vez[9]. Por esta razón, estos dos motores han sido descartados de manera casi inmediata una vez tome la decisión de probar de hacer el videojuego en 3D (o 2.5D). Incluso ignorando esto, dudo que algunos de estos dos hubiesen sido escogidos incluso si el videojuego hubiese sido en 2D por la siguiente razón que, bajo mi punto de vista, es de las mas importante a la hora de escoger un *Game Engine* o un *Framework* con el que trabajar: No tienen una comunidad, ni una documentación ni unos tutoriales tan potentes como los motores *Unity* o *Unreal Engine*. Independientemente de la potencia de un motor, una de las cosas más importantes es la facilidad de uso y la accesibilidad a los recursos necesarios para poder llevar a cabo las tareas a realizar.

Por otra parte, *Unreal Engine* es motor de videojuegos muy potente, usado en muchos juegos que han vendido miles de copias a lo largo del mundo. *Gears of wars, Dishonored* y *Daylight* [1] son alguno de los videojuegos que han sido hechos con este motor de videojuego. Si un motor de videojuego ha conseguido que los desarrolladores creen unos juegos tan buenos y completos, ¿Por qué no lo íbamos a usar para desarrollar nuestro videojuego? La razón es fácil: esta potencia tiene sus contras. No es un motor sencillo de usar, y más si se trata de un estudio con muy pocas personas. Tiene una curva de aprendizaje demasiada alta y teniendo en cuenta el tiempo disponible y además que dicho motor usa un lenguaje de programación bastante desconocido para mi como es *C++,* se ha decidido descartar. Como ya se ha comentado antes, la documentación y comunidad es algo a tener en cuenta y la de este motor sigue siendo un poco mas pequeña que la que tiene *Unity*. Todas estas son las razones por las cuales *Unity* ha sido elegido entre las decenas de motores que hay en el mercado.

Así pues, ya como se había adelantado al principio de este capítulo, el motor gráfico usado en este trabajo es *Unity*. Hemos comentado por que no se han escogido los otros motores gráficos, pero *Unity* no se ha escogido por descarte.

---

[9] Fuera de los ejercicios realizados en el máster

Lejos de eso, *Unity* ha demostrado ser un motor de videojuegos de lo más potente, tanto para desarrolladores individuales como para grandes grupos formados no solo de programadores.

Unity se puede descargar y usar gratis, siempre y cuando los beneficios obtenidos con los productos creados con el motor no sean superiores a 100.000 dólares anuales. Muchos de los recursos que proporciona la misma web oficial, ya sean creados por los desarrolladores del motor o por otros desarrolladores de videojuegos que ofrecen sus recursos a quien quiera cogerlo, son gratis y se pueden usar sin restricción alguna.

Por otra parte, la comunidad que hay detrás de este motor es enorme. Tiene un foro propio donde hay miles de usuarios y preguntas y, cualquier duda que tengas, probablemente ya haya sido resuelta antes. Si no es en este foro, en *stackoverflow* también hay muchas preguntas que tratan sobre problemas o dudas relacionadas con este motor. Además, la documentación de las funciones y clases propias de *Unity* es muy completa y con ejemplos. También tienen varios tutoriales que te permiten aprender de mano de los mismos creadores del motor. Sino son suficiente, en *YouTube* también hay miles de desarrolladores que han colgado sus conocimientos. Esto, para mí, es una de las principales razones por las cuales este motor es tan bueno, la comunidad que se ha creado alrededor.

Por último, otra razón es la experiencia que tengo. Tanto el lenguaje como el motor gráfico es algo conocido para mí. Ya he trabajado con ellos anteriormente en varias ocasiones, incluso antes de empezar este master. Esto es algo bastante importante, ya que el tiempo del que dispongo es bastante limitado debido a que trabajo a jornada completa y, el no tener que "perder tiempo" aprendiendo cosas básicas es bastante importante. Aún a pesar de haber usado *Unity,* he querido usar varios componentes nueves que he tenido que aprender a usarlos igualmente.

4.2 Requerimientos del entorno

Según la página oficial de *Unity*[10] los requerimientos mínimos son los siguientes:

- Windows 7 o 10, versión de 64bit
- O MacOs 10.13+
- O Ubuntu 18.04 o 20.4
- GPU compatible con DX10, DX11 y DX12

Para el desarrollo del TFM se ha utilizado tanto un MacOS 10.12.6.[11] como un Ubuntu 20.4[12]. Se ha ejecutado el juego en ambos ordenadores además de en un Windows 10[13].

---

[10] https://docs.unity3d.com/2020.2/Documentation/Manual/system-requirements.html
[11] Procesador Intel Core i7 2,2GHz. 16GB de RAM. Intel Iris Pro 1536 MB.
[12] Procesador Intel Core i7-10875H. 32GB de RAM, GeoForce RTX 2060.
[13] Version de 64bits. Procesador AMD Ryzen 5 1500X. 8.00GB de RAM. Radeon RX 570 series

4.3 Herramientas

Además del programa *Unity* se han utilizado dos programas para la creación y manipulación de los *sprites* que se han utilizado en el desarrollo de este videojuego*:*

- *GIMP2*[14]*:.* Es totalmente gratuito y fácil de usar. Sirve para la manipulación de imágenes, como por ejemplo recortes, escalados, composición de imágenes… Permite trabajar en distintas capas, lo que facilita la reutilización de recurso y trabajan en ellos independientemente.

- *Inkscape*[15]*:* Es totalmente gratuito, y tal vez un poco más difícil de usar que el anterior, aunque también es verdad que su objetivo es completamente distinto. Este programa se usa para tratar de forma vectorial las imágenes, lo que permite el re escalado sin perder definición. Se usa sobre todo para la creación de logos, iconos, etc…

- *Blender*[16]: Es un programa gratuito y *open source* de creación y modelaje 3D. Se ha usado para modificar los *assets* obtenidos de diversas paginas webs, permitiendo que sea un poco más único y personal.

Ambos programas se pueden usar tanto en Windows, Ubuntu y MacOs y no requiere unos requisitos demasiado altos, aunque depende más del uso que se le vaya a dar.

4.4 Assets

En este apartado se van a listar y explicar cómo se han realizado o, mejor dicho, de donde se han obtenido todos los *assets* que se han utilizado. Dado que hay muchos y ponerlos todos podría ocupar una docena de hojas se va a intentar disminuir al mínimo el espacio ocupado, intentando poner todos los ejemplos.

Se expondrán los *assets* por temas, tratando primero las casillas, posteriormente los edificios que hay y, finalmente, la *UI*.

4.4.1 Casillas

Todas las casillas se han obtenido de un conjunto de *assets* totalmente gratuito, dando libertad a todo el mundo a descargarlo y usarlo. La pagina web es https://kenney.nl/ y en ella se pueden encontrar infinidad de recursos para cualquier juego que se te venga a la cabeza. Nuestras casillas se han realizado a partir de dos *assets*: https://kenney.nl/assets/hexagon-kit y https://kenney.nl/assets/nature-kit.

---

[14] Se puede descargar gratuitamente de la página oficial https://www.gimp.org/
[15] Se puede descargar gratuitamente de la página oficial https://inkscape.org/
[16] Se puede descargar gratuitamente de la página oficial https://www.blender.org/

Como podemos observar, si se accede al primer link se encuentra una clara relación entre los *assets* usados en *Shata* y los que hay en esa página web. Esto no solo se aplica a las casillas, sino también a los edificios. Sin embargo, en la web de *Kenney* las casillas ya están compuestas, siendo un único elemento. Se ha tenido que abrir con *blender*[17] y editar. Para realizar las casillas, simplemente se ha eliminado los edificios y se ha dejado el terreno o base del modelo. Ha sido, sin duda alguna, la parte fácil de la creación de los elementos que componen nuestro juego.

A continuación, veremos un listado de todas las casillas. El funcionamiento de estas se explicará en el apartado técnico.

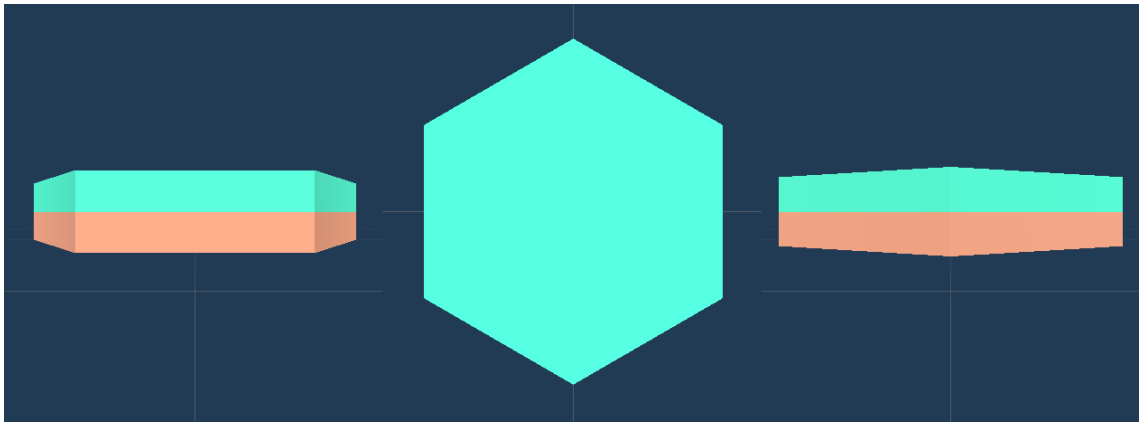

Ilustración 4 Casilla básica (grass)

Podemos ver la alzada, planta y perfil de la casilla básica. Podemos ver que es realmente simple, sin ningún extra. A partir de ahora veremos el resto de las casillas de la misma forma, alzada, planta y perfil, y, podremos ver como se les van añadiendo pequeños detalles que las hace parecer completamente distintas.

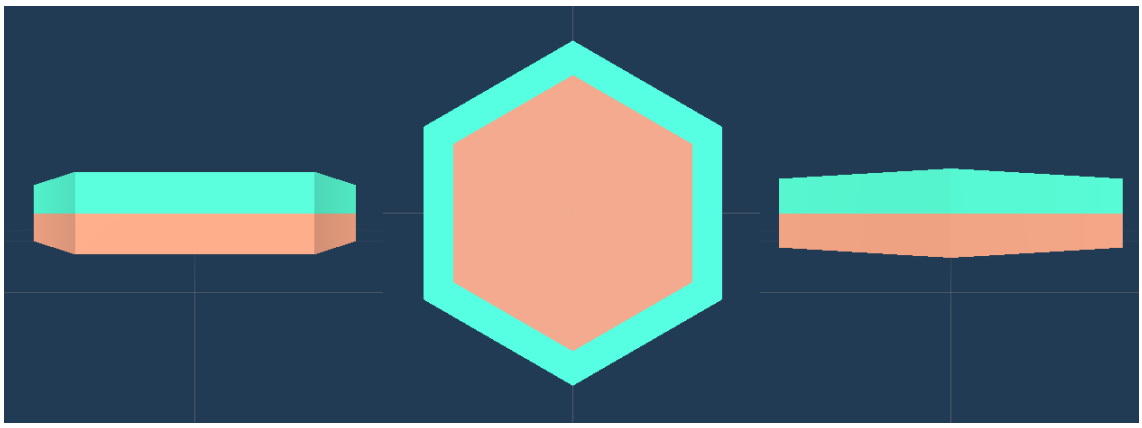

Ilustración 5 Casilla fertile (fertile)

[17] Debido al formato de la web se ha tenido que pasar previamente por un conversor que puede descargarse gratuitamente desde aquí: https://www.autodesk.com/developer-network/platform-technologies/fbx-converter-archives

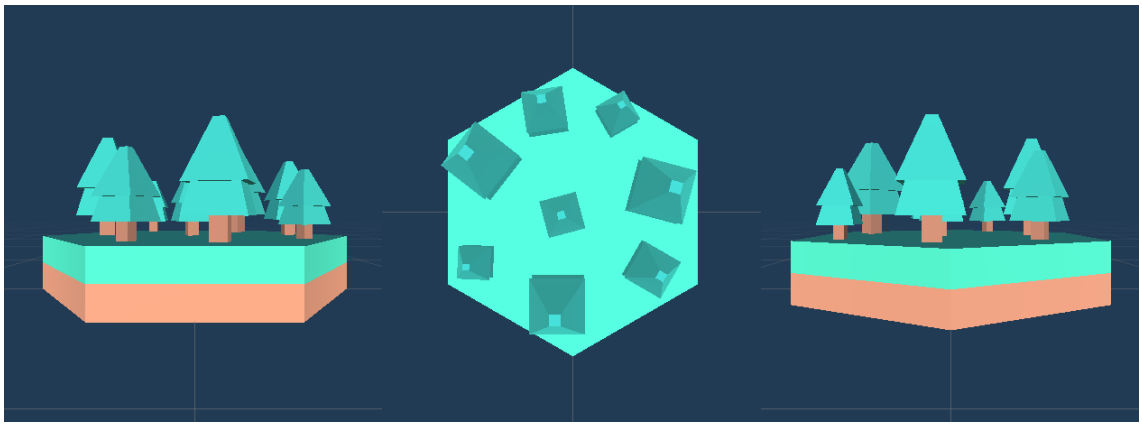

Ilustración 6 Casilla bosque (forest)

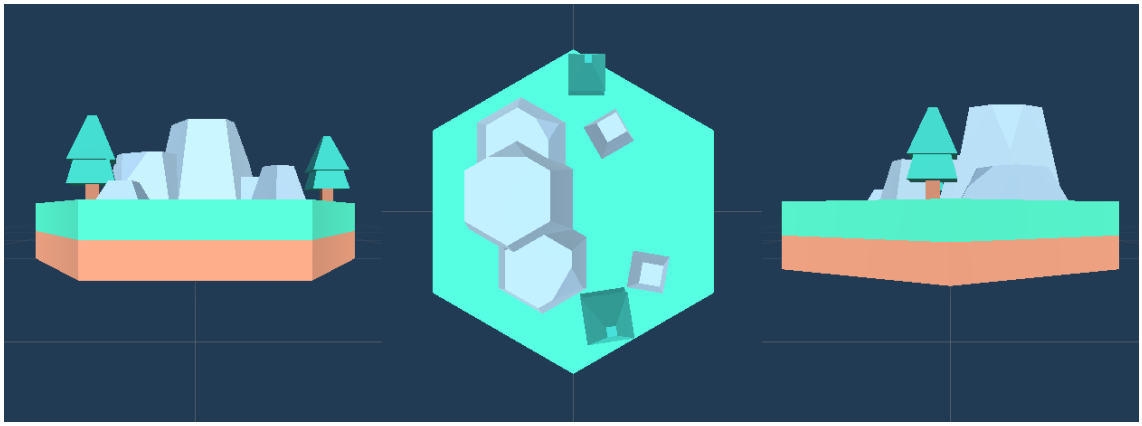

Ilustración 7 Casilla montaña baja (low mountain)

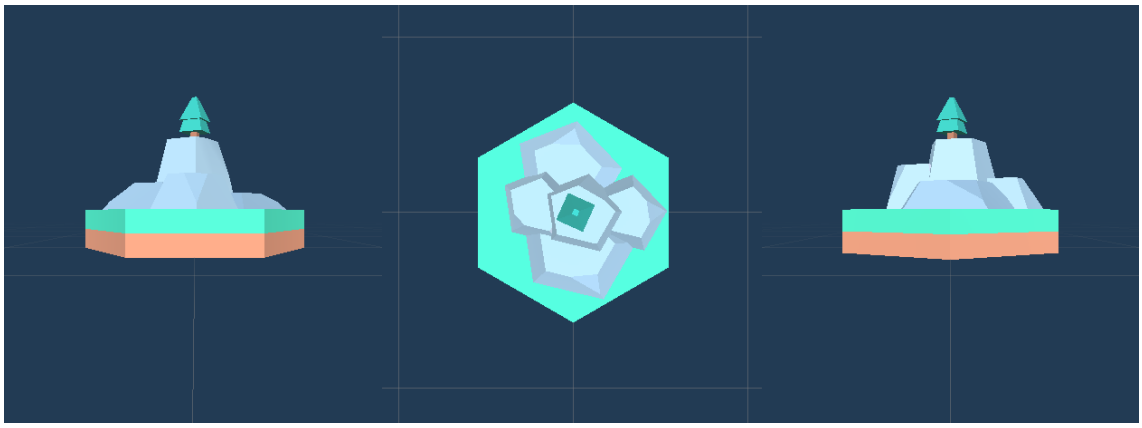

Ilustración 8 Casilla montaña (mountain)

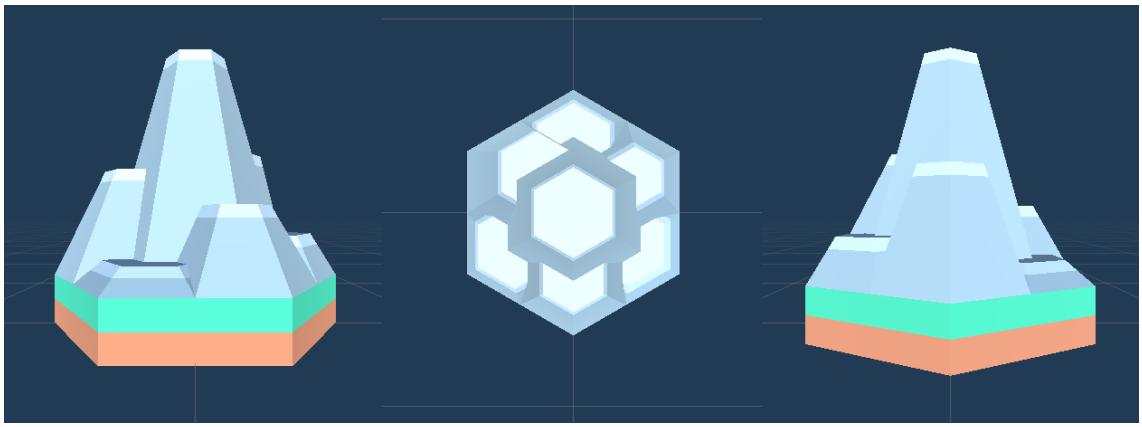

Ilustración 9 Casilla montaña alta (high mountain)

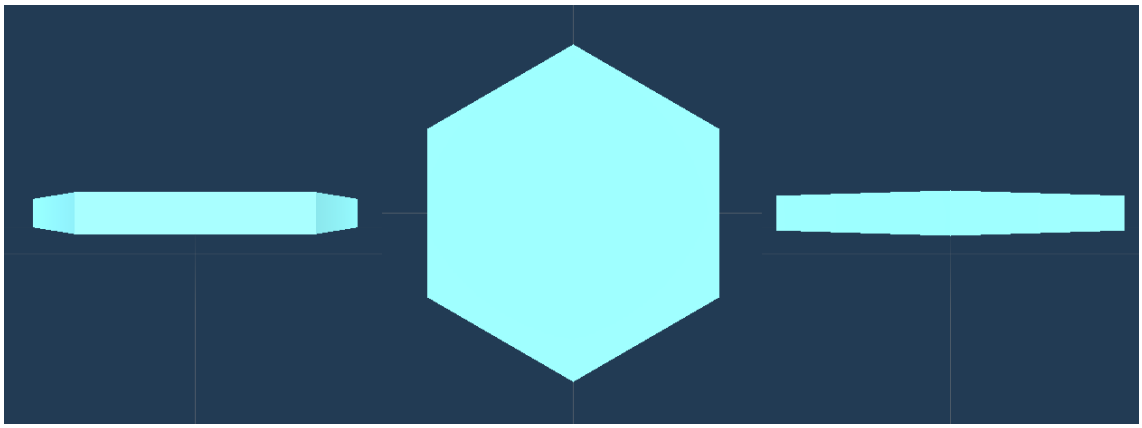

Ilustración 10 Casilla agua (water)

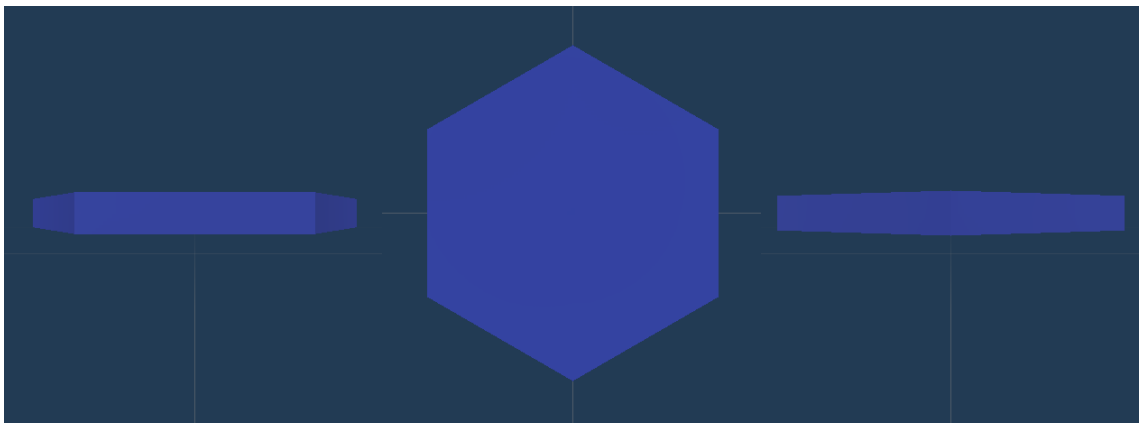

Ilustración 11 Casilla agua profunda o mar (deep water)

4.4.2 Edificios

La creación de los edificios se ha realizado siguiendo el mismo proceso que el de las casillas, partiendo de los mismos dos conjuntos de *assets* obtenidos desde la web se ha eliminado la base usando el *software blender* y se ha añadido otros elementos. Aunque es una de las tareas que más tiempo ha llevado, después del mapa, no es algo que se puede explicar demasiado ya que es mas ir modificando, ya se añadiendo elementos de otros edificios, borrando, o modificando los modelos en sí, hasta obtener los resultados deseados.

Se mostrará el listado de todos los edificios de la misma forma que se ha hecho con las casillas y solo el de nivel 1, para evitar sobrecargar demasiado el documento.

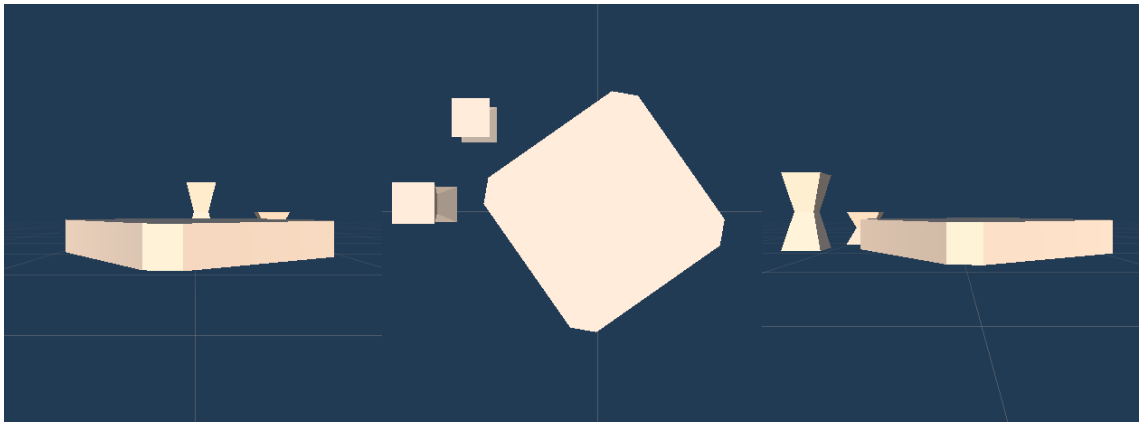

Ilustración 12 Edificio granja (farm)

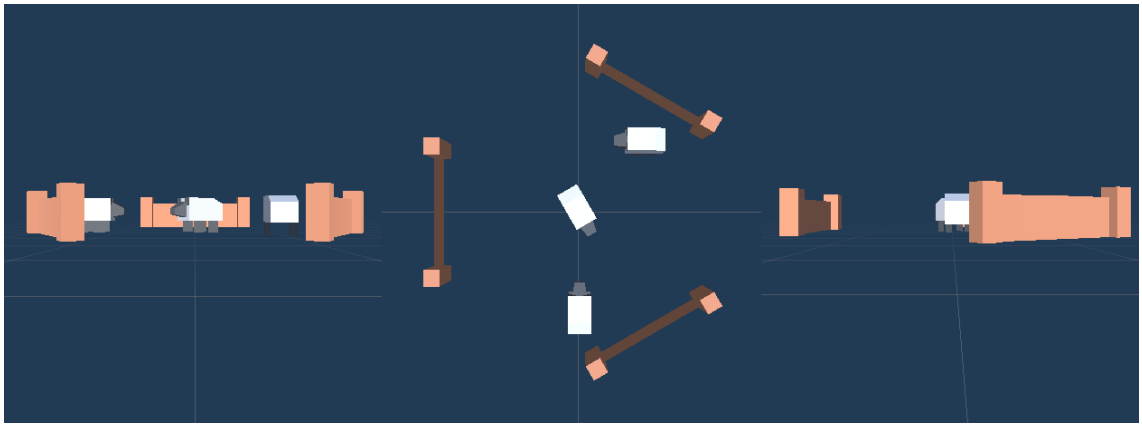

Ilustración 13 Edificio corrall (corral)

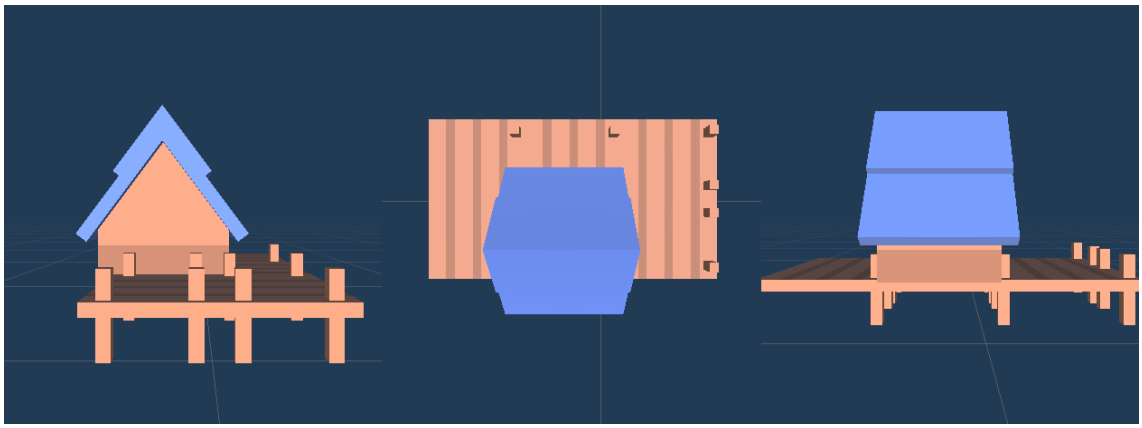

Ilustración 14 Casa de pescadores (fisher hutt)

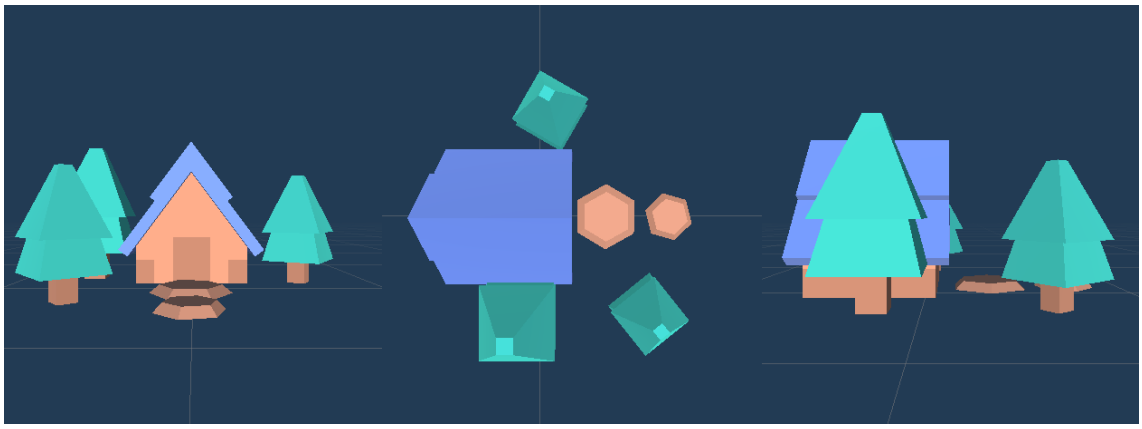

Ilustración 15 Edificio casa (house)

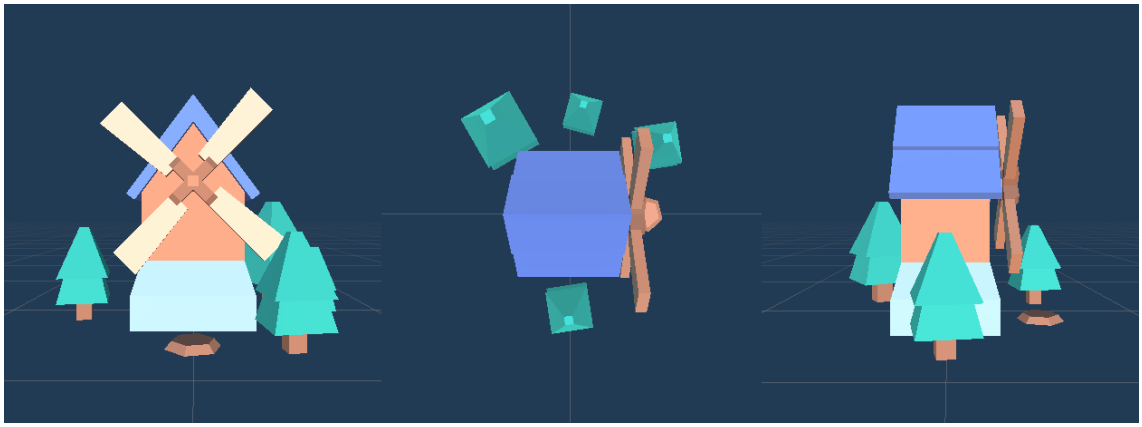

Ilustración 16 Edificio molino (mill)

Ilustración 17 Edificio mina (mine)

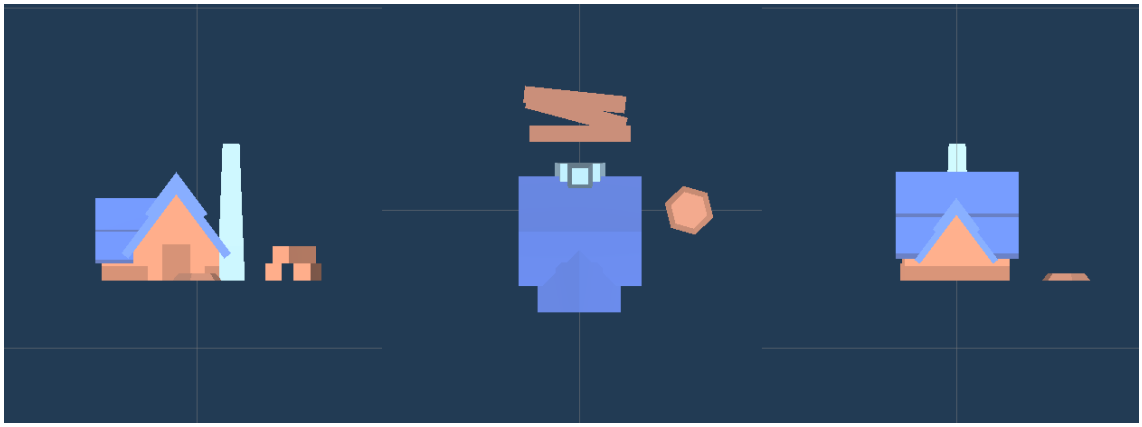

Ilustración 18 Edificio serreria (sawmill)

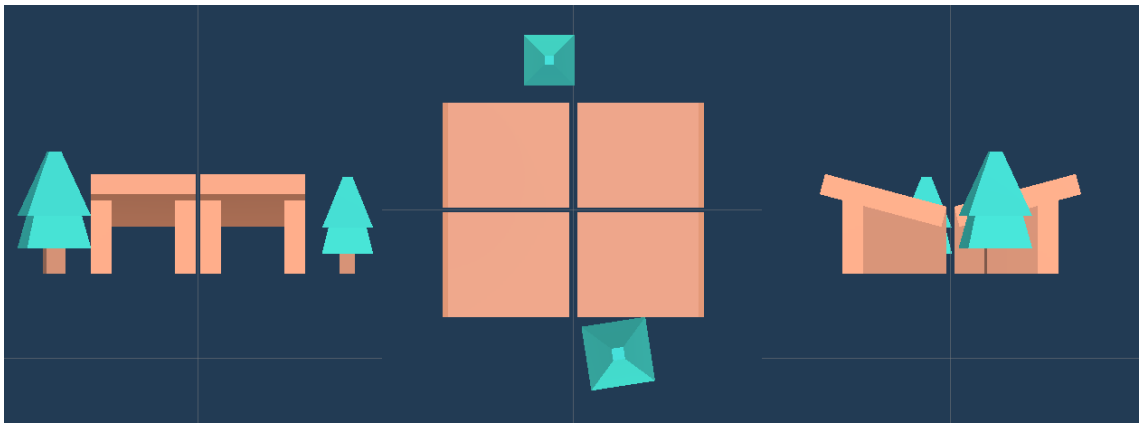

Ilustración 19 Edificio almacén (storage)

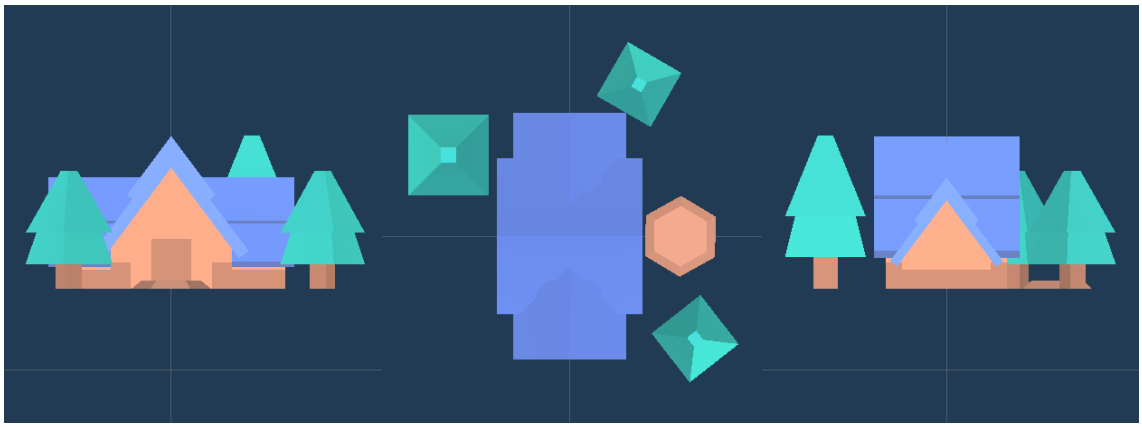

Ilustración 20 Edificio ayuntamiento (townhall)

4.4.3 Interfaz de usuario

La interfaz de usuario reutiliza varios componentes básicos que, al unirlos forman distintos elementos como pueden ser los botones, textos, títulos, paneles…

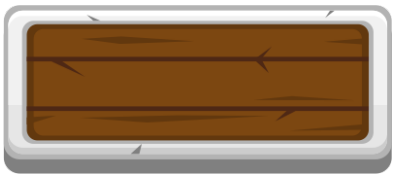

Ilustración 21 Botón UI

Este por ejemplo es el botón básico de los menús. Solamente se tiene que añadir un texto mediante *Unity* y esta completo y listo para su uso.

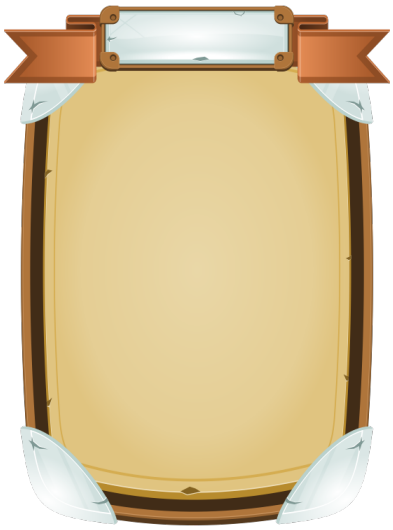

Ilustración 22 Menu de la UI

Este fondo para el menú se usa tanto dentro de la partida como fuera. Se le puede añadir un texto corto en la parte superior, así como los botones o texto que sean necesarios, dando total libertad al crear nuevos elementos.

Ilustración 23 Combinación de 3 tipos distintos de menú

En este ultimo ejemplo podemos ver como, a partir de los mismos elementos, se crean 3 menús que juegan un papel distinto en diferentes puntos de la partida. Simplemente se cambian los textos de los botones, del titulo, o se cambia el contenido por algo totalmente distintos. Esto se va a utilizar a lo largo del videojuego, teniendo un numero bajo de elementos básicos, pero que se junta de distinta forma para poder realizar diferentes paneles y composiciones que permiten al jugador entender, moverse, actuar y, en esencia, jugar a nuestro videojuego.

Ilustración 24 Imagen contenedor de indicador recursos

Ilustración 25 *Background* recurrente

Ilustración 26 Botón apertura menú *In-Game*

Ilustración 27 Botón de construcción de edificios y titulo

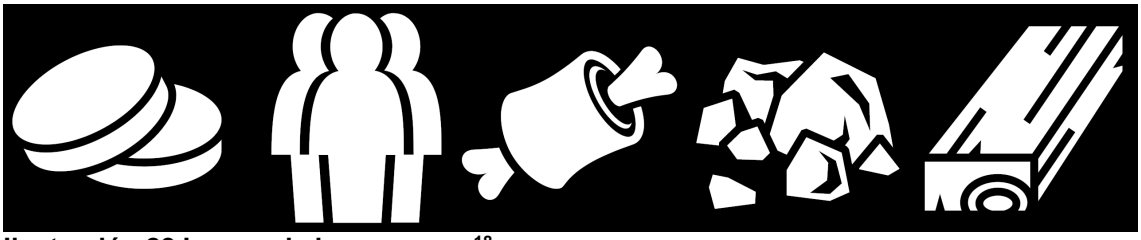

Ilustración 28 Iconos de los recursos[18]

Con solo el uso de esos 5 elementos se ha acabado generando toda la interfaz de usuario. Por ejemplo, el *sprite de la* Ilustración 27 Botón de construcción de edificios y titulo, se usa para los botones de construcción de edificios y también para los títulos de todos los paneles. El *sprite* de la Ilustración 25 *Background* recurrente se usa como *background* de paneles, *pop overs,* barra de menú… Podemos ver unos ejemplos ahora de todos los elementos y como se reutilizan.

Ilustración 29 Menu superior con elementos y recursos

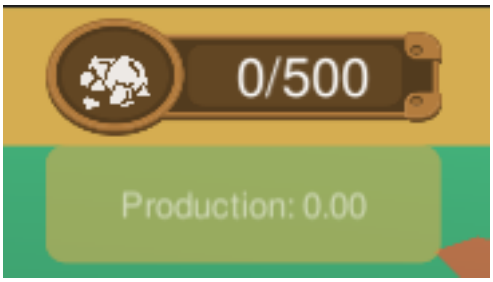

Ilustración 30 Recursos popOver para mas información

---

[18] Se pueden obtener en https://game-icons.net/

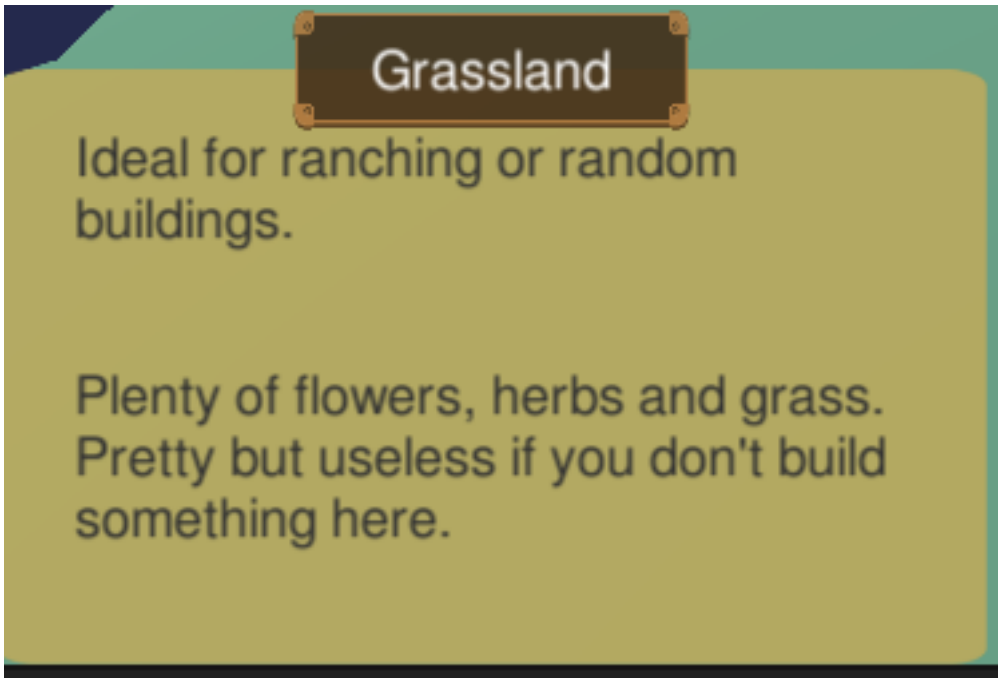

Ilustración 31 Panel con información casilla

Ilustración 32 Panel información edificios

Ilustración 33 Panel información edificio subir nivel

Todos estos *assets* se han descargado desde https://opengameart.org/.

4.4.4 Sonido

En *Shata* hay un pequeño sonido, que se usa cuando el usuario interactúa con no de los múltiples botones que hay en los menús, tanto dentro de la partida como en el principal. Este sonido se ha descargado también de https://opengameart.org/.

4.5 Arquitectura del juego[19]

Dentro de un proyecto de *Unity,* la arquitectura está bastante sujeta a lo que el motor gráfico nos impone. La libertad que existe en muchos otros proyectos, como por ejemplo el desarrollo web, donde existen diversas estrategias para abordar el problema a realizar, está limitada en este caso. No es algo único de *Unity* o de cualquier otro motor gráfico, siempre que se trabaje en cualquier *framework,* se debe seguir su arquitectura. Si bien es verdad que en el desarrollo web hay mas libertad, esta desaparece al usar uno de los *frameworks* como *Symfony* o *Laravel.*

La intención de *Unity* es que los proyectos se organicen, no orientados a objetos como se hace en la mayoría de los demás proyectos de *software,* sino orientados a componentes. La idea de los componentes es solucionar un pequeño problema, dividiendo el total en otros más pequeños [2].

[19] Aunque me gustaría explicar en su totalidad como funciona no es el objetivo de este documento. Si se desea conocer mas, puedes acceder al repositorio y verlo,.

Hasta que he realizado este proyecto, la forma de trabajar era muy parecido siempre. Dividia las entidades que componian mis escenas en objetos pequeños (con uno o mas componentes). Por encima de todos y cada uno de los componentes u objetos creaba uno, acesible desde todos los lados, estático y *singleton*[20] [3]. Este era el encargado de asegurar que el flujo del juego funcionaba correctamente y, ademas, obtenia información que podía ser útil para los demas componentes. De esta forma es como emepecé a programar *Shata,* siguiendo una estructura parecida a la siguiente.

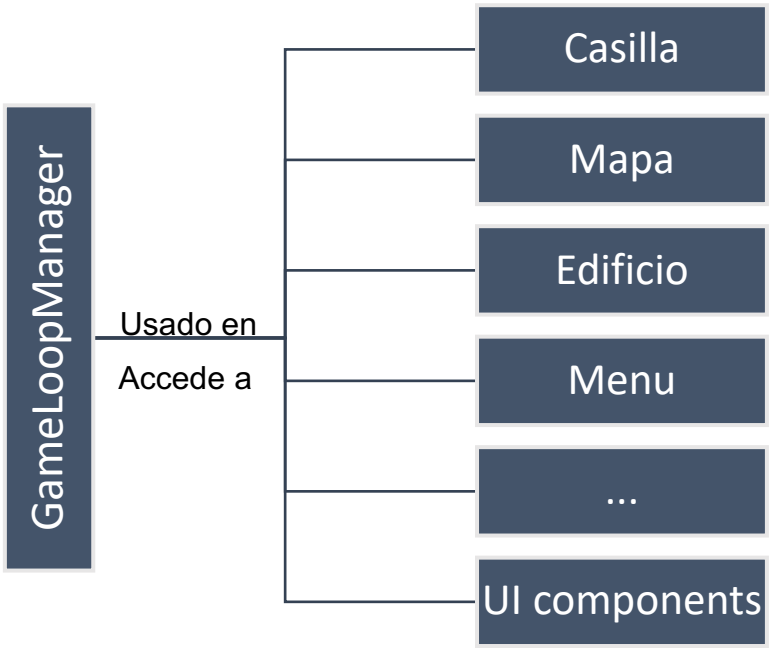

Ilustración 34 GameManager arquitectura primeriza

Para entender como funciona, pondremos un ejemplo sencillo. Imaginemos que el usuario selecciona una casilla. Al detectar que ha sido seleccionada, esta avisaría al *GameLoopManager* y, a su vez, este al *UI Component* encargado de mostrar la información correspondiente. Otra opción es ignorar el *GameLoopManager* y que el botón avise directamente al *UI Component*. Si se cambia el *UI component* o se añade otro, se debe modificar la casilla o el *GameLoopManager.*

El *GameLoopManager* tiene referencias a todos los componentes y esto es un punto negativo a la hora de estructurar un videojuego por las siguientes razones [3].

Un alto porcentaje de las veces que modificas un componente, debes también modificar el *GameLoopManager* y, probablemente, todos los componentes posteriores que se veían afectados por el componente modificado.

[20] Estrategia en la cual una clase es accesible desde cualquier parte del código y, además, se asegura que solo puede existir una.

Necesitas que todos los componentes estén presentes siempre, ya que si uno de ellos falta no se puede ejecutar el resto del flujo. Añadir o borrar componentes supone de nuevo volver a revisar y testear todo el flujo.

No puedes testear de forma unitaria cada uno de ellos, ya que en todo momento necesitas que se ejecute de una forma concreta.

Hay varias desventajas mas, como la dificultad al programar y entender el código, o la dependencia entre equipos que, al ser un equipo de una sola persona, se reduce, pero no deja de ser una mala practica.

Intentando eliminar estos problemas y hacer un código un poco mas mantenible, legible y, por que no, bonito, se ha seguido esta otra estructura que, mirando el esquema parece exactamente la misma pero la idea es muy distinta:

Ilustración 35 Arquitectura Shata [4]

La idea básica es la misma de siempre, cada objeto esta divido, separado completamente de los demás. Pero en este caso, los componentes no tienen ninguna referencia directa al *GameLoopManager,* tampoco saben de los otros componentes en ninguna medida, excepto todo lo que necesita saber el mismo para funcionar.

Utilizando el ejemplo de antes, cuando el usuario clica sobre la casilla, esta simplemente levanta un *GameEvent.* Todos los demás componentes que dependen de las casillas estarán encolados como escuchadores de los eventos y, al lanzarse, reaccionarán. La principal ventaja de esto es que, si ya no necesitamos el *UI Component* o se necesita otro de nuevo, no se debe tocar ningún otro elemento excepto el que se quiere modificar.

De forma parecida, los *Scriptable Objects* son una especie de variables globales que simplemente contienen información accesible por todos los componentes.

Por ejemplo, los recursos del jugador se encuentran en un *Scriptable Object* y desde cualquier parte del juego se puede acceder. Lo bueno de estos elementos es que el propio *Scriptable Object* no sabe ni necesita saber quien ni cuando accederán a él.

4.5.1 Mapa

Como ya se ha comentado varias veces, el mapa es el elemento en el cual se ha invertido más tiempo. No se quería tener un mapa pre generado y que el jugador siempre tuviese que jugar con el mismo, sino que se intentaba buscar una forma de crear un mapa totalmente aleatorio y único que permitiese al juego ser diferente en cada partida.

La estructura del mapa se divide de la siguiente forma:

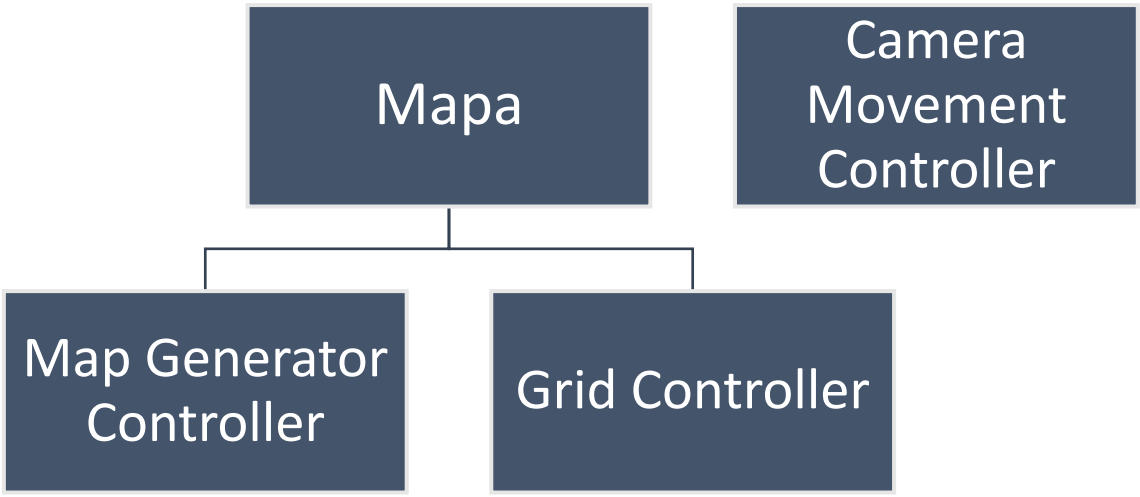

Ilustración 36 Arquitectura mapa

He querido poner este sencillo esquema ya que creo que se explica por si solo y no hace falta insistir más en su funcionamiento:

- Map Generator Controller: Genera el mapa de forma aleatoria. Una vez generado, este componente no es necesario. Para liberar memoria se podría incluso destruir el script
- Grid Controller: Se encarga de detectar el input del usuario, para poder reaccionar al click y seleccionar la casilla. Esto se realiza mediante la técnica de *Raycast*[21]. Lanza el evento correspondiente para que los demás componentes sepan que el usario ha seleccionado una casilla y cual de ellas.

[21] "Se lanza" un rayo desde la posición del rato en línea recta, y cuando choca con un objeto, este se devuelve, para comprobar si es una casilla o no realmente

- Camera Movement Controller: Se encarga del movimiento y zoom de la cámara. No esta realmente asociado al mapa, pero es una parte importante del mismo. El grid controller podría estar acoplado en este componente, pero para mejor entendimiento se ha decidido dejar separado.

4.5.1.1 Generación del mapa

Hemos mencionado varias veces que este aspecto es uno de los más importantes del videojuego así que se merece un pequeño apartado explicando por encima como se ha desarrollado, sin entrar a fondo en los detalles.

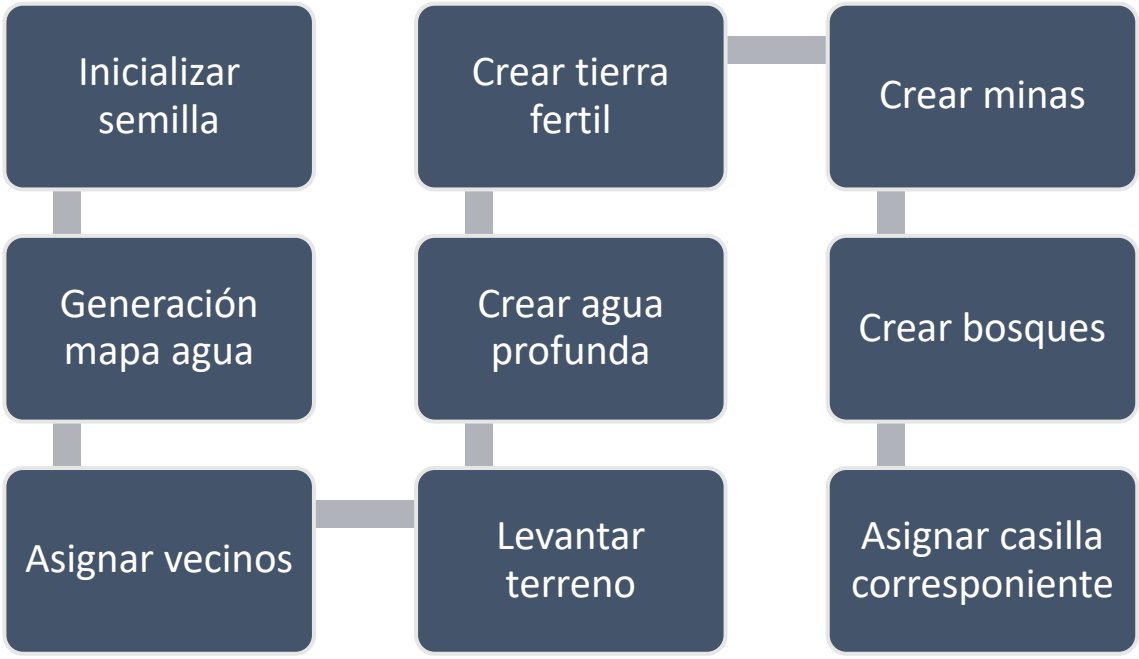

Ilustración 37 Esquema generación mapa

Este es el orden que se sigue para la completa generación del mapa:

Inicialización semilla

Se comprueba si el usuario ha establecido una semilla o si hay una por defecto. Si es así, se usará esta semilla como inicialización de la secuencia *psuedoaleatoria*. Si no existe, se generará una aleatoria que el usuario podría usar luego.

Si no se cambia ninguno de los otros parámetros, la generación de un mapa con una semilla concreta dará como resultado siempre el mismo. Esto se debe a que no existe un algoritmo de aleatoriedad como tal, sino que el algoritmo es una secuencia de números y cada vez que se pide un número aleatorio se devuelve el siguiente de esta cadena. De esta forma, estableciendo el inicio de dicha cadena, podemos obtener siempre los mismos números, provocando que se produzca siempre el mismo mapa.

Generación mapa de agua

Se inicializa todo el mapa con la casilla agua, sin tierra alguna. Se ha decido que esta es la casilla por defecto, pero perfectamente se podría haber realizado de otra forma.

Como veremos en los siguientes pasos, partiendo de una gran masa de agua, generamos el terreno, creando islas o continentes, según los parámetros de entrada.

Asignar vecinos

Se asigna a cada una de las casillas sus vecinos. Veremos que este es un punto clave para luego crear el terreno de una forma mucho más "realista" y consistente.

Este paso se realizó por primera vez una vez todo el mapa de agua estaba generado, provocando que se necesitasen un algoritmo de orden *2*N*, donde N es el número de casillas. Se decidió implementar al mismo tiempo que se generaban las casillas, haciendo que algoritmo pasase a ser de orden *N*.

Esto suponía un problema y es que no todos los vecinos de una casilla estaban generados todavía. Por eso, se ideo una forma que al pasar por una casilla asignase los vecinos ya generados y, a su vez, se asignase a si mismo como vecino de estos. Imaginase que en una fila de *1xN* se generasen las casillas de derecha a izquierda. Al pasar por la casilla numero 3, se le asignaría como vecino la casilla número 2 y, a la número 2, como vecino la casilla número 3, pero no la número 4 por que aún no esta generada. Al generar dicha casilla, se asignaría la 3 como vecino de la 4 y, obviamente, la 4 como de la 3. Esto se aplica en dimensiones *NxM* también.

Levantar terreno

Como ya se ha mencionado, partimos de un mundo completamente formado por agua y necesitamos generar la tierra y, para mas semejanza con un mundo real, las montañas. La mejor forma para similar esto es con el proceso de levantar terrenos.

Tenemos distintas alturas asignadas, siendo 0 el agua y 1 o mayor que 1, el nivel de altura sobre el nivel del mar. De esta forma, se escoge una casilla aleatoria del mapa y se incrementa en 1 el nivel sobre el mar[22]. Después, se seleccionan todos sus vecinos y se incrementa también su altura. De forma recursiva, hasta un máximo de veces establecido, se seleccionan de nuevo los vecinos de estos últimos, ignorando los ya visitados y se incrementa la altura, creando una isla en el mar de altura 1 que, posteriormente puede crecer si *colisiona* con otra isla generada de la misma forma.

---

[22] Posteriormente se decidió poner un límite de altura máxima.

Este proceso de selección de una casilla y sus vecinos se repite *N* veces, hasta que se alcanza el máximo de tierra permitido/parametrizado en el mapa.

Si se selecciona una casilla que ya tiene altura 1, se incrementará a 2, creando montañas. Esto provoca que las montañas suelan estar en el centro de un conjunto de tierra, y no pegadas al mar, lo que le da realismo.

Crear agua profunda

El mundo creado ya tiene una forma bastante bonita, con sus mares, tierra y montañas, sin embargo, en mapas donde el agua ocupa casi todo el mundo se ve muy vacío. Por eso mismo se decidió crear las aguas profundas que, aunque no afectan al *gameplay* si que dan un poco más de realismo y variedad.

Todas las casillas que estén a mas de X distancia de una casilla de tierra, se convierten en agua profunda, decrementando a -1 el nivel de altura. Esto creaba un contorno muy marcado, con un resultado bastante feo. Por eso se decidio cambia el valor de X a un número aleatorio entre 3 y 5, creando unos océanos mucho más realistas y bonitos.

Crear tierra fértil

Empezamos a crear los recursos que se necesitan para poder evolucionar en el juego. Se seleccionar un número de casillas de tipo tierra, es decir, de elevación 1 y se convierten en tierra fértil.

Crear minas

Se usa exactamente el mismo método que para crear la tierra fértil.

Crear bosques

De nuevo, se usa la misma técnica que para los dos recursos anteriores excepto que esta vez, no solo se convierte la casilla seleccionada a bosque, sino también alguno de sus vecinos, siguiendo una “copia” del proceso de levantar terreno pero a menor escala, mucho más sencillo y con un máximo de veces mucho menor.

Asignar casilla correspondiente.

Durante todo este tiempo, se ha mantenido un *array* de casillas que contiene el tipo de cada una de ellas, un identificador, pero realmente, no se ha generado ninguna casilla de tipo agua, tierra o similar. Esto se debe principalmente a dos razones.

La primera es que puede cambiar constantemente el tipo de casilla, así que es mucho mas eficiente mantener un identificador y no una clase más compleja que necesitamos instanciar cada vez.

Por otra parte, el mapa se crea en una escena, pero se mostrará y se crearán los objetos del juego en la siguiente. Por eso se ha creado un *ScriptableObject* que contiene esta información, para pasar el mapa de una escena a otra. Una vez estemos en la escena correcta, se transformaran estos identificadores en las casillas reales.

4.5.1.2 Parámetros

En el punto anterior se ha explicado, de forma muy resumida, el proceso de generación del mapa. Sin embargo, algo muy importante a destacar es que es completamente parametrizable.

- Porcentaje de tierra total
- Porcentaje de tierra fértil
- Porcentaje de minas
- Porcentaje de bosques
- Nivel de montaña
- Tamaño de mapa
- Tamaño de las islas
- Aletoriedad en selección de vecinos
- …

4.5.2 Casillas

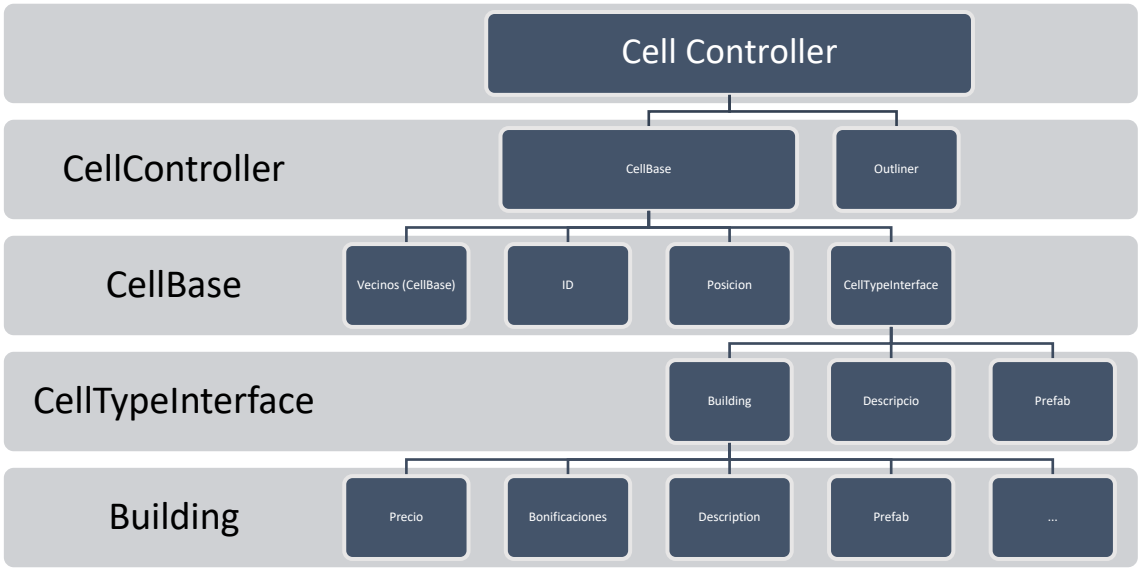

Ilustración 38 Arquitectura casilla[23]

[23] Esta pendiente de ser modificado un poco, para minimizar las clases necesarias en la arquitectura

Esta arquitectura es la que se ha decido usar para las casillas. Expliquemos uno a uno que es cada apartado.

- CellController: Sirve como punto de entrada de *Unity.* Hereda de *MonoBehaviour.* Reacciona y eleva os eventos necesarios. Tiene el controlador del outliner. Es un script puramente Unity related. Tiene el CellBase.
- CellBase: Contiene los vecinos de la casilla, la ID y la posición. También tiene el tipo de la casilla (tierra, montaña…). No tiene lógica alguna.
- CellTypeInterface: Es la definición de la casilla en si, que tipo es, la descripción, edificios que se pueden edificar… Contiene el edificio que se ha construido
- Building: Definición del edificio que se ha construido.

4.5.4 Tipo de casillas

Definir una nueva casilla es realmente fácil. Creas una clase nueva que herede de la clase abstracta *CellTypeInterface*. Necesitas declarar ciertos elementos como el nombre, descripción, edificios permitidos…. Por ultimo, necesitas una *prefab* con el modelo de la casilla y ya estaría. Tan fácil como esto.

4.5.5 Tipos de edificio

De la misma forma que se define una casilla, para crear un edificio es exactamente igual. Se necesita heredar de la casilla adecuada y definir las propiedades básicas, siendo el *prefab* una de ellas de nuevo.

Se ha trabajado bastante para que añadir edificios y casillas sea fácil, ya que, aunque la entrega tenga un numero muy pequeño de ambos, se pretende ampliar posteriormente.

4.5.6 Almacén y recursos

Se ha creado una clase que maneja todos los recursos de los que disponen el jugador y este está definido como *ScriptableObject,* es decir, accesible para todos los componentes como lectura. Se ha separado completamente la parte gráfica de la de gestión de recursos, de esta forma, seguimos trabajando con componentes que solo conocen aquello que necesitan y no saben nada de que componentes necesitan de él.

Para gestionar los recursos, es decir, el precio, lo que produce los edificios, la cantidad máxima… se ha creado una especie de modificadores que se guardan en la clase almacén. Los datos de estos modificadores son el recurso a modificar, la cantidad y el tipo de modificar (que puede ser producción, cantidad máxima o cantidad plana). De esta forma se puede recuperar en todo momento la cantidad actual y el por que de estos resultados.

4.5.7 Eventos

Otra de las estrategias con las que nos invita a trabajar *Unity* es con los eventos [2]. Esta forma de trabajo consiste en crear una función o procedimiento, que

espere a ser ejecutado cuando cierta acción, programada, suceda. Es la forma por defecto de funcionar de los botones. También se usa juntamente con las animaciones. Cuando cierto *frame* de la animación se ejecute, este lanzará el evento, que será recogido por el objeto asignado y se ejecutará.

Tan potente es que, como hemos podido ver, se ha utilizado fuera de los elementos para los que realmente estaba pensado este tipo de elementos y yo he decidido probarlos e implementarlos en este juego, con un resultado muy bueno.

4.6 Mecánicas implementadas

A medio camino entre diseño de niveles y el diseño técnico, en este apartado voy a explicar muy por encima las diversas mecánicas que se han implementado, enumerándolas individualmente.

Recursos

Este es el punto central del juego. Generar y gestionar lo recursos es lo que permite al jugador evolucionar.

Hay un total de 5 recursos: madera, comida, piedra, oro y población. Todos funcionan de forma muy similar. Los edificios necesitan alguno de estos recursos para ser construidos o mejorados y, a su vez, los edificios proporcionan algún beneficio sobre los recursos. Además de la producción o consumo de los recursos, hay un máximo almacenable, que puede ser ampliado con los edificios correspondientes.

La piedra y la madera se usa exclusivamente como se ha comentado en el párrafo anterior, pero los otros 3 tienes un poco más de mecánicas. Por ejemplo, la comida es consumida constantemente de forma directamente proporcional a los habitantes que se tiene en el pueblo. La población afecta también de forma directa a la producción del oro. A su vez, el oro se usa para el mantenimiento de los edificios.

Mapa

Se ha dedicado un apartado exclusivo a esta mecánica, ya que es también un punto elemental del juego que se puede consultar para ver la implementación de esta mecánica (4.5.1 Mapa).

El mapa es un conjunto de diferentes casillas que representan agua, montañas, bosques, tierra… cada una con sus limitaciones y ventajas, que modifican el *gameplay*.

Casillas

Las casillas y los edificios que se construyen en ellas son también el punto principal del videojuego. Las casillas permiten o impiden la construcción de un tipo de edificios. Por ejemplo, el aserradero solo puede construirse en un bosque.

Edificios

Los edificios tienen un coste asociado. Algunos proporcionan alguno de los recursos, otros permiten incrementar la capacidad máxima almacenable. Realmente es el edificio que tiene la limitación de las casillas en las cuales puede ser construido.

Nivel edificios

Los edificios construidos se pueden mejorar un máximo de 2 veces, proporcionando más recursos, sin tener que buscar una casilla extra.

Destrucción de edificios

Los edificios se pueden destruir, eliminando los beneficios que nos proporcionaban y permitiendo la construcción de otro edificio nuevo en la casilla.

Limitaciones de edificios

Se han implementado varias condiciones a la hora de construir. La más clara y que ya se ha comentado muchas veces, es la de aquellos edificios que solo se pueden construir en una casilla especial, pero hay otras.

Solo se puede construir un ayuntamiento y este debe ser siempre el primer edificio en ser construido, sin el no se permite construir otros. Los edificios no especiales tienen que construirse al lado de un edificio, no pueden construirse en medio de la nada. Las casas de pescadores deben estar adyacentes a la costa y al menos a 1 de distancia de otra casa de pescadores.

Movimiento del mapa

Se puede mover por el mapa con las teclas “A”, “S”, “D” y “W” y se puede hacer *zoom*. Se ha establecido un limite para ambos de los movimientos, impidiendo que el jugador deje de ver las casillas.

Selección de casillas

Las casillas se pueden seleccionar, mostrando un pequeño borde para distinguir la casillas actualmente en uso.

5. Diseño de niveles

En este apartado podemos ver como seria una partida en *Shata,* como si de una guía se tratase, para dar a conocer el juego y explicar al usuario como jugarlo, imaginando que es un jugador novel en el genero de *City-Building.*

Iniciamos el juego y nos encontramos con el típico menú, que nos da unas opciones básicas. Podemos jugar un poco y ver por ejemplo los créditos Ilustración 40 Menú de créditos, o darle al botón *Exit* para salir, pero dado que no que realmente queremos es jugar, le daremos al primer botón.

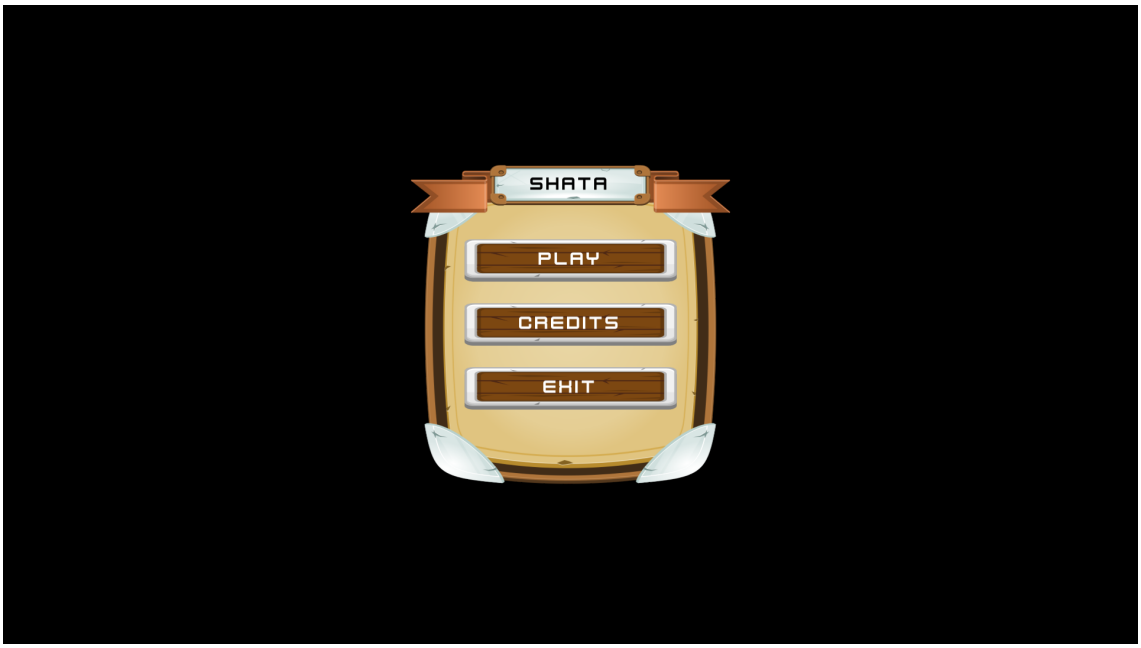

Ilustración 39 Pantalla de inicio

Ilustración 40 Menú de créditos

Nos encontramos con el mapa del videojuego. Que tendrá un aspecto parecido a esto.

Ilustración 41 Mapa de Shata al iniciar partida

Podemos movernos por el mundo con las teclas “A”, “S”, “D” y “W”. Buscamos un sitio que nos guste para construir nuestro ayuntamiento.

Actualmente no tenemos materiales, como podemos observar en la barra superior del juego. Oro, madera, comida, piedra y población, respectivamente, están a 0 de cantidad actual y a 0 de cantidad máxima que podemos almacenar. No debemos preocuparnos por eso, ya que el ayuntamiento no solo es gratis, sino que nos da materiales para empezar la partida.

Así pues, sin mas preámbulos, escogemos una casilla que nos guste para construir el ayuntamiento y, cuando aparezca en el menú inferior derecho, clicamos sobre *Townhall*.

Ilustración 42 Construcción del ayuntamiento

Podemos ver que esto nos proporciona automáticamente recursos. Ahora podemos construir alguno de los otros edificios que existen en nuestro juego.

Aunque el jugador puede hacer lo que quiera y construir otro edificio, aconsejamos construir el aserradero o *Sawmill*. Este edificio solo se puede construir sobre una casilla de bosque, así que elegimos una que nos parezca adecuada y de la misma forma que hemos construido el *Townhall,* construimos nuestro *Sawmill.*

Ilustración 43 Construcción del aserradero

Podemos ver que justo después de construirlo, empezamos a generar un poco de madera. Esto se puede ver si ponemos el ratón sobre el recurso en cuestión. Al hacerlo, un pequeño desplegable aparecerá diciéndonos la cantidad que

estamos produciendo ahora mismo. Esto es el sumatorio de todos los gastos y la producción.

Ilustración 44 Localización de la producción

El siguiente paso seria, por ejemplo, construir una casa, que permitiese que nuestro pequeño tuviese mas de 6 ciudadanos. Las casas pueden ser construidas en cualquier lugar así que seleccionamos una casilla cualquiera y procedemos a construir la casa, de la misma forma que hemos hecho con el resto de edificios, usando el panel inferior de la pantalla.

Ilustración 45 Construcción casa

Podemos construir ahora una granja, para alimentar a nuestros ciudadanos. Igual que pasaba con el aserradero, solo puede ser construido en su casilla asignada.

Ilustración 46 Construcción granja

Ilustración 47 Post construcción granja

Podemos ver en esta ultima ilustración como casi no tenemos madera y, además, la producción de la misma es bajísima. Tal vez es una buena idea construir otro aserradero.

Ilustración 48 Construcción segundo aserradero

Esperamos a tener la madera necesaria y construimos el aserradero. Podemos ver como la producción de madera se incrementa, teniendo una base ya para nuestro pueblo.

5.1 Testeo y balanceo

El diseño de los niveles no es algo que se haga sobre papel y sea fácil llevar al videojuego, sino que son fruto de muchos intentos. Todos y cada uno de los elementos se ha debido probar cautelosamente, para asegurar que no se producen puntos muertos independientemente de lo que haga el jugador.

Mientras que la palabra *testeo* se refiere más a la detección de *bugs* y la solución de los mismos, la palabra balanceo se refiere a conseguir un punto intermedio en el cual los niveles no sean muy fáciles de superar ni tampoco imposibles.

El testeo y el balanceo se consigue simplemente jugando tantas veces como sea posible, intentando todas las combinaciones posibles, incluso si no tienen sentido para pasarse el nivel o jugar al juego de forma “correcta y lógica”. Algunos de los ejemplos más comunes consisten se basan en exactamente esto, cosas que los desarrolladores no habían pensado que los jugadores hiciesen y, al hacerlo, se dan situaciones imprevistas.

Como ya se ha dicho, la única forma de detectarlos es jugando. Para ello, no solo yo, el desarrollador, he jugado mi videojuego, si no que se ha presentado a un número de jugadores que han funcionado como *testers*. En un proyecto de verdad, un gran número de personas llevarían a cabo esta tarea, durante varios meses en los cuales se irán arreglando todo lo que se vaya reportando.

En este caso, aparte de mí, se ha pedido ayuda a dos personas para que testeasen el juego, siendo uno de ellos un jugador habitual, y otro un jugador ocasional, casi sin experiencia alguna.

5.1.1 Testo

Algunos ejemplos de *bugs* reportados que impedía el correcto funcionamiento del juego son los siguientes:

- El mapa nunca acaba de generarse con según que configuración.
- Algunos edificios no se generaba bien el modelo, haciendo que visualmente se viese feo.
- Mas de una vez los modifiers de los elementos no eran los adecuados. No se considera balanceo, ya que era un *bug*, al equivocarme de las ventajas que estos daban.

5.1.2 Balanceo

Algunos ejemplos que hacían el juego desbalanceado, ya sea para bien o para mal, son:

- El ayuntamiento daba muy pocos recursos.
- Poca cantidad de recursos en el mapa. Se solucionó parametrizando, permitiendo así modificar la dificultad del juego.
- Precios demasiado altos.
- Demasiada producción de madera.

6. Manual de usuario

6.1 Controles

Los controles en este juego son realmente sencillos, ya que la mayoría de las acciones se realizan simplemente desde un menú, haciendo *click* a diversos elementos y botones de la interfaz del usuario.

Con el teclado, los controles son los siguientes: **"A"**, "**W**", "**S**" y "**D**" para moverse por el mapa hacia la izquierda, arriba, abajo y a la derecha, respectivamente. Como en la inmensa mayoría de los juegos, la **rueda del ratón** se usa para realizar *zoom* acercándose y alejándose del mapa. Además, esto también provoca una pequeña rotación para dar la sensación de que la cámara se queda en el suelo. La velocidad de movimiento se mantiene mas o menos lineal, haciendo que esto no repercuta negativamente, si no
Por último, la tecla "**ESC**" sirve para pausar el nivel. Si ya esta pausado, sale del menú, reanudando el juego.

7. Conclusiones

7.1 Lecciones aprendidas

Como se ha repetido varias veces en la introducción, la intención principal de este trabajo es aplicar y poner en practica todo aquello que se ha aprendido a lo largo del master. No nos referimos solo a la parte técnica y al uso del motor de videojuego que se haya decidido usar para realizar el proyecto, sino todo aquello previo al desarrollo, y también durante el mismo, que es necesario para realizar un videojuego.

Establecer un documento de diseño del videojuego, donde se especifique todo aquello que se quiere realizar es el primer punto que hemos tenido que tratar en este trabajo. Aunque obviamente este documento se modificará mientras se desarrolla el videojuego, tenerlo como referencia es una gran ayuda para saber qué debemos hacer y de cuánto tiempo disponemos. Cuando el desarrollo del videojuego es de una sola persona, puede no parecer tan importante, debido que dicha persona sabe lo que él desea para realizar su videojuego. Ahora bien, si este juego lo desarrollan un grupo, por pequeño que sea, que todos los integrantes tengan el mismo objetivo y tengan claro la meta es un punto muy importante para que el juego evolucione correctamente. No es el único uso de este documento. Cuando el juego se presenta a una distribuidora o al público, este documento sirve como exposición para dar a conocer como es el juego y a quien está dirigido.

Debido a la importancia de este documento, esta es tal vez la lección más importante que se ha aprendido con el desarrollo del trabajo. Pero no es el único.

Diversos componentes de *Unity* se han desarrolla en este videojuego que no se habían puesto en práctica antes: El uso de *GameEvents* y *ScriptableObjects* y el uso de nuevas arquitecturas mas complejas son los ejemplos más claros de las lecciones aprendidas referidas en cuanto al uso de *Unity.*

7.2 Logros alcanzados

A principio de curso, se pretendía crear un videojuego de construcción de ciudades y, en gran parte, se ha conseguido realizar este objetivo. Se han creado diversas mecánicas que, juntándolas pueden crear un sinfín de oportunidades que no se han llegado a desarrollar por completo en el juego. Añadir nuevos edificios, casillas o incluso recursos es bastante sencillo ahora y el tiempo de programación es bastante pequeño, aunque si se debe dedicar tiempo al balanceo si se añaden estas opciones.

Por lo general, casi todos los objeticos que se plantearon se han podido alcanzar en la medida deseada, cumpliendo con aquellos que se había establecido en el documento inicial. Algunos se han añadido a posteriori, sin estar planificados en el inicio del proyecto, ya que se veía un futuro en ellos. Pero también hay algunos

que no se han podido completar. Todos estos ejemplos se comentarán en los siguientes dos puntos.

7.3 Seguimiento y análisis

El tiempo que se necesitaba para cada uno de los puntos que se querían implementar se pueden ver en el punto correspondiente el 1.4 Planificación del Trabajo. Cumplir al cien por cien con lo establecido no ha sido posible, aunque sí que se han podido realizar la mayoría de las tareas y de los puntos.

No cumplir con el *planning* no significa haber eliminado cosas, sino que se han tenido que modificar los tiempos establecidos.

La búsqueda de arte no acabo sobre el día 22/10, sino que se ha tenido que ir buscando nuevos elementos a lo largo del desarrollo del videojuego. Aunque los edificios estaban terminados, ciertos cambios o combinaciones de edificios provocaban que no me resultasen del todo atractivos.

Se ha invertido muchísimo más tiempo del que se planifico a la creación del mapa. Esto se debe principalmente a que al principio se pensaba crear un mapa estático, como si de un tutorial se tratase. Sin embargo, a medida que se desarrollaron las mecánicas, se decidió cambiar esta idea, acercándose a la idea de crear un mapa que se generase de forma distinta en cada partida y que además fuese parametrizable. No solo me pareció divertido de programar, sino que también me pareció un gran extra en el juego para futuras entregas del juego.

7.4 Mejoras

El juego tiene aun un sinfín de posibilidades que, tarde o temprano, se implementarán para darle más jugabilidad, haciendo que deje de ser una demo donde se presenta un juego y pase a ser un juego de verdad.

Algunas de ellas ya se han comentado a lo largo de la memoria, otras se planificaron, pero no se llegaron a hacer y otras simplemente no se tuvieron en cuenta ya que el tiempo de que se disponía no era lo suficiente como para que se implementasen.

- Mas recursos como por ejemplo hierro y oro.
- Añadir felicidad
- Añadir temperatura que afecta a la felicidad
- Edificios nuevos como por ejemplo la cervecería que proporciona felicidad o mina de oro y hierro.
- Tratado de los recursos para convertirlos en nuevos. Que la granja de trigo y esta pueda convertirse en pan, por ejemplo.
- Mejora en la generación de mapa, permitiendo la creación de un mapa mucho mas grande (necesidad de la pantalla de carga para indicar al jugador que se esta creando).

- Permitir guardar y cargar la partida.

7.5 Links y recursos

El código y la versión ejecutable del juego se puede descargar desde GitHub.

- https://github.com/oalonsobu/Shata

Por otra parte, el video que muestra como es el juego está en YouTube.

- https://youtu.be/d8wiku1IG24

El enlace de UnityConnect de mi perfil es

- https://connect.unity.com/u/oscar-alonso-bustos-1
- https://connect.unity.com/p/shata-3yu

8. Glosario

Asset: Cualquier elemento o recurso que se va a usar durante el desarrollo del videojuego. Puede ser imágenes, audio, materiales…
Background: Literalmente, fondo. En este caso, fondo de la escena del videojuego.
Bug: Errores en un software que provoca un comportamiento no esperado
Cocos2d: Framework para el desarrollo de videojuegos desarrollado en *Python*.
C++: Lenguaje de programación usado por *Unreal Engine*, entre otros.
C#: Lenguaje de programación usado por *Unity*, entre otros.
Demo: Programa informático, en este caso, un videojuego, que es una versión reducida del programa o videojuego completo. Se suele usar como evaluación antes de su compra total.
Engine: Literalmente, motor. Se usa como abreviación de *Game Engine.*
Frame: Fotograma, una imagen en concreto del conjunto que compone la animación o el videojuego.
Framework: Un marco de trabajo, conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar. [4, 5]
Game engine: Motor de videojuegos.
Game Maker Studio: Game engine orientado a personas con poco conocimiento o nociones de programaci-on.
Glitch: Error que no afecta a la jugabilidad o estabilidad del programa, por lo que no puede considerarse un bug. Lo mas común son los visuales, provocando situaciones raras e incluso graciosas.
Indie: De la palabra independiente. Persona o grupo que desarrolla un videojuego sin la inversión, ya sea monetaria o de marketing, de una distribuidora o de una organización con más poder adquisitivo o reputación.
Open Source: Programa cuyo código es accesible para ver, editar o redistribuir por cualquier persona.
Python: Lenguaje de programación.
Raycast: Técnica que consiste en lanzar un “rayo” para detectar las colisiones con el entorno. El rayo es simplemente una línea recta.
Script: Texto, escrito. En el ámbito de la informática, programa o secuencia de instrucciones que realizan una tarea.
Singleton: Técnica que asegura que una instancio u objeto solo existirá una vez. Útil cuando se necesita que un objeto controle a lo largo del sistema y que no se creen inconsistencias.
Software: Conjunto de información y de instrucciones que indican al ordenador como actuar. En este caso, nos referimos al videojuego
Sprite: Imagen.
Stackoverflow: Foro donde personas de cualquier parte del mundo realiza preguntas (en este caso técnicas) y otra gente le responde, creando debate hasta encontrar una solución.
Static: Indica que dicha función o variable no es única de cada instancia de este objeto, si no que se comparten entre ellos. De nuevo, es útil para compartir información entre todos los objetos de la misma clase.

Tester: Persona cuyo trabajo es jugar al juego, para poder reportar todos los errores que encuentre
TFM: Trabajo de Final de Master.
UI: Interfaz de usuario (del ingles *user interface*)
Unity: Motor de videojuego desarrollado por *Unity Technologies.*
Unreal Engine: Motor de videojuego desarrollo por *Epic Games*
Walkthrough: Tour o demostración, en este caso, del nivel. No tiene que ser exactamente una explicación de cómo hacerlo o superarlo, sino mostrar como es.
Zoom: Método para disminuir el ángulo de visión de un *frame*, permitiendo provocar la sensación de “acercarse” al objetivo del *zoom*

Bibliografía

[1] Wikipedia, «Wikipedia,» 4 Junio 2019. [En línea]. Available: https://en.wikipedia.org/wiki/List_of_Unreal_Engine_games. [Último acceso: 29 Noviembre 2020].

[2] R. T. Bonet, «Gamasutra,» 7 Marzo 2018. [En línea]. Available: https://www.gamasutra.com/blogs/RubenTorresBonet/20180703/316442/A_better_architecture_for_Unity_projects.php. [Último acceso: 2 Noviembre 2020].

[3] Unity, «Unity,» 13 Noviembre 2018. [En línea]. Available: http://wiki.unity3d.com/index.php/Singleton. [Último acceso: 2020 Octubre 18].

[4] Wikipedia, «Wikipeda,» 17 Mayo 2019. [En línea]. Available: https://es.wikipedia.org/wiki/Framework. [Último acceso: 1 Diciembre 2020].

[5] Unity, «Youtube,» 20 Octubre 2017. [En línea]. Available: https://www.youtube.com/watch?v=raQ3iHhE_Kk&feature=youtu.be. [Último acceso: 24 Octubre 2020].

[6] Unity3D, «Unity3D,» Diciembre 2018. [En línea]. Available: https://unity3d.com/how-to/architect-with-scriptable-objects. [Último acceso: 5 Noviembre 2020].