Universidad Politécnica
de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Prototipo de un Videojuego de Supervivencia

Autor: Francisco Fernández Barroso

Tutor: Guillermo Román Díez

Madrid, junio de 2022

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática
Título: Prototipo de un Videojuego de Supervivencia
Enero 2022

Autor: Francisco Fernández Barroso

Tutor:
Guillermo Román Díez
Lenguajes y Sistemas Informáticos e Ingeniería de Software
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

En este Proyecto se pretende crear el prototipo de un videojuego de supervivencia en dos dimensiones, utilizando el motor gráfico Unity y C# como lenguaje de programación. Se han implementado las principales mecánicas de este, con el objetivo de tener una experiencia similar a la final, pero a menor escala.

En el videojuego el jugador podrá plantar semillas y recolectar la cosecha, talar árboles, utilizar el sistema de inventario para almacenar los recursos que vaya obteniendo, y controlar un sistema de hambre y sed mediante la compraventa de varios objetos que le permitirán permanecer con vida.

Esta memoria pretende recolectar todo lo relacionado con el desarrollo de este trabajo. Se comenzará con una introducción, en la que se dará algo de contexto sobre la industria del videojuego, se explicarán las motivaciones detrás del proyecto, y se analizarán los objetivos que hubo previos al desarrollo y como se modificaron durante este. En los siguientes apartados se tratará todo lo relacionado con la implementación del videojuego, comenzando con las herramientas utilizadas, y pasando a continuación a explicar los procesos de análisis y diseño técnico. Tras esto se comentarán las conclusiones personales tras el desarrollo, así como los resultados obtenidos. A continuación, se proporcionará un manual de usuario para el correcto uso del videojuego. Para finalizar se analizarán los resultados obtenidos tras el proyecto y se hará un estudio del impacto personal, medioambiental, económico y cultural de este trabajo.

Abstract

In this project the objective is to create the prototype of a survival game in two dimensions, using Unity as the game engine and C# as the programming language. The main mechanics of the videogame will be implemented with the goal of attaining an experience like that of a finished game, but with a reduced scale.

In the videogame the player will be able to plant seeds and harvest crops, chop down trees, use the inventory system to store the resources they obtain, and manage a hunger and thirst system through the trading of various items which will allow them to stay alive.

This memory aims to gather everything related with the development of this work. It will begin with an introduction, in which some context about the videogame industry will be given, the motivations behind the project will be explained, and the objectives that existed prior to development and how they were modified during it will be analyzed. In the next section, everything related to the implementation of the videogame will be discussed, starting with the tools used, and then going on to explain the analysis and technical design processes. After this, the personal conclusions after the development will be talked about, as well as the results obtained. As a follow up, a user manual will also be provided for the correct usage of the videogame. Finally, the results obtained after in this project will be analyzed, and a study of the personal, environmental, economic and cultural impact of this work will be made.

Tabla de contenidos

Tabla de Ilustraciones

1 Introducción

Desde su nacimiento en los años 70 [1], la industria del videojuego no se ha detenido en su crecimiento, llegando a superar a las demás industrias culturales, como el cine o la música, tanto en ingresos como en crecimiento [2]. Su utilidad ha aumentado mucho en los últimos años, y ven uso no solo como un entretenimiento, sino como herramientas para la educación, para el ejercicio físico, o el desarrollo de la vida social [3].

Dentro de la industria, pese a que en los últimos años se ha visto el crecimiento del formato al que se está denominando doble-A o AA [4], podemos encontrar dos tipos principales de videojuego, los triple-A o AAA, y los *indie* o independientes. La principal diferencia entre estos es el presupuesto, con los primeros siendo desarrollos que suelen durar años y requiriendo a varias decenas de personas, y los segundos normalmente siendo desarrollados por equipos muy reducidos y con recursos mucho más reducidos. En la industria triple-A se suelen encontrar las grandes producciones, videojuegos que llegan a enormes números de personas y que en muchos casos se encuentran a la vanguardia de la tecnología, mientras en la industria independiente, pese a que en apariencia no sean tan vistosos, se suelen encontrar obras que toman riesgos, y suelen aportar ideas interesantes.

Ambos formatos de videojuego son elementos culturales que, como se ha explicado antes, están teniendo un gran impacto en la cultura popular [5], por lo que por todo lo previamente mencionado, y multitud de factores más, el desarrollo de un videojuego es muy relevante hoy en día, pues son uno de los tipos de software más influyentes en la actualidad.

1.1 Motivación

La decisión de desarrollar un videojuego viene principalmente de mi gran pasión por estos, además de mi conocimiento previo tanto del medio como de las herramientas necesarias para su creación. Decidí realizar un videojuego de supervivencia pues estos requieren, por norma general, de una mayor carga de programación al estar formados por múltiples sistemas que trabajarán en conjunto para formar el videojuego.

Hay múltiples inspiraciones detrás de este proyecto, y no sólo dentro del propio medio. La más clara es el videojuego Stardew Valley, cuyo género es el de simulación de granja, y comparte con este la vista cenital y la habilidad de cultivar plantas y obtener recursos de estas. Otra de las inspiraciones es el juego Subnautica, del género de supervivencia y en el cual el jugador ha de sobrevivir en un planeta sumergido bajo el agua, necesitando manejar su hambre, sed, salud y oxígeno, teniendo también un básico sistema de cultivos. Por último, la otra influencia, especialmente en cuanto a la ambientación, es la película y libro The Martian, en la que el astronauta Mark Watney se encuentra abandonado en Marte y ha de sobrevivir con los pocos recursos de los que dispone, habiendo un punto en que cosecha patatas como manera de obtener comida. El nombre de la llanura marciana en que aterriza Mark en esta obra tiene el nombre de Acidalia Planitia, de ahí el nombre de este videojuego, Project Acidalia.

1.2 Objetivos

El desarrollo de un videojuego completo suele ser muy largo y costoso, por lo que en este proyecto se buscará realizar un prototipo de un videojuego independiente en el que mostrar, de manera reducida, todas las mecánicas disponibles en lo que sería el producto final. Como se explicó anteriormente, se realizará un videojuego de supervivencia pues estos requieren de la implementación de un mayor número de mecánicas que videojuegos de otros géneros, como los arcades o los plataformas, y las características de las que inicialmente se planteó su implementación fueron un sistema de cultivos, el ciclo día y noche, un sistema de inventario, la capacidad de crear objetos a partir de recursos, un sistema de estados del personaje con salud, hambre, sed y temperatura corporal, recolección de materias primas a partir de diversos nodos de recursos, la exploración fuera de la zona inicial, un sistema de eventos que modificaran las condiciones jugables, un sistema de energía en que habría que decidir qué secciones de la nave encender, una interfaz de usuario, una historia, efectos de sonido, y una banda sonora. Además de esto, en un primer momento se iba a encargar del arte una persona externa, por lo que no tendría que encargarme de ello de forma personal.

Finalmente, el arte tuve que realizarlo yo, y a medida que desarrollaba el proyecto me di cuenta de que no iba a resultar viable implementar todo lo que se había decidido inicialmente, por lo que tuve que recortar el enfoque del proyecto. Los atributos que decidí mantener fueron el sistema de cultivos, el ciclo día y noche, el sistema de inventario, una versión simplificada de la recolección de recursos, así como un sistema de inventario externo al del jugador en forma de cofres, el sistema de estados, pese a que la temperatura no vería uso en esta versión del videojuego, la interfaz de usuario y los eventos de sonido. Además, decidí, a modo de sustituto de la creación de objetos a partir de recursos, implementar un sistema de compraventa con el que el jugador pudiera dar uso a los recursos obtenidos del sistema de cultivos y con el que pudiera también perpetuar su supervivencia.

2 Herramientas

Para llevar a cabo el desarrollo del videojuego se ha hecho uso de varias herramientas, las cuales han simplificado el proceso de creación de este. A continuación, se realizará una explicación del funcionamiento básico de cada una de estas, así como del uso que se les ha dado.

2.1 Unity

Unity es un motor de videojuegos multiplataforma, y es la herramienta que más uso ha visto a lo largo del desarrollo. Se escogió este motor gráfico por dos motivos principales. El primero es que de manera personal ya tenía conocimientos previos sobre su uso, lo cual me permitiría trabajar con este a una mayor velocidad. Además, Unity tiene una gran comunidad formada alrededor al tratarse de uno de los motores más populares dentro de la industria [6], por lo que es sencillo encontrar documentación y ayuda por parte de los usuarios cuando es necesario. La versión de Unity se ha ido actualizando a medida que avanzaba el desarrollo, ya que múltiples de las versiones que han ido saliendo en este tiempo han ofrecido cambios útiles para el ciclo de desarrollo, como un aumento sustancial en la velocidad de compilación o el modo oscuro, siendo la 2021.3.4f1 la utilizada en la versión actual de este videojuego.

Además, se ha utilizado la herramienta Unity HUB, la cual centraliza en una sola aplicación las múltiples versiones de Unity que puedas tener instaladas, simplificando así el correcto uso de los distintos proyectos en los que estés trabajando de forma paralela.

Unity tiene además la Asset Store, en la que puedes descargar diversos elementos creados por la comunidad, desde paquetes de texturas a módulos que añaden nuevas funcionalidades a Unity. De esta se descargaron algunos paquetes de iconos para poder comenzar a programar las mecánicas sin la necesidad de crear el arte previamente.

Por último, también existe un gestor de paquetes con el que añadir distintas funcionalidades al proyecto. En este caso se ha utilizado Cinemachine para mejorar el funcionamiento de la cámara de juego.

Ilustración 1: Logo Unity

2.2 Plastic SCM

Plastic SCM se ha utilizado como software de control de versiones principalmente por su simplicidad de uso al venir instalado e integrado con Unity. La principal utilidad de este al trabajar ha sido la de asegurarme de que en caso de fallo por parte del hardware en que está almacenado el proyecto, este no sería perdido, proporcionando así mucha seguridad. Este software ofrece muchas otras funcionalidades, como la creación de ramas o la habilidad de comparar las diferencias al hacer cambios en un archivo, pero no fue necesario su uso.

Ilustración 2: Logo Plastic SCM

2.3 Microsoft Visual Studio

La herramienta Visual Studio, desarrollada por Microsoft, ha sido utilizada en su versión Community como el IDE para programar los diversos scripts que se encuentran en el proyecto. Por defecto Unity utiliza la versión de 2019, ofreciendo también la opción de instalarlo a la vez que el motor, pero se decidió utilizar la versión de 2022 debido a las mejoras que presentaba en el software de autocompletado de código, IntelliCode, y por el mejor rendimiento de esta.

Ilustración 3: Logo Visual Studio

Además, para la realización de los diagramas de clases que se encuentras más adelante en esta memoria, se ha utilizado el módulo Diseñador de Clases. La principal ventaja de este es la velocidad con la que se pueden crear estos diagramas, pues genera automáticamente una versión básica de estos analizando los distintos *scripts* del proyecto, teniendo el usuario que

preocuparse únicamente de darle el formato deseado, pero ha tenido como inconveniente las pocas modificaciones en el diseño gráfico del diagrama que permite. Este es el motivo por el que los iconos para indicar de qué se trata cada elemento son los proporcionados por la propia Microsoft dentro del programa, en lugar de los habitualmente utilizados. En la siguiente figura podemos encontrar una leyenda con estos.

Icon	Description	Icon	Description
{}	Namespace		Method or Function
	Class		Operator
	Interface		Property
	Structure		Field or Variable
	Union		Event
	Enum		Constant
	TypeDef		Enum Item
	Module		Map Item
	Extension Method		External Declaration
	Delegate		Error
	Exception		Template
	Map		Unknown
	Type Forwarding		

Ilustración 4: Leyenda diagramas de clase

2.4 Aseprite

Aseprite es un editor de gráficos rasterizados el cual se ha utilizado para crear todas las imágenes que encontramos en el videojuego. Este es un editor muy comúnmente utilizado a la hora de crear gráficos de estilo *pixel art*, y fue elegido para este trabajo debido a su sencillez de uso, y por la multitud de opciones que ofrece.

Ilustración 5: Logo Aseprite

2.5 Trello

Trello es un software de administración de proyectos el cual se utiliza desde la web, y ha servido para planificar las tareas a realizar. Es uno de los más comúnmente utilizados, y pese a que está principalmente orientado a un uso por parte de equipos, ha sido útil también trabajando en solitario, pues es habitual no recordar los detalles de cómo fueron implementadas ciertas características varias semanas o meses atrás.

Ilustración 6: Logo Trello

2.6 Draw.io

Draw.io es un software de dibujo de gráficos que ha sido utilizado para la creación del diagrama de estados del videojuego presente en esta memoria. Inicialmente se planeó utilizarlo para la creación de los diagramas de clases, pero el módulo de Visual Studio resulto de más sencillo uso para ello.

Ilustración 7: Logo Draw.io

3 Análisis

En el desarrollo de todo videojuego es importante delimitar el alcance de este de forma temprana en el desarrollo para evitar encontrar fallos en el diseño cuando este se encuentre muy avanzado, por lo que se comenzó con una fase de análisis, en la que se estructuraría el diseño general del videojuego.

3.1.1 Recolección de influencias

Lo primero que se hizo, para entender correctamente el concepto de videojuego que se pretendía desarrollar, fue experimentar sus principales influencias de forma breve y tomando notas sobre todo lo relevante.

Para ello se llevaron a cabo dos sesiones de alrededor de una hora de los videojuegos Subnautica y Stardew Valley. De estos se anotaron los tiempos de los ciclos día y noche, así como los tiempos de crecimiento de los cultivos. Del primero también se anotó el ritmo de disminución de su sistema de hambre y sed, y del segundo se tomaron algunas notas sobre el aspecto de sus gráficos. Por último, se vio la película The Martian, también tomando notas, y extrayendo de esta el nombre del videojuego, Project Acidalia, así como algunas ideas que no han sido relevantes en la versión actual del juego, como permitir al jugador tener un Rover con el que explorar la superficie del planeta en que se encuentra para de ese modo obtener recursos.

Ilustración 8: Stardew Valley

Ilustración 9: Subnautica

3.1.2 Sesión de brainstorming

Utilizando las notas previamente tomadas se creó un documento en el que en un tiempo indefinido se anotó todo lo que se me ocurrió relacionado con la imagen mental que tenía del videojuego en su versión final, sin ningún tipo de criba. A continuación, se filtraron todas las ideas que no cuadrarían con esta primera versión y se hizo una selección de las que, en primera instancia, serían implementadas.

Brainstorm Stravaganza

Game Ideas

Surviving, Top-Down, Plant Farming, Hydroponics, The Martian, Solitude, Light Exploration, Expansion, Ship, Planet, Sci-fi, Light Humor, Subnautica, Stardew Valley, Lifeline, Day-Night Cycle, Resource Management, Minimalism, Few Good Systems > Many Mediocre Ones, Ico & SotC, Darkness, Hazardous Environment, Stranded, No Escape, Engineer, Crafting, Game-System Removal, Environmental Effects, Black&White, Dark Shades of Blue, Lifeline-like Music, Rocky Surface, Self Fertilizer, Starting Water + Water from Planet's Ice, Disco Music Reference, Base Damage Analysis System (Check Breaches), The Martian 1:41:06, In Space No one Can Hear You Scream, ROVER!

Horror?, Monologue?/ AI Companion?, Plant Hybridization?, Truly Alone?, Resource Gathering?, Power Rerouting?, Ceres?/Phobos?/Fictional Planet?, Low Temperatures?/Hot Temperatures?, Temperature System?, Shelter with appliances?, Survivable?, Rescue "Timer"?, Food Spoils?,

Ilustración 10: Documento de Brainstorm

3.1.3 Desarrollo del GDD

Con las ideas de la sesión de *brainstorming* ya seleccionadas se continuó creando un Game Design Document, en el que se recogieron las partes principales que compondrían el videojuego, comenzando con una sección en que se declara lo que será el proyecto como concepto, continuando con los objetivos de este, explicando a continuación cuales serían las mecánicas principales, dando un pequeño resumen del bucle de juego básico, y por último dando detalles sobre cuál será el estilo artístico del juego.

Los GDDs habitualmente pueden llegar a ocupar varias decenas de hojas, cubriendo hasta los aspectos con menor importancia del proyecto, pero por simplicidad y por tiempo se decidió elaborar una versión simplificada que sirviera principalmente de guía a lo largo del desarrollo. Este además es un documento en constante cambio, pues diversos problemas pueden surgir durante el desarrollo, obligando a cambiar el diseño, y que pueden hacer que el contenido del GDD quede obsoleto y necesite ser actualizado.

En la siguiente ilustración podemos observar la primera versión del GDD del proyecto.

Project Acidalia

Game Overview:

The name is Project Acidalia, it is a survival game mixed with farming and its main inspiration is The Martian, although it also takes some from Subnautica, Stardew Valley, Faster Than Light and Lifeline.

Objective:

The player's main goal is to survive with the resources they can attain. It will start with the player being stranded in a deserted planet after their ship crashes, and they will have to slowly upgrade their tools and manage the ships' power to survive.

Mechanics:

The mechanics mainly center around survival, with the player being able to eat, drink and sleep, plant, water and harvest crops, craft new items, and manage the crashed ship's power. The game will feature a day-night cycle, a health, thirst, hunger and temperature meters, and a crashed ship and a planet to explore.

Gameplay:

The game is divided in days. In each of these the player would wake up in the ship, check their health, thirst and hunger meters and refill them if necessary, plant, water or harvest their crops, and go out exploring. Each day will last for a couple of minutes, and sometimes different weather effects will take effect, which will force the player to manage the ship's power or take some warmer clothes when exploring.

Style:

The game's visuals will be mainly composed of different shades of blue, using darker colors for the most part. The music will be gloomy and dark, very similar to Lifeline's, but with it ramping up slightly when the game requires it, in a similar fashion to The Legend of Zelda: Breath of the Wild.

Ilustración 11: Game Design Document

3.1.4 Organización de tareas

Con el GDD preparado se pasó a elaborar un tablero de Trello, en el que se incluirían todas las tareas que, a priori, habría que completar para dar por terminado el desarrollo de este prototipo. Además, cada elemento se asignó a una categoría de cuatro, General, Game Design, Programming y Art, cada una representada por un color distinto. Todo esto contribuiría a una mejor organización durante el desarrollo. Además, se crearon cuatro listas y se asignaron todos los elementos a una de estas:

- **Backlog:** aquí se encontraron todas las tareas pendientes con las cuales todavía no se había empezado.
- **To Do:** en esta se situaron todas las tareas que fueran a realizarse en el corto plazo, es decir, en el periodo de aproximadamente las dos siguientes semanas.
- **Doing:** en esta lista se colocaron las tareas con las que ya se había empezado.
- **Done:** todas las tareas finalizadas fueron pasando a esta lista

Durante el desarrollo esta herramienta vio más o menos uso dependiendo del sistema que se estuviera implementando, pues algunos de ellos como el inventario o los cultivos requirieron de muchos días de trabajo enfocados en una única tarea. Además, ya que durante el desarrollo se volvieron a evaluar los objetivos del proyecto, algunas tarjetas fueron archivadas y sustituidas por otras.

En la siguiente ilustración podemos observar el tablero de Trello utilizado en un momento temprano del desarrollo del proyecto.

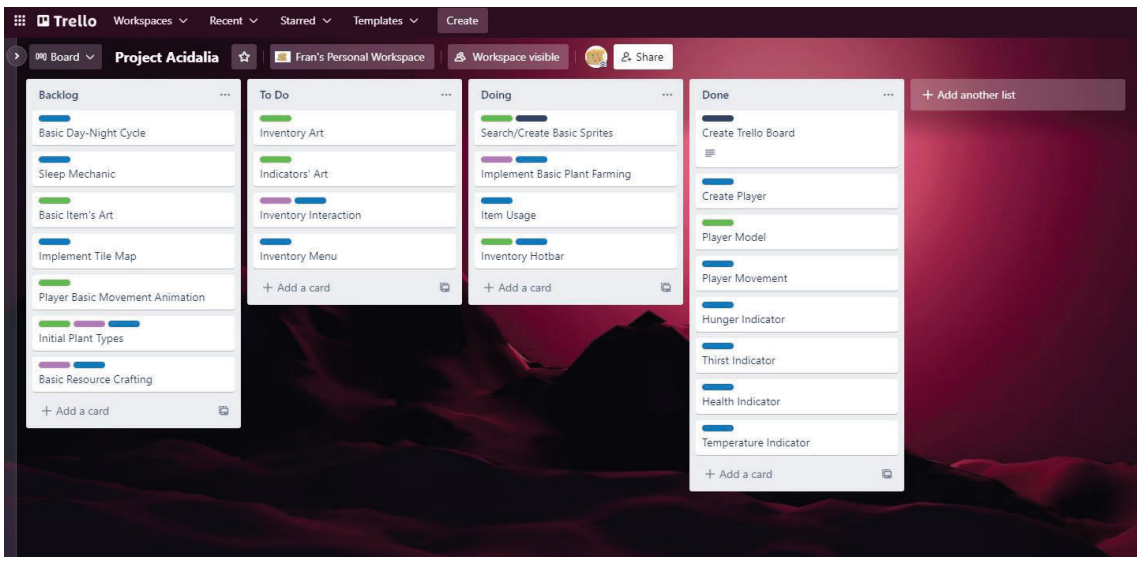


Ilustración 12: Tablero de Trello

3.1.5 Búsqueda de recursos

Por último, y para poder acelerar la implementación de los diversos sistemas, se hizo una búsqueda de paquetes de recursos en la Asset Store de Unity. Se descargaron múltiples paquetes, pero la mayoría no vieron uso. El que finalmente fue utilizado fue uno de nombre Clean Vector Icons, creado por el usuario PONETI. Este no sería el arte final del proyecto, y sería reemplazado con el tiempo por otros elementos de creación propia.

A continuación, se muestran algunos de los iconos utilizados de este paquete.

Ilustración 13: Icono inicial jugador

Ilustración 14: Icono inicial cofre

Ilustración 15: Icono inicial cultivo

4 Diseño técnico

Una vez finalizada la fase de análisis se dio pie a la fase de diseño técnico, la cual sería la de mayor duración. En esta se ha tenido como objetivo implementar las mecánicas previamente planificadas y teniendo siempre en cuenta la escalabilidad, pues ha de poderse construir un videojuego completo a partir del prototipo que se obtendrá al finalizar este proyecto.

Conviene, antes de continuar, tratar algunos aspectos básicos del funcionamiento de Unity. En este motor todo objecto que se encuentre en el videojuego, como la cámara o el jugador, son lo que se denomina GameObjects. Estos sin embargo no pueden hacer nada por si solos, por lo que requieren de Components que les darán las características necesarias para cumplir una u otra función. Los GameObjects pueden contener un número indefinido de Components, y estos pueden ser tanto los proveídos por Unity, como un Box Collider 2D para delimitar los bordes con los que colisiona la entidad, o un Mesh Renderer para renderizar su geometría, o de creación propia como, en la mayoría de los casos, los scripts. El único que es añadido por defecto en la creación de un GameObject y que no es posible eliminar es Transform, el cual sirve para indicar la posición que ocupa este en la escena.

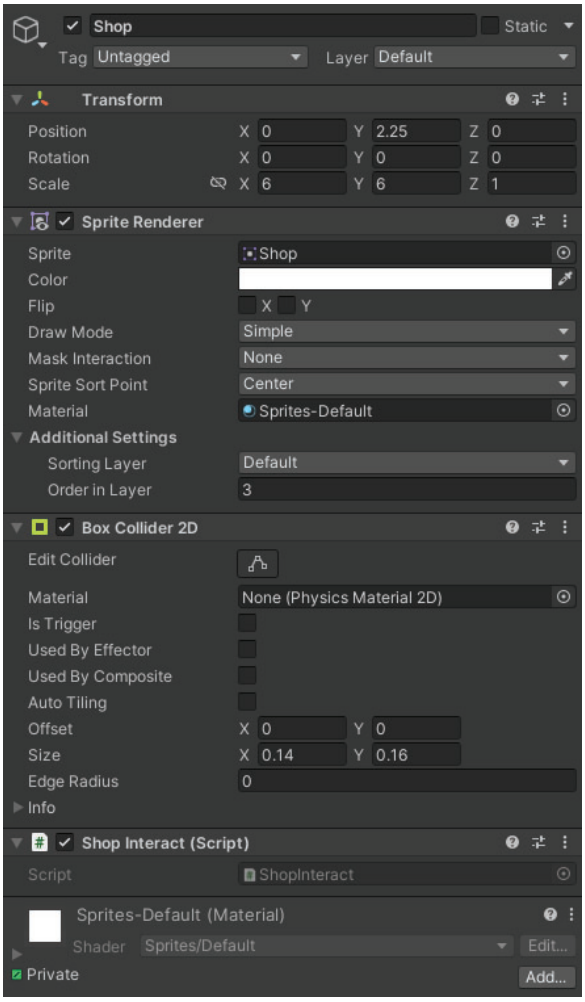


Ilustración 16: Ejemplo de GameObject con varios Components

4.1.1 Estructura del videojuego

Para entender la estructura de este proyecto habrá que realizar dos tipos de análisis. El primero se centrará en los estados en que puede encontrarse el videojuego durante su ejecución, y el segundo en la estructura de las múltiples clases que lo componen.

Para ayudar en la explicación de ambos, se han creado distintos tipos de diagramas. Para el análisis de los estados posibles, se ha creado un diagrama de estados, y para el análisis de las clases que lo componen se han creado 4 diagramas de clases distintos para facilitar su visualización.

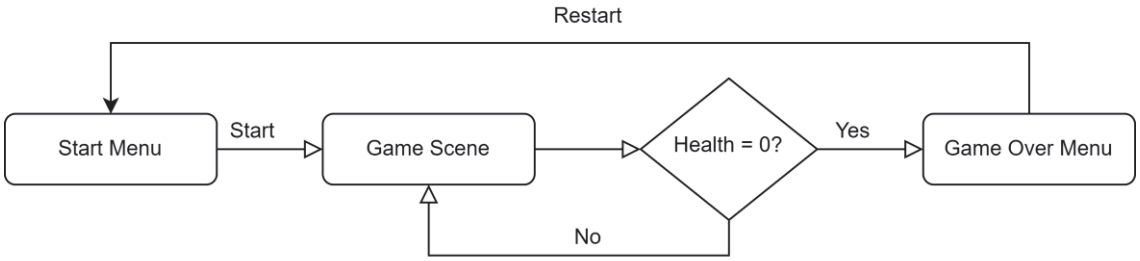


Ilustración 17:Diagrama de estados

En la figura anterior podemos observar la estructura de estados básica en que se puede encontrar el videojuego. Este comenzará en el menú de inicio o Start Menu. La única acción que puede tomar el jugador en este es la de presionar el botón central con nombre Start para dar inicio al videojuego.

Ilustración 18: Start Menu

Esto le llevará a la escena principal de juego o Game Scene y le dará control del personaje. En esta el usuario podrá jugar al videojuego en sí mismo, pudiendo realizar multitud de acciones, y se mantendrá en esta hasta que, por un motivo u otro, su indicador de vida llegue a 0.

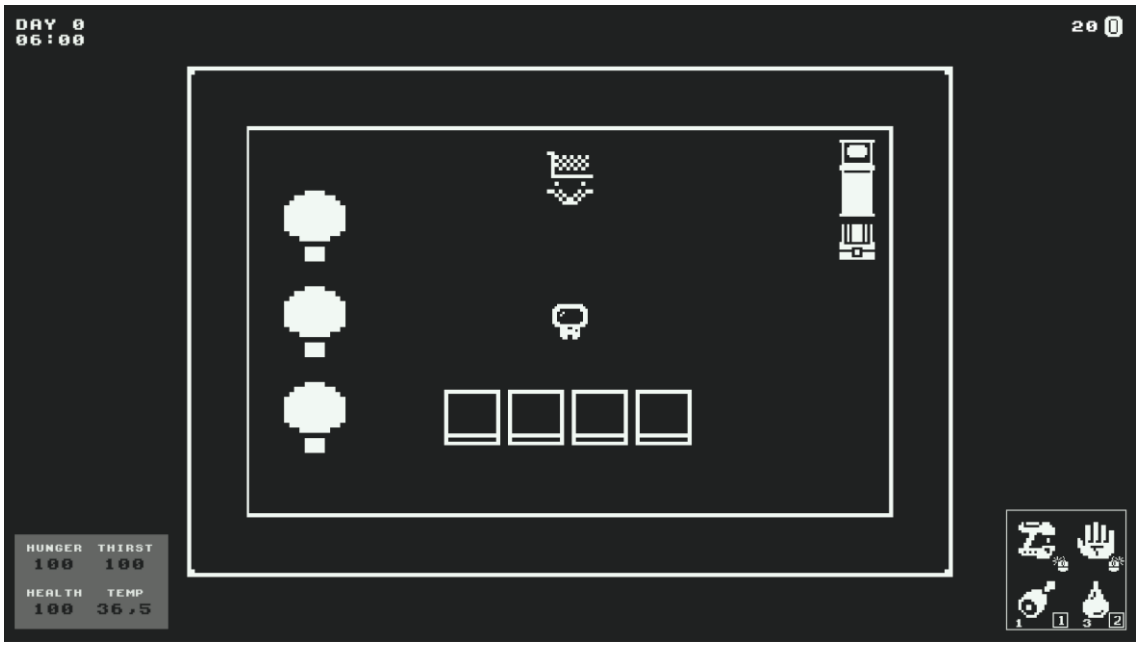


Ilustración 19: Game Scene

En caso de que esto ocurra, el control del personaje será retirado del jugador y aparecerá la pantalla de fin de partida o Game Over Menu. En esta el jugador podrá presionar el botón central con nombre Restart para así volver al menú de inicio.

Ilustración 20: Game Over Menu

Para el análisis de la estructura de clases del proyecto se han creado cuatro diagramas de clases utilizando el módulo Diseñador de Clases de Visual Studio. Como se mencionó anteriormente, la principal contra de esta herramienta es que no permite modificar apenas el diseño de los diagramas, lo cual incluye a los iconos que indican de qué se trata cada elemento. En el apartado 2.3 puede encontrarse la ilustración 4 con una leyenda donde se explican.

Los diagramas creados son, en orden de aparición, el general, donde se incluyen la mayoría de clases relacionadas con la lógica básica del videojuego y el control del personaje, el de inventario, donde encontramos la mayoría de las clases relacionadas con el inventario, tanto del jugador como la tienda o el cofre, así como la recogida de objetos, el de herencia, donde se encuentran las clases que heredan de otras clases propias, no ofrecidas por Unity, estando estas relacionadas con la interacción con diversos objetos y el control de algunos paneles de inventario, y el de Scriptable Objects, donde se han incluido todas las clases con este funcionamiento. Lo que es un Scriptable Object será explicado en la sección 4.1.6.

Antes de continuar se explicará el funcionamiento de la clase GameManager, pues esta es de mucha importancia en la estructura general del proyecto. Se ha utilizado para simplificar el acceso a ciertos elementos importantes como el jugador, el inventario del jugador, o el tiempo del juego. Para ello se ha declarado como un Singleton, asegurándonos así de que solo existe una instancia de esta a la que podamos acceder de forma global.

En las siguientes páginas podemos encontrar los diagramas de clase del proyecto

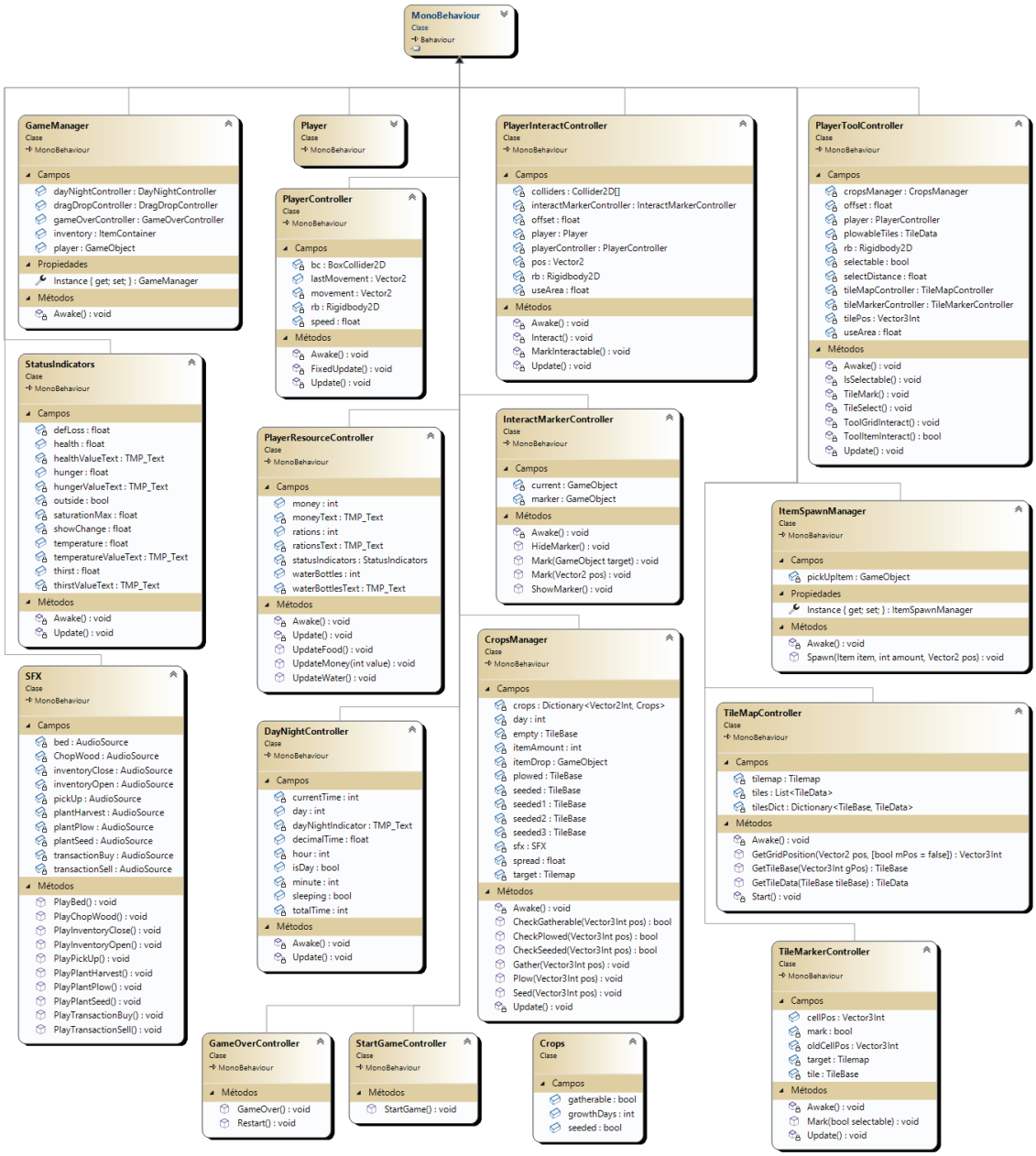


Ilustración 21: Diagrama de clases general

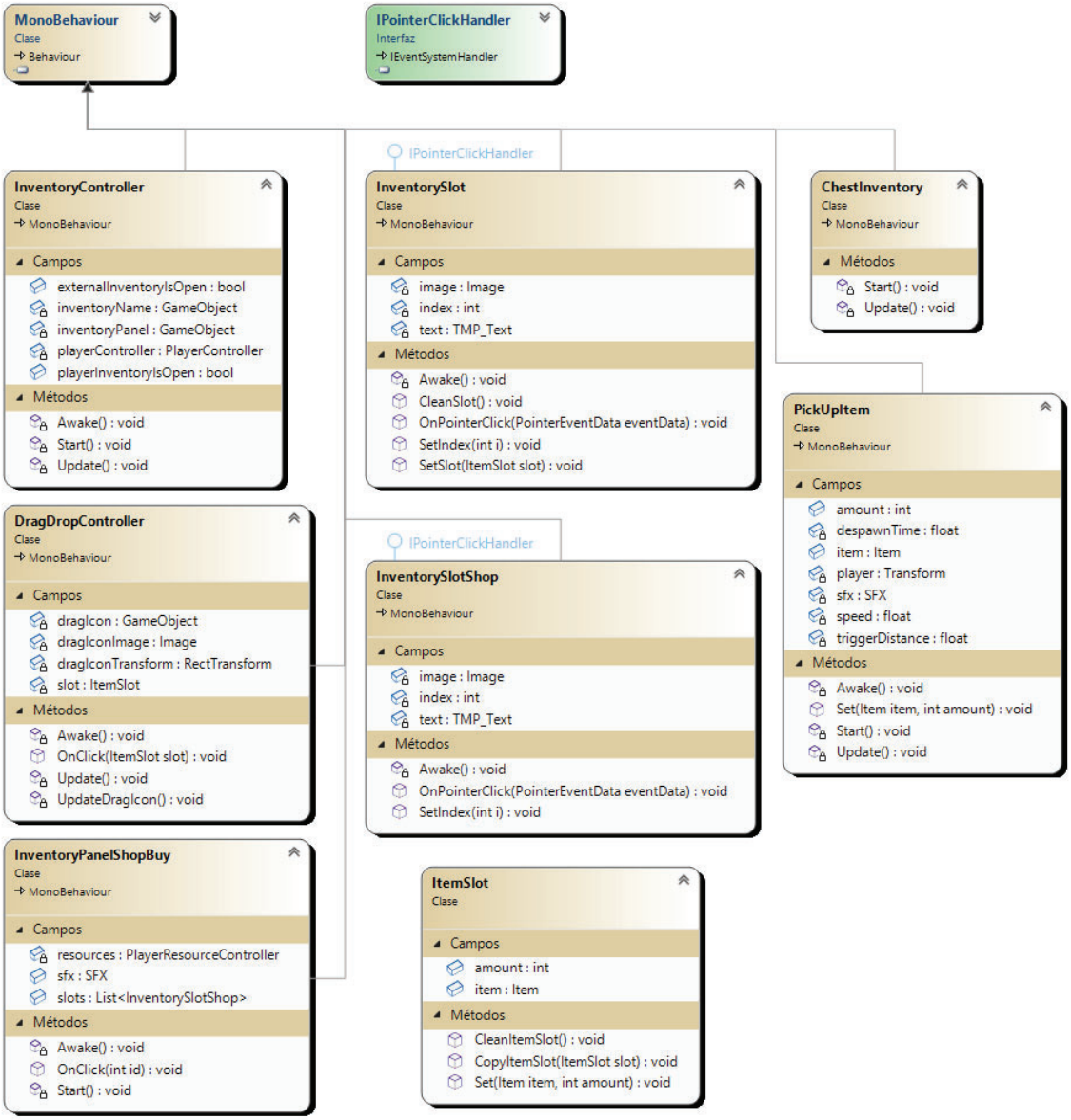


Ilustración 22: Diagrama de clases de inventario

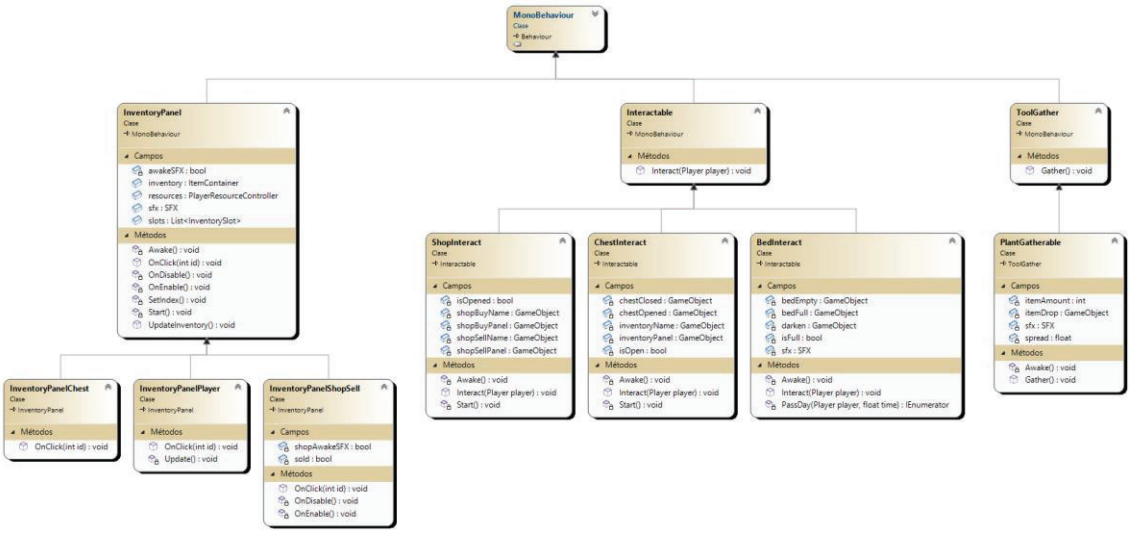


Ilustración 23: Diagrama de clases de herencia

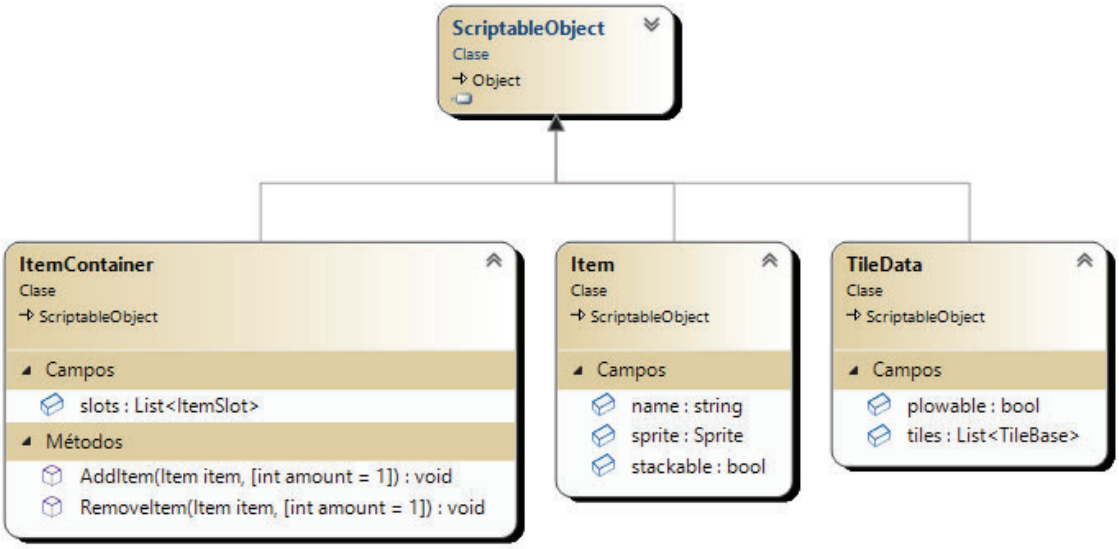


Ilustración 24: Diagrama de clases de Scriptable Objects

4.1.2 Área jugable

El área jugable es reducida en tamaño, pero contiene todo lo necesario para poder jugar al videojuego. Podemos observarla en la ilustración número 25. Se encuentra delimitada por 4 paredes, y podemos encontrar múltiples objetos en esta.

En el centro se encuentra el personaje que utilizará el jugador, el cual podrá ser movido una vez tome el control de este. En la esquina superior derecha podemos ver tanto la cama, con la que el jugador podrá pasar los días para así hacer crecer sus cultivos, como el cofre, que servirá como un segundo inventario donde almacenar los diversos recursos. En el centro de la zona superior encontramos la tienda, con la que el jugador obtendrá monedas vendiendo los objetos que logre recolectar, pudiendo además gastarlas para obtener raciones tanto de agua como de comida, con las que saciará su hambre y sed. En la zona izquierda encontramos 3 árboles los cuales podrán ser talados para recolectar madera. Estos han sido posicionados en representación de los nodos de recolección de recursos pues, con cambios mínimos en el código y en el *Sprite* utilizado podrían fácilmente ser sustituidos por, por ejemplo, rocas de donde recolectar piedras o venas de minerales de donde recoger metales y piedras preciosas. Finalmente, en la zona inferior se encuentran las macetas donde el jugador podrá arar la tierra, plantar semillas, verlas crecer, y recolectar sus cosechas.

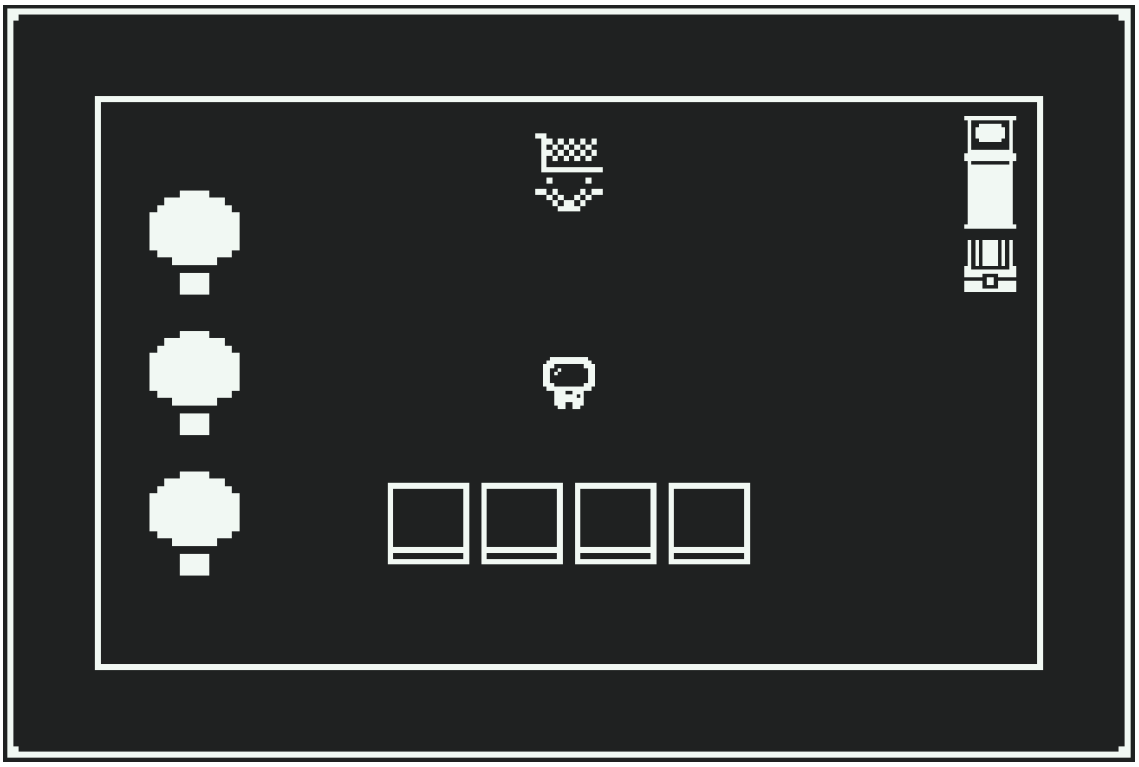


Ilustración 25: Área jugable

Para crear el suelo, las paredes y las macetas de forma sencillamente escalable se han utilizado los Tilemaps de Unity. Para ello se diseñaron las casillas en una hoja de *sprites*, también conocido como *tileset*, y la cual podemos observar en la ilustración 26, y se han creado 4 Tilemaps donde colocarlas. El primero de estos tiene de nombre Path y representa el suelo sobre el que el jugador puede caminar. Para delimitar el área jugable se creó Collidable, el cual tiene asignado un Component de tipo Composite Collider 2D, en lugar de uno estándar como Box Collider 2D, para evitar un error en que el jugador podía quedarse "enganchado" en una superficie aparentemente lisa al desplazarse pegado a esta. Otro de los Tilemaps es Crops, donde se han colocado las macetas, el cual tiene las mismas características que Path, y se creó principalmente para mantener una mejor organización en el proyecto. Por último, se creó un Tilemap de nombre Marker, el cual está superpuesto a los demás y sirve simplemente para marcar el área en que el jugador puede interactuar con los Tilemaps Path y Crips llamado Marker, lo cual está controlado por el script TileMarkerController.

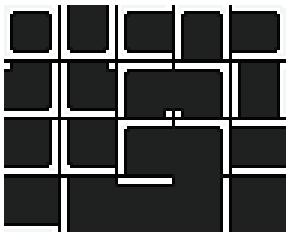


Ilustración 26: Tileset básico

Por último, como componente de GameManager existe la clase llamada TileMapController, la cual se encarga de almacenar información sobre los distintos *tiles* que se encuentran en el Tilemap Path en formato diccionario. Esta contiene también dos Scriptable Objects de nombre TilePlowable y TileNotPlowable en los que se almacena la información sobre qué tipo de casilla es apta para sembrar cultivos. Esto permite, en el futuro, crear nuevas casillas distintas a las de tipo PlanterBox con esta misma característica, como una de tierra o de arcilla, por ejemplo. Lo que es un Scriptable Object será explicado en la sección 4.1.6.

4.1.3 Movimiento, cámara e interacción

El movimiento del jugador se realiza mediante la clase PlayerController moviendo el Rigidbody2D asociado al personaje cuando una de las teclas de movimiento 'W' 'A' 'S' o 'D' es presionada. Para que el movimiento sea consistente se ha utilizado la función ofrecida por Unity FixedUpdate, pues funciona de forma independiente a la tasa de *frames* a la que esté corriendo el videojuego.

La cámara apenas ha necesitado modificaciones, sencillamente se añadió el módulo Cinemachine al proyecto y se creó y configuró una cámara adicional de este tipo, con lo que se ha logrado que esta siga al jugador con un cierto retraso. Además, como hijos de la cámara principal se han creado múltiples objetos que

funcionan como fuente de audio para los distintos efectos de sonido, aunque se profundizará más en este aspecto en la sección 4.1.9.

El jugador ha de tener la capacidad de interactuar con el entorno. Podemos considerar dos tipos de interacción, la que se da con elementos situados en un Tilemap, como los cultivos, y una más general que se dará con el resto de los objetos que podemos encontrar en la escena, como la cama, los árboles o los cofres. En las siguientes subsecciones se analizará el funcionamiento de ambos tipos.

4.1.3.1 Interacción general

La clase principal de este tipo de interacción es Interactable, la cual en sí misma no realiza ninguna acción, y solo contiene el método virtual Interact, pero es heredada por otras como BedInteract o ChestInteract, desde las que se realizarán las acciones necesarias y específicas para cada tipo de objeto sobrescribiendo su método. En el caso de los árboles se talarán, en el de la cama el personaje dormirá si es de noche, pasando así el día de juego, y para la tienda y el cofre se abrirán o cerrarán sus respectivos inventarios.

La habilidad de interactuar por parte del jugador es controlada por PlayerInteractController. Esta clase se encarga de detectar los objetos con los que el jugador puede interactuar en un cierto rango y marca uno de ellos haciendo aparecer el icono de una flecha sobre este, para lo que hace uso de la clase InteractMarkerController. Una vez marcado, si se detecta que el jugador presiona el click derecho o la tecla 'E', llamará a Interact para ejecutar las acciones específicas al objeto seleccionado.

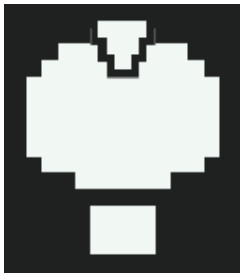


Ilustración 27: Árbol marcado

PlantGatherable es una clase que hereda de ToolGather, y está añadida como componente a los árboles que se encuentran en el videojuego. Esta se encarga de soltar 5 piezas de madera al talar un árbol, instanciando este número de instancias del Prefab de tipo Wood.

4.1.3.2 Interacción con Tilemap

La clase que permite al jugador interactuar con el Tilemap es PlayerToolInteract. Esta trabaja en conjunto con TileMapController para analizar los *tiles* que se encuentran dentro de un rango determinado y, en caso de que se pase el ratón por encima de una de ellas, las marca con la ayuda de la clase TileMarkerController. En caso de que sea posible interactuar con la casilla señalada y se detecte que se presiona el click izquierdo del ratón se llevará a cabo la interacción.

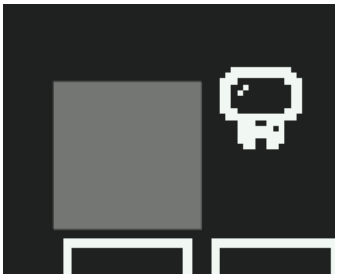
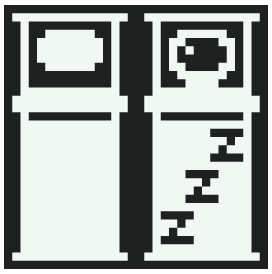


Ilustración 28: Marcador de Tilemap

La única interacción con los Tilemaps existente en la versión actual del videojuego es la relacionada con los cultivos, permitiendo al jugador arar la tierra, plantar semillas, y recolectar la cosecha una vez terminado su ciclo de crecimiento, todo lo cual será explicado en mayor profundidad en la sección 4.1.8.

4.1.4 Ciclo día y noche

El ciclo día y noche es manejado por la clase DayNightController, y el jugador puede avanzar el día durmiendo en la cama cuando es de noche, lo cual es controlado por la clase BedInteract. Para ello simplemente tendrá que acercarse a esta y, una vez aparezca el marcador manejado por InteractMarkerController, tendrá que presionar la tecla 'E'. La única utilidad de este sistema en la versión actual del videojuego es la de actualizar el número de días que llevan creciendo los cultivos, permitiéndolos así madurar para eventualmente ser recolectados por el jugador.



Ilustraciones 29: Cama y cama ocupada

El día y la hora son mostrados en la parte superior izquierda de la interfaz, y los minutos sólo avanzan en incrementos de 10, emulando al videojuego Stardew Valley. Un segundo en la vida real son 2 minutos dentro del juego, por lo que las horas duran un total de 30 segundos, y los días 12 minutos. Se considera que es de noche entre las 8 p.m. y las 6 a.m., periodo de tiempo en que el jugador podrá dormir.

Ilustración 30: Indicador de día y hora

4.1.5 Sistema de estados

El sistema de estados es mostrado en la zona inferior izquierda de la pantalla, y consta de 4 indicadores, hambre, sed, salud y temperatura, los cuales están controlados por la clase StatusIndicators, reduciendo sus valores con el tiempo si las condiciones son las adecuadas.

Ilustración 31: Indicadores de estado

El hambre se ve reducida a razón de 1 unidad cada 4 segundos, y la sed en 1 unidad cada 2 segundos. Una vez uno de ellos es reducido a 0 comienza a reducirse el valor de salud, a razón de 2.5 unidades cada segundo si solo uno de ellos se encuentra en 0, y a 5 unidades cada segundo si tanto el indicador hambre como el de sed se encuentran a 0. Además, existe una mecánica de saturación por la cual, pese a que no se ven representados en el indicador, tanto el hambre como la sed pueden aumentarse por encima de 100, hasta un límite de 120.

La temperatura no ve uso en la versión actual del videojuego, pues no existe manera de abandonar el área jugable, pero esta se vería reducida a una velocidad aún por determinar en caso de salir a explorar la superficie del planeta en que se da el videojuego, y una vez se encontrara por debajo de un cierto límite, la salud del jugador comenzaría a reducirse, pero a un ritmo más lento que en el caso del hambre y la sed.

La clase PlayerResourceController permite al jugador hacer uso de sus raciones, tanto de comida como de agua, presionando las teclas numéricas '1' y '2' del teclado respectivamente. Estas no pueden utilizarse si el jugador se encuentra saturado, es decir, si sus valores de hambre o sed se encuentra en un valor mayor o igual a 100 unidades. Ambas aumentan el valor de su estado asociado en 20 unidades, saturando al jugador si supera el límite anteriormente señalado, y el indicador con el número de raciones poseídas y el botón a presionar para su uso se encuentra en la zona inferior derecha de la interfaz.

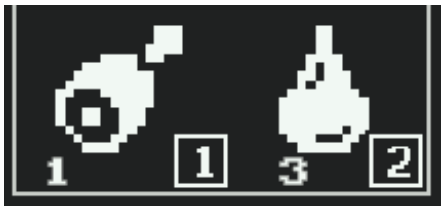


Ilustración 32: Indicador de raciones

4.1.6 Inventarios e ítems

El sistema de inventario y objetos es quizás el más complejo de este prototipo, por lo que se explicará en distintas subsecciones. Para una mejor comprensión del contenido de estas convendrá explicar previamente los conceptos de Scriptable Object y Prefab. Los Scriptable Objects son un tipo de contenedor de datos proveídos por Unity los cuales permiten almacenar grandes cantidades de información sin depender de una instancia de ninguna clase, mientras los Prefabs son un tipo de componente especial que permiten almacenar GameObjects con Components completamente configurados para facilitar su uso en distintas situaciones.

Además, es importante también mencionar a la clase InventoryPanel, la cual implementa las funcionalidades básicas de los distintos tipos de paneles de inventario, como el del jugador o el del cofre, y la cual será heredada por estos para añadir las funcionalidades necesarias.

Finalmente, Mediante el uso de la clase DragDropController se da al jugador la capacidad de desplazar objetos dentro de su propio inventario y entre inventarios distintos, como el del cofre o la tienda. Además, también permite al jugador arrastrar y soltar objetos fuera de inventario alguno, dejándolos en el mundo esperando a ser recogidos posteriormente por el personaje. Para asegurar que siempre se accede a la misma instancia de esta clase se ha añadido a GameManager como variable.

En las siguientes subsecciones se explicarán los distintos elementos relacionados con el sistema de inventario, así como los objetos que con este se almacenan.

4.1.6.1 Ítems

Dentro del videojuego el jugador podrá recolectar diversos objetos, los cuales tienen las características de nombre, apariencia, y si son o no apilables, es decir, si deben almacenarse de forma individual o si pueden agruparse dentro de un inventario. Para definirlos se creó la clase Item, la cual hereda de Scriptable Object. Se ha realizado de esta manera pues permite crear a partir de una única clase, en este caso Item, diversos tipos de objetos los cuales serán posteriormente asignados a un Prefab cuando vayan a ser cargados en escena. Los dos tipos de objetos existentes actualmente en el videojuego son Wood y Carrot. El primero se obtiene mediante la tala de árboles, soltando 5 unidades cada uno, y el segundo mediante la cosecha de cultivos, los cuales soltarán 3 unidades.

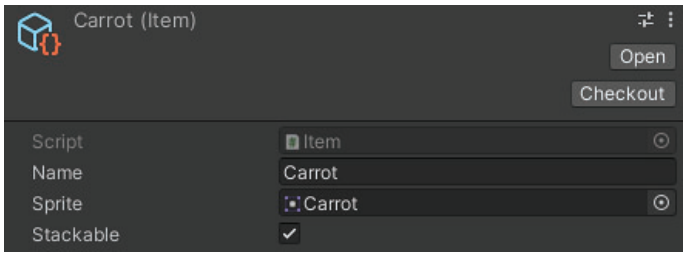


Ilustración 33: Scriptable Object Carrot

4.1.6.2 Inventario del jugador

La interfaz del inventario está formada por un GameObject de tipo panel que contiene múltiples Prefabs llamados InventorySlots. Estos son GameObjects de tipo botón y que, dependiendo del objeto que se encuentre situado en ese espacio del inventario, tomarán un aspecto u otro.

Este panel contiene además una clase llamada InventoryPanelPlayer, la cual hereda de InventoryPanel, y se encarga de actualizar los contenidos del inventario de forma continuada, además de invocar a la clase DragDropController cuando se haga click en uno de los InventorySlots que componen el inventario, permitiendo así que el jugador acceda a la mecánica de arrastrar y soltar objetos.

El Prefab Inventory Slot es un GameObject de tipo botón al que se le ha añadido como hijo otro objeto de tipo imagen, el cual actualizará su campo *sprite* para representar al objeto correspondiente, y en el que el objeto hijo de tipo texto es utilizado para mostrar la cantidad de unidades del objeto que en este se encuentra, en caso de tener el *item* a almacenar la propiedad *stackable* activada. Además, se le ha añadido un script llamado InventorySlot, el cual actualiza al Prefab de tipo Inventory Slot correspondiente actualizando sus campos en base a los del objeto.

La clase que permite al jugador abrirlo y cerrarlo mediante el presionado de la tecla 'TAB' es InventoryController, la cual simplemente activa o desactiva los distintos elementos de la interfaz que lo forman, y actualiza la lógica interna necesaria.

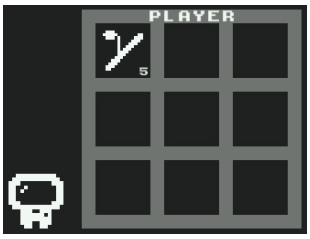


Ilustración 34: Inventario del jugador

El almacenado a nivel interno de los objetos del inventario del jugador se realiza mediante la clase ItemContainer, la cual extiende a Scriptable Object, lo cual, como se ha explicado previamente para los objetos, permite crear distintos inventarios con las mismas características, pero con distintos contenidos de forma sencilla. En esta, encontramos una lista de objetos del tipo ItemSlot, la cual es una clase que define el tipo de objeto del que se trata y la cantidad de unidades existente en este hueco del inventario en caso de ser *stackable*. ItemContainer se encarga también de añadir o eliminar objetos del inventario. El inventario del jugador fue añadido a GameManager para simplificar el acceso a este desde otras clases.

Por último, se ha de mencionar que cuando este o cualquier otro inventario es abierto el jugador pierde la habilidad de moverse. Esto se ha hecho así para simplificar el proceso de creación de la interfaz de los distintos inventarios, pues de otro modo seguirían al jugador cuando este se moviese como lo hace el resto de la interfaz.

4.1.6.3 Recogida de ítems

La recogida de los objetos que se encuentran instanciados en la escena se realiza mediante la clase PickUpItem. Esta se encuentra contenida en los Prefabs que representan a objetos que pueden ser recogidos, siendo Carrot y Wood los únicos en esta versión del videojuego, y se analiza la distancia entre estos y el jugador. En caso de que la distancia sea menor a un cierto valor, estos se desplazarán hacia el personaje, y una vez se encuentren lo suficientemente cerca, serán añadidos al inventario del jugador y eliminados de la escena.

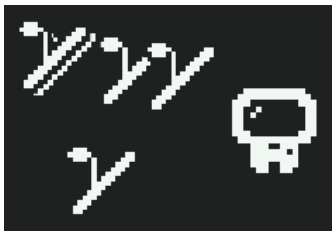


Ilustración 35: Wood dirigiéndose al jugador

4.1.6.4 Inventario del cofre

Para la creación del cofre se creó un Prefab con el mismo nombre para permitir la instanciación de nuevos objetos de este tipo de forma sencilla en caso de necesitarse en el futuro. Este tiene como hijos dos GameObjects con los *sprites* de sus dos posibles estados, abierto o cerrado, entre los cuales alternará mediante la clase ChestInteract. Esta hereda de Interactable, y permite al jugador interactuar con el cofre.

Una vez abierto el cofre, la clase InventoryPanelChest es la encargada de abrir el menú de inventario de este, el cual está implementado de forma muy similar al del jugador, con un GameObject de tipo panel con hijos de tipo InventorySlot. La principal diferencia es que este no es actualizado de manera constante, solo cuando se abre, se cierra, o se arrastra un *item* dentro o fuera de este. El nombre del Scriptable Object que maneja el inventario del cofre es InventoryChest.

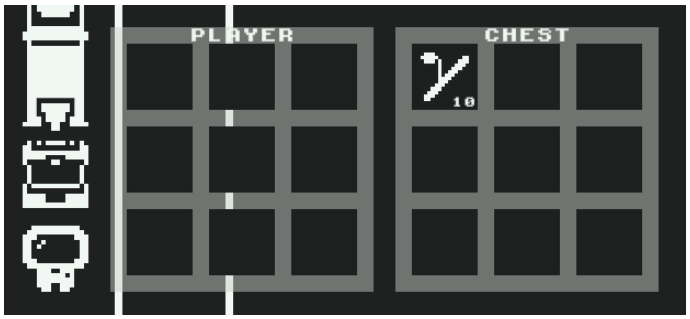


Ilustración 36: Inventario del cofre

4.1.7 Tienda y dinero

Inicialmente en lugar de una tienda se planeó la creación de un sistema de *crafting*, mediante el cual podrían transformarse las materias primas obtenidas en diversos objetos de utilidad para el jugador. Sin embargo, esto resultó fuera del alcance de este proyecto, pues habría llevado algo más tiempo del disponible, y habría supuesto recortar en otros apartados del videojuego, por lo que se decidió implementar un sencillo sistema de compraventa con el que proveer al jugador de los recursos *necesarios* para poder jugar al juego sin un límite de tiempo fijo, pues podría mantener gracias a este sus indicadores de hambre y sed en valores superiores a 0.

La tienda se encuentra en el centro de la zona superior del área jugable, y para interactuar con ella basta con presionar la tecla 'E' en su cercanía, lo cual, manejado por la clase que hereda de Interactable, ShopInteract, abrirá los paneles de compra y venta, manejados a su vez por las clases InventoryPanelShopBuy e InventoryPanelShopSell respectivamente, heredando la segunda de InventoryPanel. El panel de compra permite al jugador obtener raciones de comida a cambio de 10 monedas y raciones de agua por 5. Para ello basta con hacer click en el icono correspondiente, y serán añadidas automáticamente al inventario de raciones del jugador. El panel de venta tiene

su propio Scriptable Object, de nombre InventoryShopSell, que se encarga de su inventario, y para poder vender objetos el jugador tendrá que colocar los ítems que desee en uno de los InventorySlotShop que forman el panel, los cuales serán vendidos únicamente cuando el panel sea cerrado. La madera se vende a 1 moneda la unidad y las zanahorias a 3 monedas la unidad. El dinero es manejado mediante la clase PlayerResourceController.

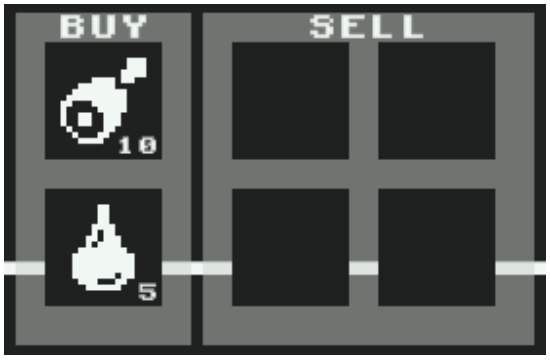


Ilustración 37: Ventana de compraventa

El jugador podrá ver cuánto dinero posee observando el indicador que se encuentra en la zona superior derecha de la pantalla. Al comienzo del juego tendrá 20 monedas, y como se ha explicado previamente, el jugador podrá o gastarlas en la tienda, u obtener más vendiendo madera o zanahorias. Este indicador se encuentra también controlado por la clase PlayerResourceController.

Ilustración 38: Indicador de monedas

4.1.8 Cultivos

Los cultivos representan una parte importante del bucle jugable del videojuego, y en este prototipo se ha implementado una versión básica de este sistema, con un único tipo de planta, las zanahorias, y permitiendo al jugador arar la tierra, plantar semillas, ver crecer a las plantas, y cosecharlas una vez maduran. Por simplicidad, y ya que no era necesario para el funcionamiento básico de los cultivos, las semillas se consideran infinitas y solo existen una vez colocadas en la maceta. Además, no será necesario regar las plantas, a diferencia de otros juegos del género como Stardew Valley.

Para poder colocar semillas y cultivarlas el jugador ha de posicionar su ratón sobre una de las casillas válidas para cultivar plantas y presionar el click izquierdo en su ratón. Esto es controlado por PlayerToolInteract, como vimos en

la subsección 4.1.3.2. Esto solo puede realizarse sobre la casilla de tipo TilePlanterBox en la versión actual del juego, aunque como también se mencionó anteriormente, en versiones futuras podría realizarse también en otras como la tierra o la arcilla.

Ilustración 39: PlanterBox vacía

En caso de encontrarse vacía la interacción transformará la casilla del tipo TilePlanterBox en TilePlanterBoxPlow, señalando que la tierra ha sido arada.

Ilustración 40: PlanterBox arada

A continuación, si la tierra se encuentra arada, la interacción transformará la casilla de tipo TilePlanterBoxPlow en una del tipo TilePlanterBoxSeed.

Ilustración 41: PlanterBox con semillas

Al interactuar con una casilla del tipo TilePlanterBoxSeed no habrá ningún efecto. Para observar una progresión en el crecimiento del cultivo el jugador habrá de pasar la noche, o esperando, o mediante el uso de la cama, pues la edad de las plantas está medida en incrementos de días.

Para dar las características que definen a un determinado cultivo se utiliza la clase Crops, y la clase encargada de realizar el seguimiento de estos es CropsManager. Esta se encarga de almacenárlos en un diccionario con su ubicación en el área jugable, y actualizando su fase de crecimiento cuando es necesario.

El crecimiento de los cultivos se ha dividido en 3 fases. La primera muestra los primeros brotes de la planta, y ocurre un día tras el plantado de las semillas. La segunda muestra la planta con algunos brotes más y algo más madura, ocurre al tercer día de crecimiento. La tercera y final muestra el cultivo completamente maduro y listo para cosechar, y ocurre al quinto día.

Ilustración 42: Fase 1 del cultivo

Ilustración 43: Fase 2 del cultivo

Ilustración 44: Fase 3 del cultivo

Una vez el cultivo se encuentra en la fase 3, si el jugador interactúa con la casilla en que se encuentra, esta será devuelta a su estado vacío original, y se soltarán 3 Prefabs del tipo Carrot, los cuales el jugador podrá recoger y almacenar en su inventario o vender. Esta funcionalidad también es manejada por la clase CropsManager.

Ilustración 45: Carrots

4.1.9 Gráficos y efectos de sonido

Para poder comenzar a implementar los distintos sistemas del juego de forma rápida, inicialmente se descargó el paquete Clean Vector Icons de la Asset Store de Unity. Estos, sin embargo, fueron reemplazados progresivamente por los creados personalmente. Podemos encontrar diversos ejemplos de los *sprites* en las distintas ilustraciones de esta memoria. La estética *pixel art* fue elegida pues tenía algo de práctica previa con esta técnica y era más sencilla de ejecutar que otras alternativas.

Además, todo lo encontrado en este proyecto está en uno de dos colores, #222323 como el color oscuro, y #F0F6F0 como color claro. La estética de 1 bit ayudó a crear el arte de forma más rápida, a la vez que mejoró los resultados obtenidos al facilitar la creación de un arte relativamente limpio y entendible con mis reducidas habilidades de dibujo.

Se ha de mencionar también que la fuente utilizada en todos los textos encontrados en el prototipo es Kongtext, creada por el usuario codeman38. Esta fue seleccionada por su buena coherencia con la estética *pixel art* utilizada en los gráficos del videojuego.

El sonido fue lo último añadido al videojuego, y se compone principalmente de efectos de sonido que ayudan a dar *feedback* al jugador al realizar ciertas acciones. Estos se invocan en las distintas clases del proyecto mediante llamadas a la clase SFX, la cual contiene las funciones encargadas de reproducir los múltiples efectos de sonido. Todos los efectos de sonido que se encuentran en el videojuego han sido descargados desde la librería de sonidos online Zapsplat.

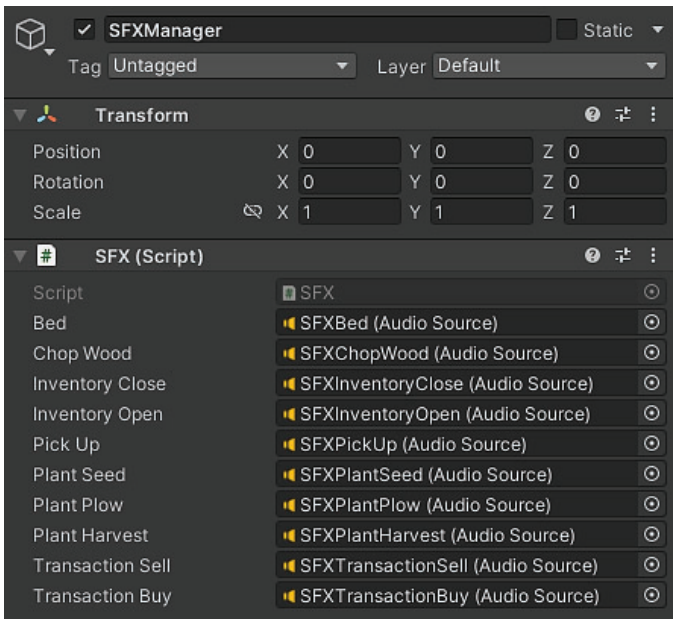


Ilustración 46: SFXManager

5 Manual de usuario

Para iniciar el juego bastará con ejecutarlo y presionar el botón START de la pantalla de inicio.

Ilustración 47: Botón START

Esto llevará al jugador a la escena de juego, donde podrá comenzar a interactuar con este utilizando el teclado y ratón mediante los siguientes controles:

Controles de movimiento:

Moverse arriba:	'W'
Moverse izquierda:	'A'
Moverse abajo:	'S'
Moverse derecha:	'D'

Controles de interacción:

Abrir inventario:	'TAB'
Talar árbol:	'Click Izquierdo'
Plantar/Cosechar:	'Click Izquierdo'
Interactuar:	'Click Derecho' o 'E'

Controles de recursos:

Consumir comida:	'1'
Consumir agua:	'2'

El jugador podrá además utilizar el click izquierdo para recolectar objetos en su inventario o entre distintos inventarios, así como dejar objetos en el suelo al hacer click fuera de inventario alguno.

Para poder vender objetos el jugador tendrá que abrir la tienda y poner los objetos que desee vender en el recuadro de venta de esta. La transacción será realizada cuando este sea cerrado.

Dentro del propio juego también se ofrece una pequeña guía en la zona inferior derecha de la interfaz donde se indica cómo interactuar con el entorno y cómo consumir las raciones disponibles.

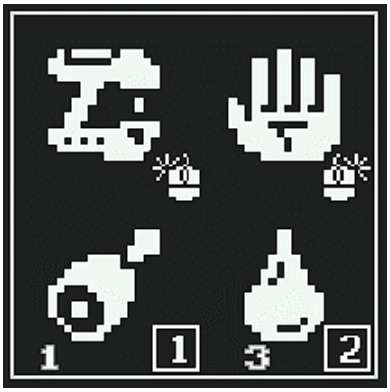


Ilustración 48: Ayuda de controles

El primer icono representa la herramienta del personaje, la cual es usada en acciones que la requieran como talar árboles o arar la tierra, mientras que el segundo icono representa sus manos, las cuales serán utilizadas en acciones más simples como abrir el cofre, interactuar con la tienda, o irse a la cama una vez sea de noche.

Los dos iconos inferiores muestran el número de raciones disponibles mediante el número del lado izquierdo, y el control que hay que presionar para consumirlas con el icono del número del lado derecho.

La salud solo se verá reducida una vez uno o ambos indicadores de hambre y sed lleguen a cero, y en caso de que la salud del personaje llegue a 0 se mostrará la pantalla de fin del juego. Para poder volver a iniciar el juego el jugador tendrá que presionar el botón RESTART.

Ilustración 49: Botón RESTART

6 Resultados y conclusiones

De forma inicial se hizo una propuesta excesivamente optimista sobre lo que podría ser logrado en este proyecto, lo cual sumado a finalmente tener que realizar el arte personalmente hizo que se tuvieran que volver a evaluar los objetivos planteados, modificándolos a algo más realista. Se vio reducido el espacio de juego a una sola habitación, por lo que no se dio uso al sistema de temperatura, y se tuvo que modificar el sistema de *crafting* por uno de compraventa, ya que era más sencillo de implementar, entre otros cambios menores.

Las mayores complicaciones en la implementación se han encontrado en el sistema de inventario y el sistema de cultivos. El sistema de inventario ha requerido de aprendizaje sobre el uso de los Scriptable Objects, y ha requerido de más tiempo del que se calculó inicialmente, pues se ha necesitado de mucho código para lograr su correcto funcionamiento. Con el sistema de cultivos no ha sido necesario realizar un aprendizaje específico sobre ningún tema, pero también ha requerido de muchas horas de diseño e implementación para lograr buenos resultados. La mecánica de regado de los cultivos fue también eliminada por falta de tiempo.

Considero, sin embargo, que el resultado obtenido, especialmente en el aspecto de los sistemas implementados, ha sido muy positivo, siendo este prototipo una buena base sobre la que se podrá elaborar posteriormente un videojuego completo si así se desea. La estructura de clases está preparada para añadir nuevas funcionalidades de forma rápida, y en general la gran mayoría de decisiones de diseño se han tomado pensando en facilitar la capacidad de expansión futura.

7 Análisis de impacto

A nivel personal este proyecto ha tenido un gran impacto. No solo me ha servido como una excelente experiencia de aprendizaje, sino que también me será de utilidad en un futuro a la hora de buscar un trabajo en la industria del videojuego, sumando a la experiencia que tengo en la creación de estos. Además, creo que me ha aportado un gran nivel de desarrollo personal, pues me ha permitido mejorar múltiples competencias como mi habilidad de estimar el tiempo que llevará la implementación de ciertos sistemas en este tipo de proyectos, o mi capacidad de fijar metas alcanzables.

La ONU propuso en el año 2015 una serie de objetivos globales a alcanzar en los siguientes 15 años [7]. Algunos ejemplos son el fin de la pobreza, la reducción de las desigualdades, mejorar la producción y el consumo para no dañar con estos el planeta, o proteger a la vida submarina. Al tratarse este proyecto de un videojuego no creo que la mayoría de estos puntos sean aplicables. Lo único relacionado con estos será su modo de distribución, el cual al tratarse de un videojuego *indie* y con muy poco presupuesto, muy probablemente deberá ser exclusivamente digital, lo cual será una producción responsable al ser un formato mucho menos contaminante que el físico.

Los desarrollos de videojuegos suelen tener un alto riesgo, pues tienen costos muy elevados, pero a su vez la industria se encuentra en su punto más alto, y está presentando cifras muy positivas [3]. Pese a que la relevancia económica del proyecto en su estado actual es mínima, en caso de continuar con su desarrollo y crear un videojuego completo, este podría ser vendido a través de múltiples plataformas para obtener beneficios.

Como se ha dicho antes, la relevancia del videojuego está en el punto más alto desde su aparición, por lo que a un nivel cultural se trata de un tipo de software con mucho impacto. En caso de continuar con el desarrollo de este proyecto y distribuirlo, este llegaría a muchas personas las cuales se verían impactadas de una forma u otra por esta obra. Además, incluso en su estado actual, tanto esta memoria como el prototipo desarrollado pueden servir para otras personas que en el futuro se enfrenten a algún proyecto de características similares, ayudando a detectar posibles errores antes de cometerlos.

8 Bibliografía

8.1 Recursos de documentación

[1] Nadala Fernández. Historia de los videojuegos. [Online]. Available:

https://www.fib.upc.edu/retro-informatica/historia/videojocs.html

[2] Michael "Flashback" Mamerow. (2022). Gaming Industry vs. Other Entertainment Industries. [Online]. Available:

https://raiseyourskillz.com/gaming-industry-vs-other-entertainment-industries-2021/

[3] Frankie Wallace. (2020, March 18). Beyond Entertainment: Nontraditional Uses For Video Games. [Online]. Available:

https://headstuff.org/entertainment/gaming/nontraditional-uses-video-games/

[4] Janet Garcia. (2020, April 16). Video Game Dictionary Wiki Guide, AA (double-A). [Online]. Available:

https://www.ign.com/wikis/video-game-dictionary/AA_(double-A)

[5] University of Minnesota. (2016, March 22). Understanding Media and Culture, 10.4, The Impact of Video Games on Culture. [Online]. Available:

https://open.lib.umn.edu/mediaandculture/chapter/10-4-the-impact-of-video-games-on-culture/#:~:text=The%20aesthetics%20and%20principles%20of,of%20new%20means%20of%20interaction.

[6] Joseph Allen. (2021, August 25). Unity Is The Most Popular Steam Game Engine By Far. [Online]. Available:

https://techraptor.net/gaming/news/unity-is-most-popular-steam-game-engine-by-far

[7] Naciones Unidas. (2015). Objetivos de Desarrollo Sostenible. [Online]. Available:

https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/

8.2 Recursos técnicos

PONETI. (2020, April 21). Clean Vector Icons [Online]. Available:
https://assetstore.unity.com/packages/2d/gui/icons/clean-vector-icons-132084#description

codeman38. Kongtext Font. [Online]. Available:
https://www.1001fonts.com/kongtext-font.html

All Sound Effects. [Online]. Available:
https://www.zapsplat.com/

Este documento esta firmado por

Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
Fecha/Hora	Thu Jun 30 20:02:50 CEST 2022
Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
Numero de Serie	561
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)