ZOMBIE ATTACK

Tarea realizada por el
EQUIPO I:

Luís Javier Moreno Rojas

Elena Milara Mencía

David Lozano Varona

Cristina Martínez Otto

Gustavo Herrero Lapuente

Ingeniería de Videojuegos

Grado en Diseño y Desarrollo de Videojuegos.

Índice

1. Introducción 4

2. Patrones 4

2.1. Observer 4

2.2. Command 5

2.3. ObjectPool y Prototype 6

2.4. State 7

2.5. DirtyFlag 9

3. Assets Propios 9

4. Referencias y elementos de terceros 9

5. Créditos 9

6. Tabla de Requisitos 10

Índice de Imágenes

Ilustración 1 UML patrón observer....4
Ilustración 2 UML patrón command....5
Ilustración 3 UML patrón objectpool y prototype....7
Ilustración 4 Diagrama de estados....8
Ilustración 5 UML patrón state....8

1. Introducción

Zombies Attacks es un juego de terror con condición de victoria y derrota, en el cual el jugador tendrá que intentar matar a unos zombis en un total de 5 rondas en las cual irá aumentando la dificultad ya que cada vez irán apareciendo más zombis y en la última aparece un boss final. Además, tiene la posibilidad de iniciar el juego y poder salir del mismo mediante “Escape” y “Quit”, para así poder reiniciar la partida. Si se desea cargar la posición, vida y balas de una anterior partida, se puede hacer click en Load.

2. Patrones

A continuación, se mostrarán los patrones de diseño implementados en el juego para incorporar una buena arquitectura.

2.1. Observer

El patrón Observer se ha implementado mediante unos Power Ups que sueltan los zombis al morir. Estos Power Ups se tratan de un cargador de munición y un botiquín. El Observer implementa un sujeto, los cuales se tratarán de los GameObjects munición y botiquín. Por otro lado, están los observadores a los cuales se avisa cuando tienen que ejecutar la acción, los observadores de munición se tratan de rellenar la munición del jugador y ejecutar un audio, mientras que el botiquín tendrá unos observadores que curarán la vida del jugador y ejecutarán un audio. Para avisar el sujeto a los observadores, estos se ejecutarán cuando la munición o el botiquín entran en contacto con el jugador. Gracias a este patrón se nos ha permitido desacoplar los power ups de sus funciones, las cuales dependerían del jugador.

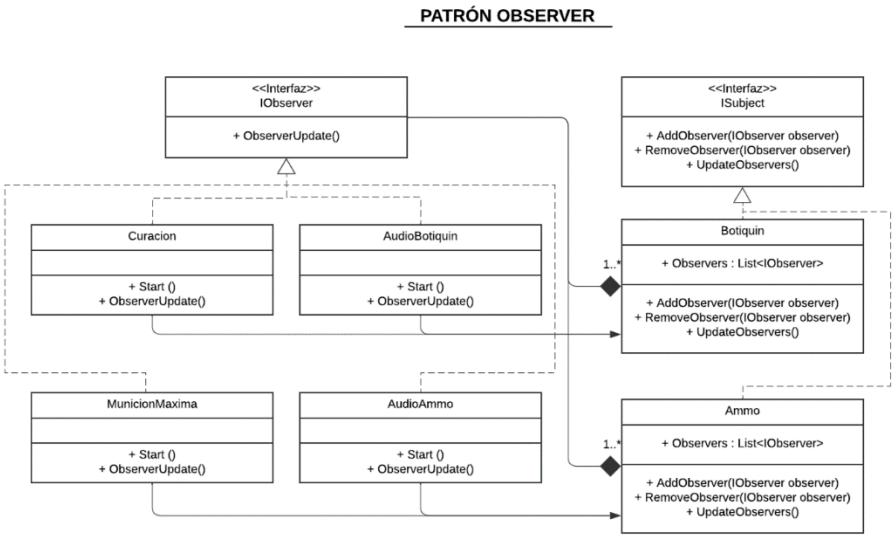

Ilustración 1 UML patrón observer

2.2. Command

Para desacoplar las entradas con respecto a unas acciones, se ha implementado el patrón command, encapsulando así las acciones. Estos comandos afectarán a 3 diferentes receivers, los cuales, implementarán la acción. Estos receivers se dividen en:

- IPlayerReceiver: para acciones del jugador o cualquier personaje controlable.
- IArmaReceiver: para acciones de las armas.
- IUIReceiver: para acciones que involucren la interfaz.

Para recibir las entradas, se implementa "Input Manager" que hereda de Monobehaviour y se encarga de generar los comandos con su respectivo receiver dependiendo de la entrada por teclado. Posteriormente mediante un CommandManager se ejecutan los diferentes comandos.

Los comandos integrados según su entrada son los siguientes:

- IPlayerReceiver:
    - Tecla "Espacio": Ejecuta el comando Jump, el cual permite al jugador saltar.
    - Teclas "A" o "D": Ejecuta el comando Move, el cual permite al jugador moverse a derecha e izquierda.
    - Teclas "W" o "S": Ejecuta el comando MoveForward, el cual permite al jugador moverse hacia delante o atrás.
    - Tecla "Left Shift": Ejecuta el comando Sprint, el cual permite al jugador correr.

- IArmaReceiver:
    - Click izquierdo: Ejecuta el comando Shoot, a lo cual, la respectiva arma dispara.
    - Tecla "R": Ejecuta el comando Reload para recargar el arma.

- IUIReceiver:
    - Tecla "Escape": Ejecuta el comando Ajustes, a lo que se abre la pestaña de ajustes.

Por último, los receivers que implementan la interfaz en el contexto del juego se tratan de Player, Arma y ajustes

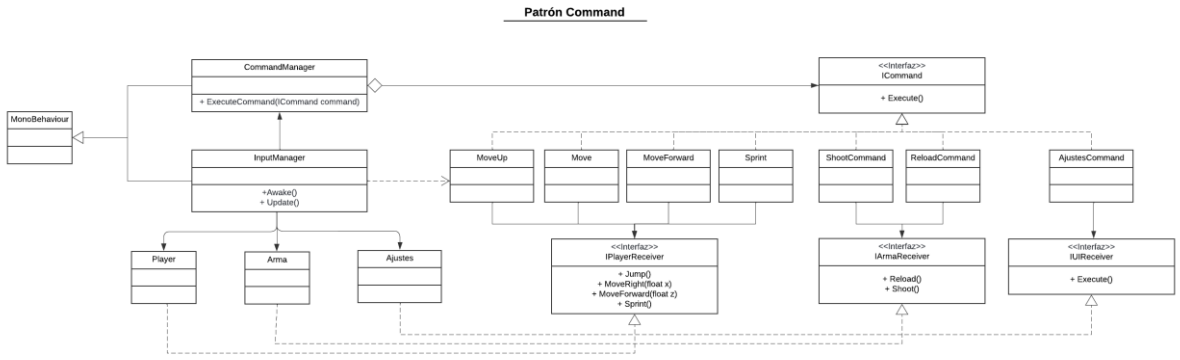

Ilustración 2 UML patrón command

2.3. ObjectPool y Prototype

Para evitar la fragmentación y hacer un mejor uso de la memoria se ha implementado un ObjectPool, el cual, permite aumentar el tamaño, a lo que, para volver a reducir el tamaño, se destruyen los objetos sobrantes y se eliminan del ObjectPool. El ObjectPool, por otro lado, implementa el patrón Prototype, para clonar los objetos.

Todo ObjectPool, contiene un grupo de IPooleableObjects, los cuales, son los objetos que van a ser activados, desactivados y clonados en el ObjectPool. En el juego, se encuentran los siguientes IPooleableObjects:

- Bullet: Se trata de las balas que dispara el arma, este se desactiva cuando colisiona con un enemigo o una pared, o si desciende hasta cierta altura, por si se pierde la bala. Para limpiar la bala, se deja la posición, rotación y velocidad en 0.
- Zombie: Se trata de los zombis que aparecen en escena, estos se desactivan cuando cambian al estado DeathState, es decir, cuando tienen una vida menor que 0. Para limpiarlo, se deja la posición, rotación, vida y velocidad en 0, además de desactivar la colisión.
- Botiquín: Se trata de uno de los sujetos implementados con el patrón observer. Este se desactiva cuando el jugador entra en contacto con él. Además, se trata de un deep copy, ya que el método Clone de Prototype copia incluso sus observadores. Para limpiarlo se deja la posición y la rotación en 0.
- Ammo: Se trata de uno de los sujetos implementados con el patrón observer. Este se desactiva cuando el jugador entra en contacto con él. Además, se trata de un deep copy, ya que el método Clone de Prototype copia incluso sus observadores. Para limpiarlo, se deja la posición y la rotación en 0.

Por otro lado, el juego tiene unos clientes, los cuales, implementan el mismo ObjectPool y se encargan de indicar cuando activar o clonar los IPooleableObjects. Estos clientes son:

- Arma: Cuando se ejecuta el comando Shoot, clona o activas balas en el ObjectPool.
- ZombiesRespawn: Cada 3 segundos activa o clona un zombi.
- Zombie: Cuando un zombi muere, tiene una probabilidad de clonar o activar un Botiquín o una Munición, para ello se utilizan unos mediadores con métodos específicos para Botiquín o para Munición. Estos mediadores se tratan de Power Ups Pool y BotiquinPool.

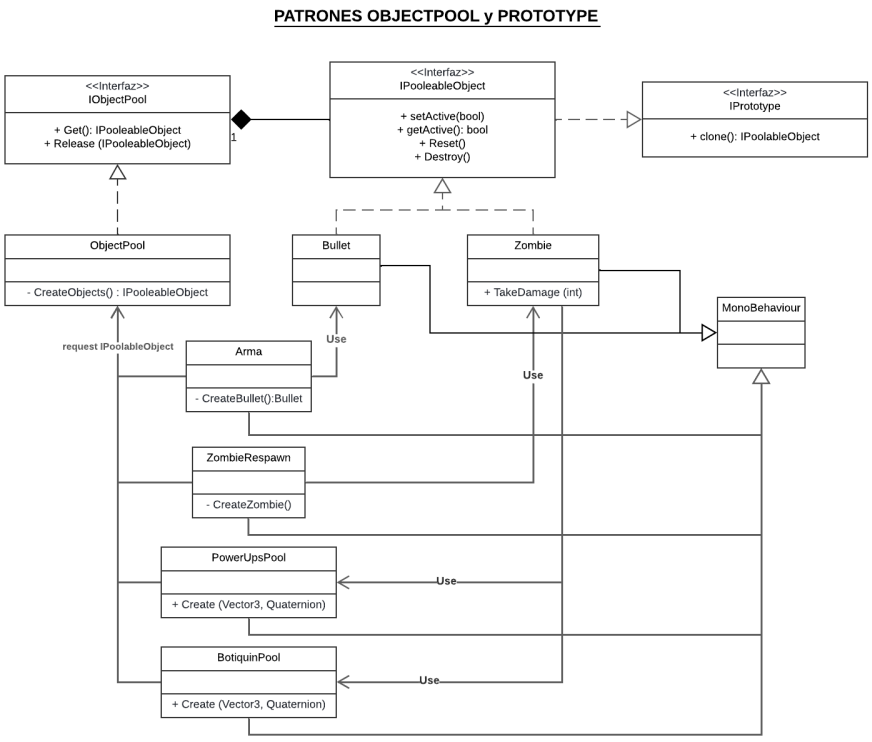

Ilustración 3 UML patrón objectpool y prototype

2.4. State

Para los estados de los zombis se ha implementado el patrón State. Esto permite al zombi cambiar de un estado a otro dependiendo de ciertos eventos. Los estados son los siguientes:

- IdleState: Estado en el que el zombi se queda inmóvil durante 3 segundos, a lo que al pasar el tiempo se elige al azar entre un 50% entre los estados de caminar y correr.
- WalkState: Estado en el que el zombi implementa una velocidad aleatoria en un umbral considerado para caminar. Posteriormente el zombi sigue al jugador hasta alcanzar una distancia de 1.5 cambiando de estado a AttackState.
- RunState: Estado en el que el zombi implementa una velocidad aleatoria en un umbral considerado para correr, una velocidad mayor a caminar. Posteriormente el zombi sigue al jugador hasta alcanzar una distancia de 2.5 cambiando de estado a AttackState.
- AttackState: Estado en el que el zombi si se encuentra a una distancia dentro de un umbral, ataca al jugador. Por otro lado, si no se encuentra dentro del umbral, vuelve al anterior estado del que procede, pudiendo ser WalkState o RunState.
- DeathState: Estado en el que el zombi se queda inmóvil y se desactiva del ObjectPool tras 5 segundos.

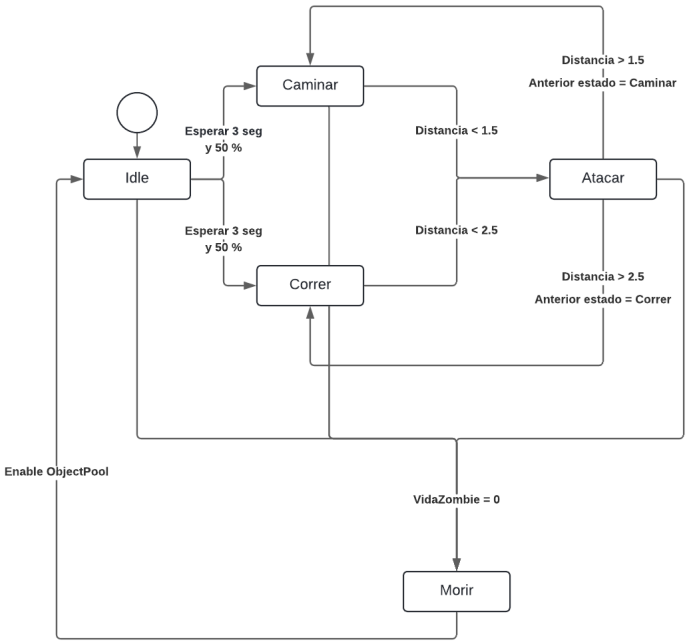

Ilustración 4 Diagrama de estados

Para su implementación el zombi tendrá un IState, con el cual empezará en IddleState. Los IState tendrán los métodos Enter, para cuando entra en el estado, Update, para las acciones a hacer mientras se encuentra en ese estado, y Exit, para cuando se sale del estado. El zombi, implementa un método SetState, que permite cambiar a otro estado, ejecutando el Exit del anterior estado y posteriormente el Enter del nuevo estado. Además, en el método Update de zombi se ejecutará constantemente el Update del estado actual. Por otro lado, para poder cambiar de estado, los propios estados reciben como parámetro el zombi, para así implementar el método SetState cuando se desee cambiar el estado. Por último, para cambiar de estado a DeathState, se trata del propio contexto quien cambia a este estado.

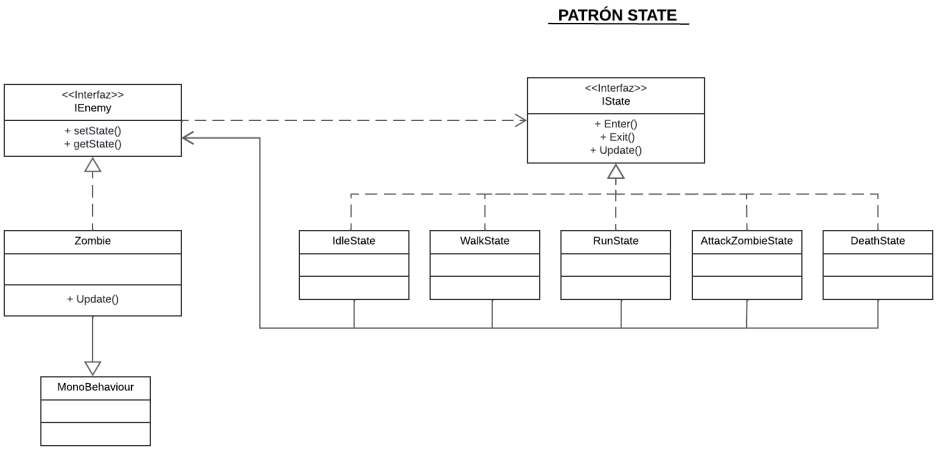

Ilustración 5 UML patrón state

2.5. DirtyFlag

Para poder cargar la posición del jugador y sus balas de una partida anterior se ha implementado el patrón DirtyFlag, el cual, guardara los datos únicamente cuando se modifique o cada 15 segundos por si se cierra el juego.

3. Assets Propios

Para el juego se han modelado diferentes assets mediante 3Ds Max, estos assets son los siguientes, además de otros elementos que también han sido realizados por nosotros:

- Bala
- Caja de munición
- Botiquín
- Niebla/Partículas de Unity
- Todas las interfaces
- Audio de pasos al caminar

4. Referencias y elementos de terceros

Zombies – Asset Store de Unity, Hungry Zombie

Boss final – Asset Store de Unity, Creep Horror

Arma – Asset Store de Unity, Free FPS Weapon

Escenario – Asset Store de Unity, Industrial Set v2.0 - Dimitrii Kutsenko

Efectos de Sonido – Pixabay

Videos – State, Disparo , Barra de vida , Texto flotante

5. Créditos

Luís Javier Moreno Rojas: diseño e implementación de la interfaz de juego, participación en la programación del patrón Observer y Command.

Elena Milara Mencía: modelado de assets que no se han podido implementar, y ejecución del patrón DirtyFlag.

David Lozano Varona: Implementación de los patrones Observer, Command, State, Object Pool y Prototype, además de algunas funcionalidades básicas y elementos estéticos.

Cristina Martínez Otto: diseño e implementación de las interfaces de inicio, muerte, victoria, menú, ayuda, créditos. Modelado de los assets propios, botiquín, caja de munición y balas. Participación en la implementación de los patrones observer, command, objectpool, prototype y state. Búsqueda e implementación de los efectos sonoros.

Gustavo Herrero Lapuente: participación en la implementación del patrón state.

6. Tabla de Requisitos

ID	FUNCIONALIDAD	IMPLEMENTACIÓN
RF0001	Implementación de Command para desacoplar las entradas y encapsular las acciones. Diagrama de clases, Diagrama Command propio.	**Assets**: Scripts/Patterns/Command/Interfaces/* Scripts/Patterns/Command/Commands/* Scripts/Patterns/Command/Components/* Scripts/Patterns/ObjectPool/Components/Arma
RF0001	Implementación del patrón Object Pool para el mejor uso de memoria con objetos clonados. Los objetos se tratan de Zombie, Bullet, Botiquin, Ammo.	**Assets**: Scripts/Patterns/ObjectPool/Interfaces/* Scripts/Patterns/ObjectPool/Components/* Scripts/Patterns/ObjectPool/ Components/PooleableObjects/* Scripts/Patterns/Observer/Sujetos/Ammo Scripts/Patterns/Observer/Sujetos/Botiquin
RF0001	Implementación del patrón Prototype para clonar balas, zombis, botiquines y cajas de munición.	**Assets**: Scripts/Patterns/ObjectPool/ Components/PooleableObjects
RF0002	El juego se compone de un único nivel con 5 rondas en las que se tendrán que matar a los distintos zombis	**Assets**: Scene/Juego
RF0003	La condición de victoria se da cuando mata a todos los zombis de las 5 rondas.	**Assets**: Scripts/Patterns/ObjectPool/ Components/ZombieRespawn
RF0004	La condición de fracaso se da cuando los zombis le quitan toda la vida al jugador.	**Assets**: Scripts/Patterns/Command /Components/Player

RF0005	Nuestra interfaz de créditos está presente en una escena aparte.	**Assets**: Scene/ InterfazInicio
RF0006	Nuestra interfaz de créditos está presente en una escena aparte.	**Assets**: Scene/Credits
RF0007	Nuestro menú de pausa se corresponde con el menú de juego al cual se accede pulsando escape mediante el patrón command.	**Hierrarchy de la escena juego**: Menu/* **Assets**: Scene/InterfazInicio Scene/Juego Scene/Interfaces/*
RF0008	Desde el menú principal se accede a la partida a través del botón “Play”	**Assets:** Scene/ InterfazInicio Scene/Juego Scene/Interfaces/Scripts/BotonPlay
RF0009	Desde el menú principal se puede acceder a la pantalla de créditos pulsando el botón correspondiente y de este volver al inicio con el botón de la casa.	**Assets:** Scene/ InterfazInicio Scene/credits Scene/Interfaces/* Scene/Interfaces/Scripts/BotonPlay Scene/Interfaces/Scripts/BotonMenu
RF0010	Se puede salir del juego desde el menú principal pulsando el botón de la puerta	**Assets**: Scene/ InterfazInicio Scene/Interfaces/Scripts
RF0011	Se puede salir al menú principal desde el juego accediendo al menú que sale al pulsar la tecla escape y luego dando a “Quit”.	**Hierrarchy de la escena juego**: Menu/* **Assets**: Scene/InterfazInicio Scene/Interfaces/* Scene/Interfaces/Scripts/BotonMenu
RF0010	Implementación del patrón State para el control de los estados del zombie. Posibles estados del zombie: IdleState, WalkState, RunState, AttackState, DeathState.	**Assets**: Scripts/Patterns/State/Interfaces Scripts/Patterns/State/States Scripts/Patterns/ObjectPool/ Components/PooleableObjects/Zombie
RF0010	Implementación del patrón DirtyFlag para guardar la posición del jugdor, rotación, balas y vida de una partida anterior.	**Assets**: Scripts/Patterns/DirtyFlag

RF0013	Los modelados de los Power Ups y las balas son de creación propia, y el sonido de caminar.	**Assets**: Modelados_Propios/* Musica/*
RF0014	Se han intentado hacer las interfaces lo más atractivas posibles y de tal forma que sea consistente e intuitiva, la mayoría de estas son escenas aparte.	**Assets:** Scene/Help Scene/Credits Scene/InterfazInicio Scene/Juego/*
RF0015	Interfaz de ayuda que se accede desde el menú principal.	**Assets**: Scene/Help