Framework de comportamientos de enemigos para videojuegos 2D
An enemy behaviour framework for 2D videogames

Trabajo de Fin de Grado
Curso 2024–2025

Autor
Francisco Miguel Galván Muñoz
Cristina Mora Velasco

Director
Guillermo Jimenez Díaz

Grado en Desarrollo de Videojuegos
Facultad de Informática
Universidad Complutense de Madrid

Framework de comportamientos de enemigos para videojuegos 2D
An enemy behaviour framework for 2D videogames

Trabajo de Fin de Grado en Desarrollo de Videojuegos

Autor
Francisco Miguel Galván Muñoz
Cristina Mora Velasco

Director
Guillermo Jimenez Díaz

Convocatoria: *Junio 2025*

Grado en Desarrollo de Videojuegos
Facultad de Informática
Universidad Complutense de Madrid

13 de Junio de 2025

Dedicatoria

A nuestras gatas por inventar el arte de convertir el dolor de un arañazo en la alegría de un beso
Y a nuestras familias por el apoyo incondicional mostrado en la consecución de este trabajo

Agradecimientos

A Guillermo por ser el mejor tutor de Trabajo de Fin de Grado posible y por habernos dado la oportunidad de cumplir un sueño al llegar a la consecución de este grado. A todo el resto del profesorado que nos ha enseñado tantas cosas y nos han acompañado estos años. Y sobre todo a nuestras familias que día tras día han estado con nosotros, ayudándonos a crecer, apoyándonos incondicionalmente y dándonos fuerzas para seguir.

Resumen

Framework de comportamientos de enemigos para videojuegos 2D

El arte de hacer un videojuego es la combinación homogénea de varias artes para crear una experiencia interactiva. Entre estas partes encontramos arte, sonido, programación y diseño. A la hora de hacer el arte, el ser humano siempre se ha ayudado de herramientas las cuales ha fabricado él mismo y le ha ayudado a llevar a cabo las tareas de manera más eficaz y precisa. A la hora de hacer un videojuego de cualquier tipo, el equipo detrás de la obra tiene que dejar clara dos ideas al jugador: el objetivo del juego y los obstáculos a los que se va a tener que enfrentar para alcanzarlo. Haciendo especial hincapié en el tipo de juegos analizados en este trabajo, el principal obstáculo serán enemigos, y para que estos supongan un desafío para el jugador deberán contar con una inteligencia artificial acorde al diseño del mismo.

El objetivo de este trabajo es crear una herramienta cuyo propósito sea generar enemigos para videojuegos de plataformas 2D. Para ello se creará un catálogo de componentes para Unity con los que poder crear, sin tener conocimientos previos de programación, inteligencias artificiales de enemigos. Este catálogo de componentes es el resultado de nuestro trabajo de investigación y posterior abstracción de comportamientos de enemigos en juegos de plataformas 2D.

Palabras clave

Inteligencia Artificial, IA, Unity, Enemigo, Maquinas de Estado, Estado, Sensor

Abstract

An enemy behaviour framework for 2D videogames

The art of making a video game is the homogeneous combination of various arts to create an interactive experience. Among these parts we find art, sound, programming and design. When it comes to making art, humans have always used tools which they have made themselves and which have helped them to carry out tasks more efficiently and accurately. When making a video game of any kind, the team behind the work has to make two ideas clear to the player: the objective of the game and the obstacles that the player will have to face to achieve it. With special emphasis on the type of games analyzed in this work, the main obstacle will be enemies, and for them to be a challenge for the player they must have an artificial intelligence according to the game design.

The objective of this work is to create a tool whose purpose is to generate enemies for 2D platform video games. For this purpose, a catalog of components for Unity will be created with which to create, without previous programming knowledge, enemy artificial intelligences. This catalog of components is the result of our research work and subsequent abstraction of enemy behaviors in 2D platform games.

Keywords

Artificial Inteligence,AI, Unity, Enemy, State Machine, State, Sensor

Capítulo 1

Introducción

"Los seres humanos no nacen para siempre el día en que sus madres los alumbran, sino que la vida los obliga a parirse a sí mismos una y otra vez"
— Gabriel García Márquez

1.1. Motivación

A lo largo de los años, los videojuegos han evolucionado considerablemente, llegando a convertirse en una parte muy importante del entretenimiento de hoy en día. Desde las sencillas primeras experiencias de las consolas hasta los complejos mundos actuales, los videojuegos han conquistado audiencias de todas las edades y culturas. A lo largo de esta evolución los enemigos han jugado un papel fundamental, haciendo que los videojuegos presentasen desafíos que atrapan a los jugadores. En el caso de los videojuegos 2D de plataformas, los enemigos son más que una simple oposición del jugador si no que son clave para mostrar la esencia del juego. El Pac-Man no sería lo mismo sin sus fantasmas ni el Super Mario sin sus Goombas.
Diseñar enemigos, especialmente en este tipo de juegos, es una tarea compleja. No solo trata de darles cierta apariencia si no que tienen que tener unos comportamientos y características únicas. Esto implica que la persona encargada de esta tarea tiene que tener ciertos conocimientos en arte, diseño y programación. En la actualidad, este trabajo es realizado por personas de los campos anteriomente mencionados: artistas, diseñadores y programadores. En los últimos años han surgido herramientas que simplifican significativamente el trabajo de los diseñadores, pero muy pocas están centradas específicamente en el diseño de enemigos. El objetivo de estas herramientas es el de facilitar el trabajo de los diseñadores para, incluso sin tener ni idea de programación, tener la capacidad de crear enemigos totalmente funcionales.

1.2. Objetivos

Este trabajo tiene como objetivo principal el diseño y desarrollo de un herramienta en C sharp para el motor de videojuegos Unity, que simplifique y agilice

el proceso de creación de enemigos en juegos plataformas 2D. La herramienta contará con un catalogo de comportamientos fácil de manejar para cualquier persona independientemente de sus conocimientos de programación. El catálogo estará compuesto por tres categorías diferentes: acciones, sensores y eventos. Estos se definirán más adelante.
Para llevar a cabo el desarrollo de nuestra herramienta, vamos a documentarnos con herramientas similares y de como funcionan los enemigos en este tipo de juegos tomando como referencia títulos de renombre que serán presentados posteriormente.

1.3. Plan de trabajo

Para llevar a cabo este trabajo, se ha seguido la metodología ágil Scrum. Esta metodología, permite crear un flujo de trabajo enfocado en la iteración y continua mejora, asegurando un avance en el desarrollo eficiente y posibles adaptaciones frente a problemas detectados durante el proceso. El trabajo se dividirá en cuatro bloques: investigación y planificación, desarrollo de la memoria, desarrollo de la herramienta y pruebas con usuarios. Cada bloque a su vez se dividirá en subsecciones explicadas a continuación.

- Investigación y planificación:
  - Estudio del problema: En esta primera fase se realizará un estudio del estado del arte, centrado en el papel de los enemigos en los videojuegos, su importancia en la jugabilidad y las diferentes técnicas utilizadas para su diseño y comportamiento.
  - Selección y estudio de herramientas: Esta fase implicará un análisis comparativo de distintas técnicas y motores de videojuegos evaluando sus ventajas y desventajas, así como un estudio de su funcionamiento y arquitecturas.
  - Diseño de la herramienta: En esta etapa, se definirá la arquitectura de la herramienta propuesta, describiendo las técnicas empleadas, esquemas de funcionamiento y organización de elementos principales.
- Desarrollo de la memoria:
  - Redacción inicial: Es esta fase del trabajo se procederá a la redacción inicial de los contenidos cubriendo todos los puntos especificados en el índice.
  - Revisión y corrección: Una vez completada la redacción inicial, se realizarán las correcciones necesarias tras revisar exhaustivamente el documento.
  - Conclusiones y trabajo futuro: Tras finalizar los desarrollos y las pruebas de usuario, se redactarán las conclusiones obtenidas en base a los resultados y se detallarán los posibles pasos a seguir en un futuro.
- Desarrollo de la herramienta:

- Implementación de funcionalidades principales: En esta etapa se implementarán las funcionalidades principales de los movimientos básicos incluyendo la integración con sensores y emisores permitiendo la interacción entre ellos.
- Implementación de ayuda visual: Se desarrollarán ayudas visuales destinadas a servir como referencias para los diseñadores, incluyendo elementos gráficos que faciliten la comprensión de los comportamientos. Además se implementará una herramienta que permita elegir y configurar de una forma más sencilla los componentes.
- Pruebas y depuración: Se llevará a cabo un proceso iterativo de pruebas que aseguren la funcionalidad de la herramienta, corrigiendo los errores detectados durante su implementación.

- Pruebas con usuarios:
  - Primera fase de pruebas: Se harán pruebas con usuarios que no hayan probado la herramienta antes, siguiendo un plan de pruebas especificado en el apartado .evaluación con usuarios". Las pruebas estarán centradas en: detectar posibles errores en las funcionalidades principales, validar la funcionalidad y evaluar la usabilidad y claridad.
  - Segunda fase de pruebas: Tras implementar mejoras recibidas del feedback de la primera prueba, se llevará a cabo una segunda prueba de verificación.
  - Corrección y resultados: Tras analizar los resultados de cada fase de prueba,se documentarán los errores y dificultades encontrados. Tras esto se procederá a implementar las correcciones necesarias para mejorar los resultados.

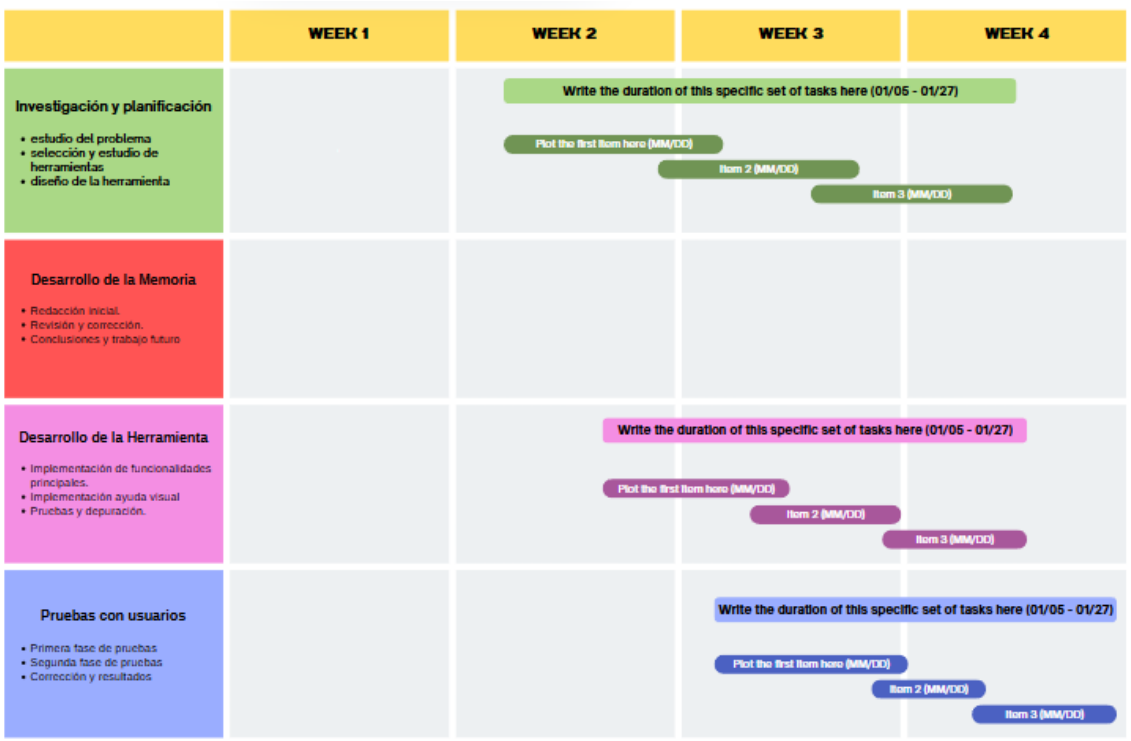

Figura 1.1: Tabla de Gantt

Capítulo 2

Estado de la Cuestión

En este capítulo se hará una investigación sobre las técnicas, herramientas y formas de crear inteligencias artificiales para enemigos. Para ello vamos a comenzar haciendo un recorrido por todo lo relacionado con la inteligencia artificial en general y cómo se usa esta en videojuegos 2D de plataformas, ya sea para crear un NPC, un enemigo o para optimizar alguna función más a bajo nivel del juego. Se mencionarán además herramientas para conseguir los fines descritos anteriormente y se hablará de algunos motores de videojuegos que han inspirado en cierta manera algunos aspectos de nuetras herramienta.

2.1. Introducción a la Inteligencia Artificial

La Inteligencia Artificial (IA) es la capacidad que tiene un sistema o software de realizar tareas diferentes entre sí de manera autónoma aplicando reglas, algoritmos o patrones de aprendizaje automático, simulando así comportamientos propios de la inteligencia humana. El potencial de la IA hoy día y por lo que está generando tanto furor es su capacidad de autonomía, ya que no solo sigue reglas, si no que tiene la capacidad de tomar decisiones
En el ámbito de los videojuegos, la IA se ha hecho paso a base de demostrar su gran capacidad de adaptación a contextos diversos como la adaptación del Alien en *Alien Isolation*[1] , la manera en la que puede generar contenido procedural para juegos roguelike como *Hades*[2] o ,por último, entrenar una IA para que se acerque lo máximo posible al comportamiento de un humano como en la serie *Forza Motosport*[3], usando redes neuronales.

[1] `https://www.gamedeveloper.com/design/the-perfect-organism-the-ai-of-alien-isolation`
[2] `https://hades.fandom.com/es/wiki/Hades_(juego)`
[3] `https://forza.fandom.com/wiki/Forza_Wiki`

2.2. Tecnicas

Es muy importante a la hora de escoger una técnica para modelar la IA de una entidad el saber ciertos factores para tomar la decisión correcta, como por ejemplo la complejidad esperada del comportamiento, la escalabilidad y flexibilidad de la técnica, si queremos invertir mucho tiempo en implementar estas técnicas o queremos algo rápido de hacer y funcional, los recursos que consumen...

A continuación, se presentarán una serie de técnicas que se usan en los videojuegos para crear IAs.

2.2.1. Maquinas de estado finitas

Las máquinas de estado finitas, en inglés finite state machine, con siglas FSM, son un modelo matemático que representan un número finito de estados y una serie de transiciones entre ellos.

Una FSM se representa como un grafo, siendo este una representación abstracta de un conjunto de objetos, eventos, acciones o propiedades conectados entre sí, siendo estos elementos nodos (estados) que realizan acciones y comprueban la posibilidad de que haya que cambiar de nodo.
En el ámbito de los videojuegos, las FSM son el conjunto de estados que puede tomar una entidad y la forma de llegar a estos, teniendo en cuenta que solo puede haber un estado activo en cualquier instante.

El primer videojuego documentado que utilizó FSM para implementar la lógica de juego fue *Spacewar!(1961)*[4] desarrollado en el MIT por Steve Russell. Este videojuego implementaba una lógica basada en estados para manejar el comportamiento de las naves, la detección de colisiones y la física del juego. Aunque no usaba una implementación formal de máquinas de estado, sí modelaba cambios entre estados bien definidos, como el movimiento de las naves o la activación de los disparos.

Pac-Man[5] es un videojuego en el que el jugador controla un personaje amarillo en forma de círculo con una boca que se abre y cierra constantemente y fue lanzado en 1980 por la compañía japonesa Namco (actual Bandai Namco). El objetivo de este videojuego es el de recorrer un laberinto e ir comiendo todos los puntos mientras evitamos cuatro fantasmas hasta que comemos una píldora de poder que nos hace invulnerable y nos da la capacidad de comer a los fantasmas. Estos huirán tras comernos la píldora.

La complejidad en la IA de Pac-Man es asombrosa ya que se le quiso dar profundidad al juego haciendo que cada fantasma tuviera una personalidad diferente. Para ello se implementó una máquina de estado por fantasma haciendo que la forma

[4] https://www.ijarsct.co.in/Paper2062.pdf
[5] https://pacman.fandom.com/es/wiki/Pac-Man_Wiki:Portada

en la que estos interactúan con el entorno sea ligeramente diferente. A continuación se enumerarán los fantasmas y sus formas de comportarse.

- Blinky: es el fantasma rojo y su papel es el de cazador, siendo su personalidad la más agresiva, hecho que se refleja en que es el único fantasma que comienza fuera de la casa de los fantasmas y que tras salir empieza a perseguir al jugador incansáblemente. Tiene otra característica propia, a medida que el jugador va comiendo bolitas, comienza a aumentar su velocidad.
- Pinky: como su nombre indica es el fantasma de color rosa. En japonés se llama *M*achibuse, el que tiende emboscadas. Pinky es el interceptor del juego por lo que va a tratar de cortar el camino del jugador. Es un fantasma relativamente rápido, por lo que calculará constantemente hacia donde se dirige el jugador para usar su velocidad para adelantarse y cortar el paso.
- Inky: el fantasma azul es el más impredecible de todos, ya que su función es la de adoptar temporalmente la personalidad de sus otros tres compañeros.
- Clyde: el fantasma naranja y el más tranquilo de todos. Suele ser el último en salir de la casa de los fantasmas y no intentará atrapar al jugador a no ser que este esté muy cerca de él. El resto del tiempo deambula por el mapa e intenta evitar al jugador.

Para ilustrar el funcionamiento del juego se usará la Figura 2.1 que representa una posible FSM para el jugador, lo que haría que las decisiones tomadas fueran lo más eficientes posibles en el momento.

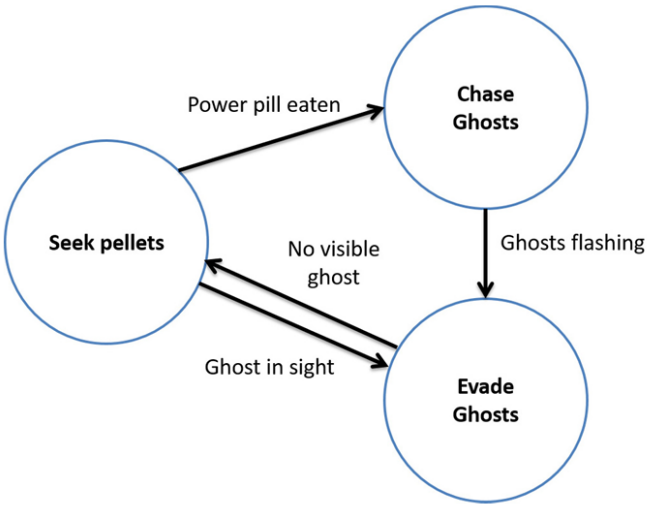

Figura 2.1: Comportamiento jugador Pac-Man, extraído del libro de Yannakakis y Togelius (2018)

Un punto en contra de las FSMs es que son muy inflexibles y estáticas, de manera que las posibilidades de escalado de la lógica son limitadas. También son algo predecibles una vez el jugador ha estudiado los estados y transiciones de una

entidad, este punto negativo puede paliarse implementado probabilidades o reglas que no estén tan claras a la hora de hacer las transiciones.

2.2.2. Árboles de comportamiento

Un árbol de comportamiento o Behavior Tree (BT) es un sistema similar a las FSMs ya que contamos con nodos y transiciones y solo uno de esos nodos, que ahora pasan a ser comportamientos en lugar de estados, puede estar activo al mismo tiempo.
En esencia, es un árbol de nodos que se organizan jerárquicamente y que atienden a unas normas que controlana el flujo de cambios de nodo.

La principal ventaja respecto a las FSMs es su modularidad, la capacidad que tiene un sistema de dividir la lógica del comportamiento en piezas independientes y reutilizables, pudiendo agrupar estas piezas en grupos que a su vez funcionan como una pieza.
Su facilidad para ser diseñados y testeados han hecho que los árboles de comportamiento se conviertan en una opción real para modelar IA en la industria del videojuego, con juegos como *Bioshock (2K Games, 2007)*[6] y *Halo 2 (Microsoft Game Studios, 2004)*[7] como referencias en el uso de BTs.

Un ejemplo no tan conocido de uso de BTs en la industria del videojuego es *Spore (2008, Maxis)*. Spore es un videojuego en el que el jugador va a comenzar creando una célula y va encarnarla durante todo el proceso de su evolución hasta que esta se convierta en un ser mucho más complejo llegando incluso a construir una civilización muy avanzada. La inteligencia artificial de las entidades que nos rodean en este videojuego están fundamentadas en BTs.

La gran diferencia de como usa BTs Spore a los juegos anteriormente mencionados es que Spore separa el concepto de *decider* de *behavior*. Los BTs de Halo 2 están compuestos por *behaviors* ya sean estos comportamientos grupales, en los que se elige entre varias opciones, o individuales, que ejecutan acciones específicas, e impulsos que son los saltos que se dan entre comportamientos y que se toman dependiendo de una prioridad, que a futuro resulta en problemas de escalabilidad. Para solucionar este problema, en Maxis deciden unificar el concepto de impulso y comportamientos grupales bajo el nombre de *deciders*, dejando completamente separado el concepto de *behavior* y permitiendo que estos puedan ser reutilizados en múltiples lugares del árbol, asegurando la escalabilidad.

La Figura 2.2 es un ejemplo de un BT sacado de las documentación[8] que hay publicada del juego. Un punto importante a tener en cuenta es que si un nodo tiene más de un hijo, estos se comprueban si son o no elegidos en un orden determinado hasta que un de los *deciders* acepte la petición del padre o ninguno acepte y se

[6] `https://es.wikipedia.org/wiki/BioShock`
[7] `https://www.gamedeveloper.com/programming/gdc-2005-proceeding-handling-complexity-in-the-i-ha`
[8] `https://chrishecker.com/My_Liner_Notes_for_Spore/Spore_Behavior_Tree_Docs`

vuelva al nodo actual volviendo a repetir el proceso.

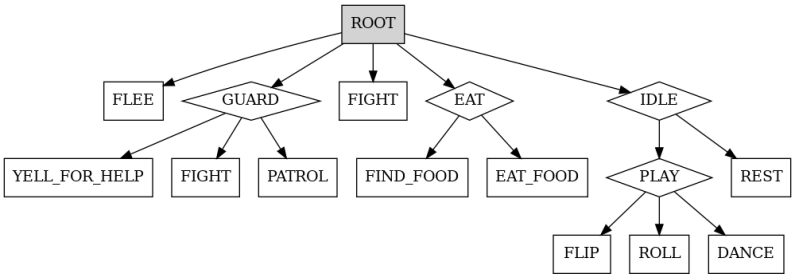

Figura 2.2: Ejemplo BT, Documentación Spore

2.2.3. Goal-Oriented Action Planning

GOAP es un sistema basado en planificación de acciones. En lugar de definir comportamientos fijos como hemos visto anteriormente, la entidad analiza la situación y construye un plan óptimo para alcanzar el objetivo designado. Para que un sistema de esta índole pueda llevarse a cabo deben tenerse una serie de elementos clave.

- Blinky: es el fantasma rojo y su papel es el de cazador, siendo su personalidad la más agresiva, hecho que se refleja en que es el único fantasma que comienza fuera de la casa de los fantasmas y que tras salir empieza a perseguir al jugador incansáblemente. Tiene otra característica propia, a medida que el jugador va comiendo bolitas, comienza a aumentar su velocidad.

- Pinky: como su nombre indica es el fantasma de color rosa. En japonés se llama *M*achibuse, el que tiende emboscadas. Pinky es el interceptor del juego por lo que va a tratar de cortar el camino del jugador. Es un fantasma relativamente rápido, por lo que calculará constantemente hacia donde se dirige el jugador para usar su velocidad para adelantarse y cortar el paso.

- Inky: el fantasma azul es el más impredecible de todos, ya que su función es la de adoptar temporalmente la personalidad de sus otros tres compañeros.

- Clyde: el fantasma naranja y el más tranquilo de todos. Suele ser el último en salir de la casa de los fantasmas y no intentará atrapar al jugador a no ser que este esté muy cerca de él. El resto del tiempo deambula por el mapa e intenta evitar al jugador.

(Comentario: https://github.com/crashkonijn/GOAP herramienta que podriamos analizar)

2.2.4. Aprendizaje por Refuerzo (RL) o Redes Neuronales

2.3. Análisis herramientas

2.3.1. Behaviour Bricks

2.3.2. Play Maker

2.3.3. Animator

2.4. Motores de videojuegos

2.4.1. Unity

2.4.2. Unreal

2.4.3. Godot

2.4.4. GameMaker

2.4.5. Construct 3

2.5. Conclusiones

En el estado de la cuestión es donde aparecen gran parte de las referencias bibliográficas del trabajo. Una de las formas más cómodas de gestionar la bibliografía en LaTeX es utilizando **bibtex**. Las entradas bibliográficas deben estar en un fichero con extensión *.bib* (con esta plantilla se proporciona el fichero biblio.bib, donde están las entradas referenciadas más abajo). Cada entrada bibliográfica tiene una clave que permite referenciarla desde cualquier parte del texto con los siguiente comandos:

- Referencia bibliografica con cite: Bautista et al. (1998)
- Referencia bibliográfica con citep: (Oetiker et al., 1996)
- Referencia bibliográfica con citet: Krishnan (2003)

Es posible citar más de una fuente, como por ejemplo (Mittelbach et al., 2004; Lamport, 1994; Knuth, 1986)

Después, LaTeXse ocupa de rellenar la sección de bibliografía con las entradas **que hayan sido citadas** (es decir, no con todas las entradas que hay en el .bib, sino sólo con aquellas que se hayan citado en alguna parte del texto).

Bibtex es un programa separado de latex, pdflatex o cualquier otra cosa que se use para compilar los .tex, de manera que para que se rellene correctamente la sección de bibliografía es necesario compilar primero el trabajo (a veces es necesario compilarlo dos veces), compilar después con bibtex, y volver a compilar otra vez el trabajo (de nuevo, puede ser necesario compilarlo dos veces).

Capítulo 3

Descripción del Trabajo

En este capitulo se describe el framework de enemigos creado, mediante el diseño de componentes más sencillos. Primero se describirá el contexto y por tanto la utilidad de la herramienta, aquí se detallaran juegos analizados y sus características en común. Después se explicará que elementos la componen y por último se detallarán algunos ejemplos de uso.

3.1. Contexto

Como se señala en Patashnik (2014), los enemigos bien diseñados son clave para evitar que los niveles queden planos y, en consecuencia, aburridos para el jugador.

3.1.1. Enemigo

Los enemigos son entidades programadas para enfrentarse al jugador y crear desafíos dentro del juego. Su diseño incluye características, comportamientos y habilidades diseñadas para interactuar con las mecánicas del juego y contribuir a la experiencia del jugador.

3.1.2. Análisis de enemigos en videojuegos

El análisis de diversos documentos muestra que la mejor forma de hacer que un enemigo destaque y de un gran potencial al juego es que tenga unos comportamientos únicos. Estos comportamientos han ido evolucionando con el tiempo volviéndose cada vez más sofisticados. Cada enemigo se define mediante una forma única de un conjunto de componentes bien seleccionados que permiten identificarlo y recordarlo. Con dicha sofisticación, ha aumentado la complejidad del trabajo en el diseño. Para solucionarlo hemos propuesto una herramienta con la composición indicada a continuación

3.2. Composición

En este trabajo, se ha decidido entender como enemigo a cualquier entidad que pueda repercutir de forma negativa en el jugador, esto significa que no se limita el concepto de enemigo a figuras típicas, como monstruos o soldados hostiles, sino que se amplía su definición a toda entidad que suponga un riesgo, dificultad o amenaza para el progreso o el bienestar del jugador dentro del juego, como pueden ser pinchos, lava o gotas de ácido. Además separamos cada enemigo por comportamientos diferentes, implicando que elementos que clásicamente aparecen en conjunto, como la tubería y la gota de ácido o la bala y el pistolero, serán considerados como dos enemigos distintos.

3.2.1. Máquina de estados finita

La Máquina de Estados Finita (FSM, Finite State Machine) es el núcleo de la lógica que define el comportamiento de los enemigos en nuestro diseño. Cada enemigo tiene su propia FSM, configurada específicamente para representar sus patrones de acciones, reacciones y relaciones en el juego. La FSM organiza el comportamiento de los enemigos mediante estados y transiciones: Los estados agrupan las acciones que el enemigo puede realizar en un momento dado. Las transiciones permiten cambiar de un estado a otro y son activadas por sensores y emisores.
Estos conceptos (estados, sensores, emisores y eventos) se desarrollan con mayor detalle en los apartados siguientes.

El objeto enemigo se define mediante una máquina de estados finita y un script que contiene información relevante. A continuación, se describen ambos conceptos y sus propiedades.

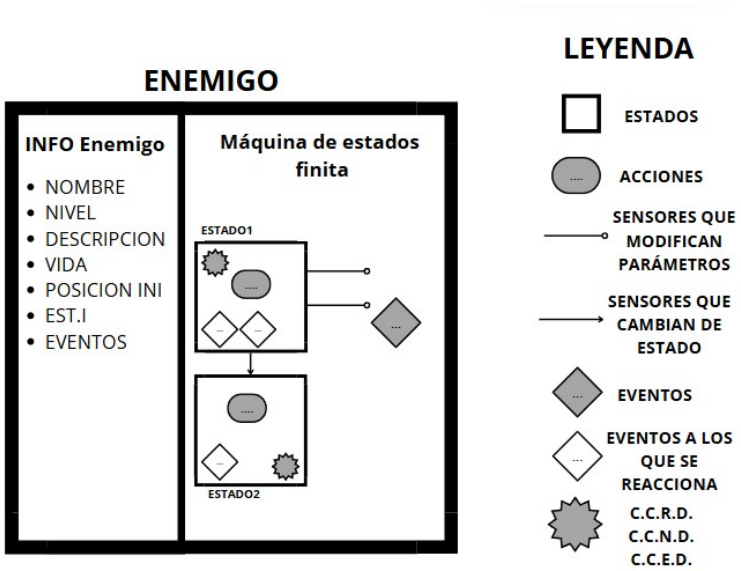

Figura 3.1: Enemigo General

3.2.2. Estado

Un estado dentro de la FSM se define como un conjunto de acciones y la posible activación de sensores y emisores, que en conjunto describen el comportamiento del enemigo. Las acciones pueden incluir uno o más tipos de movimiento compatibles entre sí, así como la capacidad de generar otros enemigos mediante un spawner. Es importante destacar que la muerte del enemigo no se considera un estado dentro de la FSM. Este concepto se desarrolla en profundidad en el apartado de eventos.

Además cada estado contiene una lista que definen las dimensiones, tiempo de recarga y valor de daño de las cajas de colisión (explicadas en la sección de Daño).

3.2.3. Acciones (Actuadores)

Hace referencia a un conjunto de movimientos y habilidades que definen lo que un enemigo puede hacer en el videojuego. Esto incluye distintos tipos de desplazamientos y la capacidad de crear otros enemigos de forma independiente mediante spawners. Estas habilidades no siempre son compatibles entre sí, teniendo una tabla en la que se indicaran las relaciones entre ellas. Además no es necesario utilizar siempre todas las acciones de forma que a veces un enemigo podrá realizar un tipo de movimiento o usar un spawner de manera exclusiva, mientras que en otras podrá combinar varias habilidades según su diseño y complejidad. Esto permite adaptar las capacidades de los enemigos para diferentes situaciones en el juego.

Spawner

Capacidad que poseen los enemigos para generar otros enemigos independientes, es decir, poder crear nuevas unidades que actúan de forma autónoma dentro del juego.
Los spawners presentan características únicas dentro del sistema de enemigos, ya que su objetivo no es solo enfrentarse al jugador, sino también aumentar el número de amenazas de manera continua o en función de ciertas condiciones. Implementar un spawner requiere establecer ciertas reglas de generación de enemigos, tales como la frecuencia de aparición, el número máximo de enemigos generados o si las unidades generadas son temporales o persistentes.

Movimiento

Podemos definir el término movimiento como el desplazamiento o cambio de posición de un enemigo dentro del juego. Sin embargo, el concepto de movimiento también puede aplicarse en el caso de enemigos que permanecen en una posición fija, haciendo referencia a la ausencia de éste. Los movimientos son fundamentales para definir el comportamiento de los personajes, ya que permiten la interacción con el jugador y el entorno. A continuación se describen en una tabla todos los movimientos que se han considerado esenciales y en la figura **??** podemos ver la compatibilidad entre ellos

Tipo movimiento	Descripción	Atributos	Ejemplo
Movimiento Horizontal	Desplazamiento simple de lado a lado en el escenario.	Velocidad, aceleración	Woomba
Movimiento Vertical	Desplazamiento simple hacia arriba y abajo.	Velocidad, aceleración	Planta Carnívora
Quieto	El enemigo se mantiene en el sitio sin ninguna modificación aparente.	—	
Movimiento hacia un punto	El enemigo va hacia un punto. Puede ser a un punto fijo (estático) o a un punto en movimiento (dinámico).	Punto a seguir, actualizar, lista de puntos a seguir, velocidad, zona de puntos aleatorios.	N/A
Circular/Rotación	El enemigo describe trayectorias circulares a través de su movimiento.	Radio, velocidad angular, punto de rotación, sinusoidal, aceleración, rango de giro.	

Tabla 3.1: Tabla Movimientos

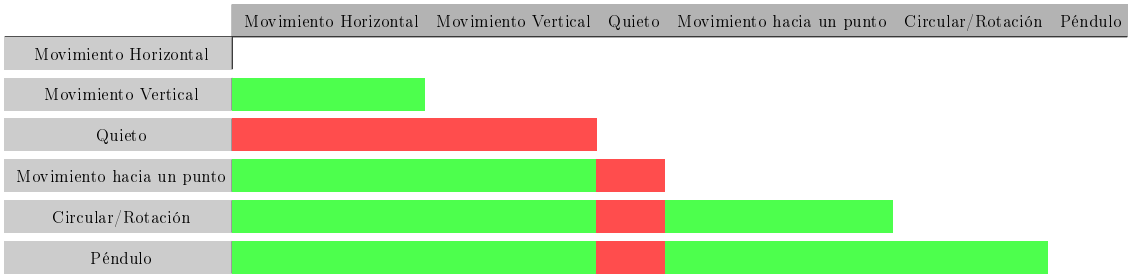

Tabla 3.2: Matriz de compatibilidad de movimientos

3.2.4. Sensores y Emisores

Podemos definir el término sensor como el elemento necesario para medir variables exteriores y enviar la información al enemigo. Los sensores permiten producir eventos.
Definimos Emisores como anónimo de sensor, es decir, es aquel elemento que envía información del enemigo al exterior. Para explicar su funcionamiento se proponen algunos ejemplos.

3.2.5. Daño

Cuando se habla de daño, el concepto de enemigo se deja a un lado y se habla de volúmenes de colisión.
Se define daño como la consecuencia negativa que recibe una entidad con respecto a su vida. Se produce como la consecuencia de que un volumen de colisión que emite

Sensor/Emisor	Descripción	Atributos	Ejemplo
Colisión	Encargado de proceder al cambio de estado en caso de colisión. Puede especificarse la capa con la que se debe colisionar para proceder al cambio.	Capa con la que se colisiona.	
Distancia	Encargado de medir la distancia entre un punto y otro, ya sea en uno de los ejes (X o Y) o la magnitud de la misma, y que en caso de que se llegue a una distancia deseada se procederá al cambio de estado.	Posición desde donde se detecta. Distancia Transición,Ignora paredes,Entidad Objetivo,Eje Objetivo: X o Y (en su defecto magnitud).	
Tiempo	Encargado de medir el tiempo transcurrido desde un inicio. Funciona a modo de contador	TiempoTransición	
Daño	Encargado de detectar si algún volumen de colisión ha recibido algún impacto dañino	Volumen de colisión, recibe o no daño, aplica o no daño, daño persistente o no, .	
Estado Jugador	El enemigo cambiará de estado dependiendo de las características del jugador	Struct Estado Jugador. Ej: dirección a la que mira, si tiene seleccionada un arma counter del enemigo, si es vulnerable momentáneamente, si está siendo atacado por otros enemigo	
Sonido	El enemigo mide el nivel de ruido que hace el jugador y si supera cierto nivel lo detecta	Sensor de círculo por distancia. Hay que generar estímulos que se puedan recibir en algún momento	

Tabla 3.3: Tabla Sensores/Emisores

daño colisione contra otro volumen de colisión que recibe daño.
Existen diferentes tipos de daño y parámetros específicos que determinan cómo los volúmenes de colisión reciben, emiten y procesan daño. La primera consideración que hay que tener en cuenta es que el daño no es bidireccional, lo que implica que un volumen que recibe daño no tiene porqué emitir daño. Para reflejar esta diferenciación, en este trabajo se estructuran tres tipos de cajas de colisión:

- Caja de Colisión Recibir Daño (C.C.R.D): Áreas en las que el enemigo es vulnerable.
- Caja de Colisión Emitir Daño (C.C.E.D): Áreas desde las que el enemigo inflige daño.
- Caja de Colisión No Daño (C.C.N.D): Zonas invulnerables del enemigo que no interactúan con el sistema de daño.

Además, estas cajas pueden activarse/ desactivarse y desplazarse dinámicamente en función del estado del enemigo o de las mecánicas del juego. Esto permite modelar comportamientos más complejos, como:

- Añadir funcionalidad a enemigos simples
- Alterar zonas de ataque en función de animaciones o patrones.
- Optimizar las interacciones desactivando cajas no relevantes en momentos específicos del juego.

Además, se puede distinguir entre tres tipos de daños:

3.2.6. Eventos

Un evento es una funcionalidad concreta y puntual que se activa mediante un sensor, está diseñado para ser procesado por cualquier entidad del juego que lo requiera. Cuando se cumplen ciertas condiciones específicas, un evento puede desencadenar otro evento como consecuencia, creando una cadena de interacciones. El uso de eventos contribuye a un diseño más orgánico y dinámico, ya que permite que los cambios de estado en los enemigos sean más evidentes para el jugador, mejorando así la experiencia de juego. Siendo especialmente visible en efectos gráficos y sonoros.

La muerte de un enemigo se implementa como un evento. Este evento se genera cuando el enemigo agota su salud o se encuentra en una condición específica de eliminación. Al activarse, se lanza para que otras entidades del juego lo procesen, representando el fin de la ejecución de la FSM y deteniendo cualquier acción del enemigo.

Nombre	Descripción	Atributos	Ejemplo
Instantáneo	Se aplica de manera inmediata al jugador en el momento en que colisiona con C.C.E.D El daño se aplica de manera instantánea a la entidad la cual tenga el volumen de colisión que lo recibe .	cantidadDaño (si el valor es <0, instakill).	
Persistente	Este tipo de daño se mantiene mientras el volumen de colisión que lo recibe permanezca en contacto con el volumen de colisión que lo ocasiona.	ralentización movimiento, inmediata, daño residual	
Residual	Representa el daño aplicado tras un impacto inicial, pero que permanece activo durante un corto período, infligiendo unas pequeñas cantidades de daño, incluso si el volumen de colisión ya no está en contacto con el volumen que inició el daño.	cantidadDaño numerointervalos tiempoIntervalos (cada cuanto hace daño)	Un enemigo con un ataque envenenante puede infligir daño progresivo durante unos segundos después de golpear al jugador.

Tabla 3.4: Tabla Daños

Procesamiento de eventos

Todos los eventos generados por sensores se manejan a través de una cola de eventos. Esta cola asegura que los eventos se procesen en orden de llegada (First In First Out, FIFO), evitando conflictos entre ellos. Cada enemigo evalúa los eventos de su cola de manera secuencial, ejecutando las acciones correspondientes antes de pasar al siguiente evento.

Capítulo 4

Conclusiones y Trabajo Futuro

Conclusiones del trabajo y líneas de trabajo futuro.

Antes de la entrega de actas de cada convocatoria, en el plazo que se indica en el calendario de los trabajos de fin de grado, el estudiante entregará en el Campus Virtual la versión final de la memoria en PDF.

Introduction

Introduction to the subject area. This chapter contains the translation of Chapter 1.

Conclusions and Future Work

Conclusions and future lines of work. This chapter contains the translation of Chapter 4.

Contribuciones Personales

En caso de trabajos no unipersonales, cada participante indicará en la memoria su contribución al proyecto con una extensión de al menos dos páginas por cada uno de los participantes.

En caso de trabajo unipersonal, elimina esta página en el fichero `TFGTeXiS.tex` (comenta o borra la línea `\include{Capitulos/ContribucionesPersonales}`).

Estudiante 1

Al menos dos páginas con las contribuciones del estudiante 1.

Estudiante 2

Al menos dos páginas con las contribuciones del estudiante 2. En caso de que haya más estudiantes, copia y pega una de estas secciones.

Bibliografía

Y así, del mucho leer y del poco dormir, se le secó el celebro de manera que vino a perder el juicio.
(modificar en Cascaras\bibliografia.tex)

Miguel de Cervantes Saavedra

Bautista, T., Oetiker, T., Partl, H., Hyna, I. y Schlegl, E. *Una Descripción de LaTeX 2ε*. Versión electrónica, 1998.

Knuth, D. E. *The TeX book*. Addison-Wesley Professional., 1986.

Krishnan, E., editor. *LaTeX Tutorials. A primer*. Indian TeX Users Group, 2003.

Lamport, L. *LaTeX: A Document Preparation System, 2nd Edition*. Addison-Wesley Professional, 1994.

Mittelbach, F., Goossens, M., Braams, J., Carlisle, D. y Rowley, C. *The LaTeX Companion*. Addison-Wesley Professional, segunda edición, 2004.

Oetiker, T., Partl, H., Hyna, I. y Schlegl, E. *The Not So Short Introduction to LaTeX 2ε*. Versión electrónica, 1996.

Patashnik, O. Build a bad guy workshop. 2014. Disponible en `https://www.gamedeveloper.com/design/build-a-bad-guy-workshop---designing-enemies-for-retro-games` (último acceso, january, 2025).

Apéndice A

Título del Apéndice A

Los apéndices son secciones al final del documento en las que se agrega texto con el objetivo de ampliar los contenidos del documento principal.

Apéndice B

Título del Apéndice B

Se pueden añadir los apéndices que se consideren oportunos.

Este texto se puede encontrar en el fichero Cascaras/fin.tex. Si deseas eliminarlo, basta con comentar la línea correspondiente al final del fichero TFGTeXiS.tex.

–¿Qué te parece desto, Sancho? – Dijo Don Quijote –
Bien podrán los encantadores quitarme la ventura,
pero el esfuerzo y el ánimo, será imposible.

Segunda parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes

–Buena está – dijo Sancho –; fírmela vuestra merced.
–No es menester firmarla – dijo Don Quijote–,
sino solamente poner mi rúbrica.

Primera parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes