# Alicientes de los juegos basados en máquina de estados.

## 1. Introducción

### 1.1. Introducción

En este proyecto se demuestra la efectividad de máquina de estados orientadas a videojuegos, mostrando la amplia funcionalidad de esta técnica y sus mejoras en cuanto a refactorización de código. El juego tiene un modelo de comportamiento en el que pasa por diferentes estados en los que se determinara funcionalidades propias de cada uno.

### 1.2. En que consiste el juego

Es un videojuego 2D de naves tipo arcade, en el que se proporciona al jugador la opción de elegir sus habilidades principales, contando inicialmente con solo cinco puntos de habilidad, tras elegir el jugador se encuentra en un mundo abierto en el que puede viajar con su nave espacial a diferentes planetas, en dichos planetas habrá un enemigo al cual habrá que derrotar, una vez derrotado se recompensa al jugador con un punto de habilidad que podrá gastar en la habilidad que desee, y seguir así invadiendo planetas, hasta acabar con todos los enemigos, aparte de la recompensa de un punto de habilidad por haber derrotado a un enemigo cada 5 enemigos derrotados la nave del jugador se actualizara, haciendo notar al jugador un avance en el juego y una recompensa por sus habilidades.

## 2. Estado del arte

### 2.1. HTML5

Es la versión 5 de HTML (HyperText Markup Language). Es un lenguaje basado en etiquetas, que permite la elaboración de páginas web, definiendo su contenido (imágenes, texto, videos...) en la versión 5 se incluyen nuevas mejoras.

#### 2.1.1. Canvas

Es un elemento de HTML5 que permite generar gráficos estáticos o dinámicos, accediendo a él mediante código en JavaScript.

### 2.2. CSS3

Es la versión 3 de CSS (Cascading Style Sheets) es un lenguaje que sirve para crear el diseño de un código HTML, sosteniendo la idea de separar el diseño de una página web de su estructura.

### 2.3. JavaScript

Es un lenguaje de programación interpretado (quiere decir que no necesita compilación) Es un lenguaje orientado a objetos, que se basa en prototipos, es débilmente tipado y dinámico. Normalmente se usa para programar del lado cliente, pero también puede usarse para programar del lado servidor, gracias a Node.js. Comúnmente se usa para añadir funcionalidad a las páginas web, dicho código se escribe en un archivo independiente y se enlaza con la página web definiendo en el código HTML las etiquetas <script></script> y entre medias la ruta del fichero.

### 2.4. lluviaProject

Lluvia es un DSL (Domain Specific Language) que se refiere a una especificación de un lenguaje de programación, en este caso JavaScript, proveyendo al lenguaje de más funcionalidades de las que posee el lenguaje nativo. Ha sido utilizado para desarrollar la mayor parte del software de este proyecto.

#### 2.4.1. Device

Los Device son objetos independientes capaces de comunicarse entre ellos mediante eventos.

#### 2.4.2. States

Los States son automatas que proporcionan determinadas funcionalidades en cada instante.

#### 2.4.3. Gates

Son puertas de entrada/salida por las que pasan los eventos con los cuales se comunican los Device.

#### 2.4.4. Boids

Son una inteligencia artificial, en la que cada boid es programado con un comportamiento independiente del resto de boids, creando de esta forma una inteligencia artificial de un conjunto de comportamientos definidos.

## 3. Arquitectura de la aplicación

### 3.1. Clases

Una clase es un modelo o plantilla diseñado para crear objetos, las clases poseen atributos y métodos y los objetos los heredan. Cuando un objeto se crea a partir de una clase se llama instanciación de la clase.

#### 3.1.1. Point Dealer

Es un Device cuya funcionalidad es repartir puntos de habilidad al jugador, con un máximo determinado. Las habilidades en las que puede repartir los puntos son “damage”, “resistance” y “speed”. Esta clase tiene una interface con la que se comunica en todo momento el jugador. Esta provista de 2 botones (button_plus  y button_minus) con los que el jugador puede aumentar o diminuir los puntos de cada habilidad, estos se comunica al Device con el uso de Gates. Posee otro botón (button_play) el cual tiene la funcionalidad de enviar al jugador al siguiente Device, únicamente si el jugador ha gastado todos los puntos de habilidad.

Point Dealer posee cuatro métodos, que junto a los Gates dotan al device de funcionalidad.

get_back_from: Permite disminuir los puntos de habilidad, de la habilidad en la cual se esté pulsando el “button_minus”.

transfer_to: Realiza la tarea opuesta a get_back_from, pues si este quita puntos, transfer_to los suma hasta llegar al máximo de puntos permitido.

render: Cambia las imágenes de la barra de puntos de cada habilidad dependiendo del botón que pulse el jugador. Dando a conocer al usuario los puntos que esta agregando o quitando en cada momentos y en cada habilidad.

attend_show_skills: Permite que Point Dealer se muestre al iniciar el juego.

#### 3.1.2. Space

Es un Device en el que se crean veinte Gates siendo cada uno de ellos un planeta diferente al que el jugador podrá viajar, siendo elección libre para el jugador cual escoger.

Sus métodos son:

attend_show_space: Permite que Space se muestre cuando el jugador a pulsado el botónbutton_play  de Point Dealer.

up: Proporciona la funcionalidad del botón situado encima del canvas para mover el mapa de los planetas hacia arriba.

right: Proporciona la funcionalidad del botón situado a la derecha del canvas para mover el mapa de los planetas hacia la derecha.

down: Proporciona la funcionalidad del botón situado debajo del canvas para mover el mapa de los planetas hacia abajo.

left: Proporciona la funcionalidad del botón situado a la izquierda del canvas para mover el mapa de los planetas hacia la izquierda.

### 3.1.3. Planet

Es el Device en el que se desarrolla la pelea contra el jugador, aquí se crean dos boids (Player y Enemy) y los estados por los que podrá pasar la pelea.

Los métodos son:

Initialize: Inicializa a Player y a Enemy para que se creen en Planet.

attend_show_planet: Hace que el planeta se muestre, con una alerta que indica el número del planeta que es.

has_born: Permite que el boid se cree.

Draw: Permite dibujar el boid.

new_boid: Crea una nuevo boid.

new_boid_of: Crea un boid a partir de una clase, previamente definida.

get_boids:

repaint(ctx): Pinta el canvas de negro.

resize: Permite que el jugador pueda poner el juego en apantalla completa usando el botón "button_screen"

Los estados son:

running_up: Este estado se ejecuta a la vez que el Device. Llama al método draw para pintar a los boids sobre el canvas.

running_fight_playing_up: Despues de que se ejecute running_up y pinte en el canvas se jecuta este estado, que será cuando los boids luche entre si.

running_fight_win: Si la pelea entre los boids la gana Player se ejecuta este estado, que muetsra una alerta diciéndole al jugador que ha ganado si quiere continuar y cambia de estar en el Device Planet a estar en Point Dealer donde podrá sumarse otro punto de habilidad.

running_fight_lose: Si Player pierde la pelea se le mostrara una alerta en la que se pregunta al jugador si quieres salir o empezar de nuevo, en caso de que juegue de nuevo se volverá a Point Dealer y tendrá que repartirse los puntos iniciales otra vez.

### 3.1.4. Player

Es un Boid totalmente controlado por el jugador mediante el teclado. Los atributos de player damage, resistance y speed son definidos por el jugador en el pointDealer, en cambio la posición, la vida y la imagen están predeterminados y el jugador no podrá cambairlos.

Los métodos son:

random(max): Escoge un numero aleatorio para empezar a pinta esrellas en el canvas de Planet.

Move: Hace que se mueva Player, los disparos que lanza y las estrellas del canvas.

paint(ctx): Pinta a Player, a los disparos, a las estrellas y los FPS.

Draw: Ejecuta el método repaint(ctx), declarado en Planet, después el método move y después el método paint.

3.1.5. Enemy

Es un array de Boids, que tienen sus propios comportamientos definidos así como moverse y atacar al jugador, tienen unos atributos compartidos como la posición, y otros atributos individuales como: life, damage, resistance, speed y la imagen de la nave.

Los métodos son:

Move_enemy

draw

3.2. Diagrama de clases

3.3. Diagrama de estados

## 4. Desarrollo de la aplicación

La aplicación ha sido desarrollada con varias tecnologías anteriormente mencionadas y con el software descrito a continuación.

### 4.1. Entorno utilizado

Un entorno de desarrollo puede ser una o varia aplicaciones, que se compone de un editor de texto, un depurador y un compilador, en este caso como el lenguaje utilizado ha sido JavaScript no ha sido necesaria la utilización de un compilador.

#### 4.1.1. SublimeText 3

Es un editor de texto o editor de código fuente y ha sido utilizado para crear todo el código fuente de la aplicación.

#### 4.1.2. Mozilla Firefox

Es un navegador web libre y de código abierto. Su consola JavaScript ha sido utilizada para comprobar los errores en el código fuente y su herramienta de depuración incorporada ha sido de gran ayuda a la hora de realizar las tareas de testing y debug.

#### 4.1.3. PhotoShop CS6

Es un editor de gráficos, con el cual se ha desarrollado gran parte de los diseños del videojuego.

#### 4.1.4 Git Hub

Es un control de versiones on-line, basado en Git, desde el cual se descarga lluviaProject y ha sido muy útil para ordenar el código y poder portarlo de un ordenador a otro.

## 5. Diseño

### 5.1. Diseño de interface

La interface está diseñada con HTML y CSS siendo esta una interfaz sencilla e intuitiva para el jugador.

La primera pantalla se compone de 3 y barras y 3 botones, de los cuales la funcionalidad está hecha con JavaScript.

La siguiente pantalla se denominaría como el entorno abierto y de libre elección del videojuego, que consiste en una simulación del espacio exterior con diversos planetas a los que el jugador puede viajar.

### 5.2. Diseño de mundo

Está enfocado al espacio, en el que hay diversos planetas, cada uno diferente, en el que al entrar en cada planeta hay un enemigo y el jugador deberá enfrentarse a él.

### 5.3. Diseño de Enemy y Player

Los diseños de los enemigos son dibujos obtenidos a través de internet, el diseño de los sprites de Player están realizados por mí, dibujado a mano, escaneado y coloreado y sombreado con PhotoShop CS6.

### 5.4. Diseño de nivel

El diseño de nivel es una característica fundamental a la hora de desarrollar un videojuego, ya que si el diseño no es bueno el juego se hará aburrido y tedioso, para ello un buen diseño de nivel ha de tener unas características, que son objetivo, recompensa y dificultad, las cuales se describen en los siguientes apartados.

#### 5.4.1. Objetivo

El objetivo en el juego es acabar con todos los enemigos cada uno de ellos esta resguardado en un planeta y el jugador tendrá que derrotarlo para así liberar al planeta del invasor.

#### 5.4.2. Recompensa

La recompensa en un videojuego siempre está clara, la victoria, pero eso no es suficiente para hacer un juego divertido, tiene que tener algo más, en este caso, cada vez, que el jugador lucha contra un enemigo y sale victorioso, el jugador es recompensado recibiendo un punto de habilidad, para seguir mejorando su nave y poder enfrentarse así, con enemigos más poderosos.

#### 5.4.3. Dificultad

El orden de dificultad es el orden numérico de los planetas, siendo el planeta 1 el que posee al enemigo más débil y el planeta 20 el que posee el enemigo más poderoso.

Al ser un mundo abierto y el jugador no tener consciencia sobre el orden de los planetas, hace que la dificultad del juego aumente y posiblemente también la curiosidad de saber que pasara en el siguiente planeta al que viaje, eso refuerza enormemente las ganas de seguir jugando.