|  |
| --- |
| [Escriba el nombre de la compañía] |
| Trabajo de Fin de Grado |
| Aplicación de los videojuegos a la medida de la inteligencia |
|  |
| **Alejandro Ortega Álvarez** |
| **Curso 2019/2020** |

|  |
| --- |
| TFG del grado de Desarrollo de Videojuegos de la Universidad Complutense de Madrid |



# Indice

Objetivo

2.Planificacion

2.1 El juego:Edge

3.Desarrollo

# Objetivo

Una gran parte de los videojuegos diseñados para plataformas móviles (sobre todo) están destinados a resolver puzzles o niveles donde se requiere habilidad y precisión. Estos juegos pueden ser muy distintos entre sí mismo pero todos tienen una característica propia común: necesitan un avance progresivo notable en la dificultad de los desafíos.

Este aspecto hace que el jugador tenga que ir memorizando ciertos patrones propios del juego, con los que poder superar los nuevos retos que vayan apareciendo. Se podría decir, pues, que el jugador tendrá que desarrollar cierta inteligencia si quiere afrontar desafíos más complejos.

El objetivo de este proyecto es crear un videojuego desde 0 en Unity ya existente para alguna plataforma (para no tener que preocuparse del diseño de los niveles) que tenga este aspecto ya mencionado, con el fin de realizar un pequeño experimento en el que participen jugadores reales para ir observando y midiendo el desarrollo de la inteligencia mientras juegan.

# Planificación

Somos 3 integrantes en este proyecto de fin de grado. Lo primero de todo era darnos a conocer el objetivo y las distintas partes de proyecto que íbamos a llevar a cabo.

Después, tocaba elaborar una pequeña planificación para que pudiéramos repartir el trabajo lo más equitativamente posible entre los integrantes del grupo y entre el tiempo del que disponíamos. La primera parte fue fácil, ya que cada integrante debe desarrollar un juego por separado, por lo que esa tarea repartía ya bastante equitativamente trabajo para un considerable periodo de tiempo.

Para la segunda parte también fue fácil, de hecho no hizo falta planificarla demasiado. Al disponer de 2 cuatrimestres para realizar el TFG, nuestra idea fue la siguiente:

* El primer mes lo empleamos en planificación, investigación de fuentes que nos pudieran resultar útiles en el desarrollo; y en la elección de cada uno de los juegos a desarrollar. Para facilitarnos el desarrollo, los juegos a desarrollar serían juegos que ya existen, y que podrían ser de distintas plataformas (ordenador, Tablets, móviles, Wii); con el objetivo de ahorrarnos tiempo en diseño de niveles, al tener que reconstruir tal cual los de los juegos ya existentes y no tener que inventárnoslos.
* El resto del primer cuatrimestre nos centraríamos principalmente en el desarrollo del juego elegido para Unity, con tutorías cada 2-3 semanas con los profesores para ir informando del avance del juego y poder ir resolviendo los posibles inconvenientes que surgieran con la funcionalidad del juego o con programación. Esta tarea, al ser tan extensa, la teníamos que subdividir en varias sub-tareas que serían las siguientes:

1. Lo primero de todo plantear cómo iba a organizarze el proyecto de Unity de forma cómoda para que fuera fácil, dependiendo del juego, añadir niveles, funcionalidades nuevas o elementos.

Con ello podíamos determinar, por ejemplo, cómo iban a ser las escenas, cuántas serían necesarias y por qué, cuántos managers serían imprescindibles, cuáles de ellos globalmente, cómo se iban a comunicar las escenas entre sí, cómo se iban a guardar los posibles datos, etc.

Aunque posiblemente cambiara gran parte en el desarrollo del juego, pero es bueno ir teniendo una idea en la que centrarse.

1. Una vez fijada la idea, comenzar el desarrollo en base a esa idea. Aunque los juegos no eran especialmente complicados, siempre iban surgiendo obstáculos que requerían un cambio en alguna de las ideas o incluso, en ocasiones, tener que prescindir de funcionalidades o elementos del juego si su coste era elevado y vencía en peso a su importancia en el juego.
2. Una vez desarrollada una parte considerable del juego, tendríamos que programar la parte de guardado de datos que nos indicara la psicóloga que nos ayuda con el proyecto. Estos datos son las variables que nos interesan registrar en el juego para la medición del objetivo del proyecto sobre el juego.
3. Al tener el juego “acabado” (los niveles que se nos indique que sean necesarios), debemos realizar una pequeña fase de testeo para resolver todos los posibles bugs que pudieran afectar al jugar e introducir, si da tiempo, algunos detalles visuales o arreglar elementos estéticos que no hubiera dado tiempo en el desarrollo.

* Antes de comenzar el experimento, lo del server y “testeo” para ver que funciona bien el juego y tal.
* Experimento

# Elección de los juegos

Nuestro profesor nos proporcionó información con una lista de juegos que podíamos desarrollar para este proyecto.

Tal como nos indicó la psicóloga, hay tres aspectos a tratar en la medida de la inteligencia:

- Gf: fluid reasoning o razonamiento fluido, es la capacidad de pensar con flexibilidad (fluidez) a la hora de resolver problemas.

- Gv: visuospatial ability o habilidad visoespacial es la capacidad de representar, analizar y manipular objetos en la mente.

- Gs: processing speed o velocidad de procesamiento, referido a la capacidad de poder realizar acciones con mayor velocidad cuando son aprendidas por repetición.

En cada uno de los juegos, hay al menos una de estas características posibles que corresponde con la que tendríamos que medir posteriormente en el análisis de resultados.

Cada integrante debería elegir 1 distinto con total libertad, siempre cuando se acondicionara a :

* Cada integrante debería desarrollar un juego distinto.
* Cada integrante debía escoger un juego que tuviera un aspecto a medir que no tuviera otro compañero, para así poder medir todos los aspectos.
* Cada integrante debía considerar las posibles complicaciones que pudiera tener el juego para el desarrollo en Unity, ya que el construir el juego desde 0, al ser lo que más tiempo nos lleve y no la parte esencial del proyecto, debería tener el mínimo de complicaciones y poder hacerse sin problema alguno. Con este punto tuvimos que descartar más de un juego de la lista.

**Los juegos**

La lista de juegos que se nos proporcionó fue la siguiente:

* **Blek (Gf):** Juego 2D para móviles donde el jugador ha de realizar trazas con el dedo para poder resolver puzles.

[http://blekgame.com](http://blekgame.com/)

* **Rail maze (Gf):** Juego 2D para móviles donde el jugador tiene que completar un escenario de vías de tren y otros elementospara que el tren pueda llegar al destino.

<https://apps.apple.com/es/app/rail-maze-train-puzzler/id445853367>

* **Ski jumping pro (Gs):** Juego 3D para móviles que es un simulador de saltos de esquí.

<https://apps.apple.com/us/app/ski-jumping-pro/id585599497>

* **Unpossible (Gs):** Juego 3D para móviles y tablets donde el jugador se desplaza por un circuito cilíndrico tratando de esquivar obstáculos girando la cámara mientras va aumentando la velocidad.

<https://apps.apple.com/us/app/unpossible/id583577503>

* **Art of Balance (Gf/Gv):** Juego 3D para WiiU que consiste en tratar de encontrar el equilibrio apilando varios objetos sobre una plataforma.

<https://www.nintenderos.com/2014/10/analisis-art-of-balance-eshop-wii-u/>

* **Crazy pool (GV):**    Juego 2D de física simulador de billar.

<https://apps.apple.com/us/app/crazy-pool-3d-free/id391267734>

* **Edge (Gv):** Juego 3D para WiiU y móviles donde el jugador ha de atravesar un escenario manejando un cubo.

<https://en.wikipedia.org/wiki/Edge_(video_game)>

Nuestras elecciones fueron las siguientes:

-Ming: Unpossible, versión para móvil (Gs)

-Pablo: Blek, versión para móvil (Gf)

-Yo: Edge, versión para ordenador (Gv)

# El juego: Edge

Edge es un juego originario de Wii U que también se ha desarrollado para móviles y Tablets.

El juego consta de varios niveles o desafíos en los que el jugador controla un cubo que puede moverse en 4 direcciones posibles. El mundo es un escenario 3D compuesto por casillas discretas por las que irá desplazándose el cubo. El objetivo del jugador es tratar de llegar a la casilla final del nivel en el menor tiempo posible consiguiendo unos items o prismas que estarán distribuidos por todo mapa. Al final de cada nivel sale un menú de información con las estadísticas conseguidas dependiendo de la cantidad de prismas cogidos con respecto al total que hubiera, y el tiempo. También se tiene en cuenta el número de “muertes” que haya habido desde que comenzó el nivel (el jugador “muere” cuando cae por un borde del escenario y vuelve a aparecer desde un checkpoint o punto de guardado).

Todos los datos se quedarán guardados y podrán ser superados si se vuelve a repetir el nivel. Se irán desbloqueando nuevos niveles a medida que se vayan completando.

Más datos sobre el juego:

-El cubo puede subir escalones, pero sólo de uno en uno y si no hay obstáculos encima.

-El cubo no podrá moverse si tiene un bloque justo encima de él.

-El cubo puede ser desplazado por obstáculos movibles.

-Hay plataformas que pueden ser accionadas con activadores cuando se cae sobre ellos.

-Existen otros activadores que empujan al cubo unas cuantas casillas en una dirección determinada.

-Hay bloques de suelo quebradizo que se caen al poco tiempo de pasar por encima de ellos. Estos bloques sólo volverán a recolocarse si el jugador muere.

-El nivel puede reiniciarse en cualquier momento, perdiendo el progreso que se tuviera.

-Se podrá ver en todo momento el cronómetro con el tiempo desde que comenzó el nivel.

**Problemas en el desarrollo**

-A nivel de programación, lo único complejo fue el movimiento rotatorio del cubo, ya que era muy particular y dependiente de obstáculos el que se pudiera o no mover. El movimiento del cubo es una rotación sobre uno de sus bordes que hace que se desplace exactamente su longitud en una de las direcciones, de este modo tenemos movimientos discretos. Pero en Unity no era tan sencillo, era necesario hacer correcciones de posición constantemente porque al llevar un tiempo jugando, el jugador al final llegaba a “salirse” de las casillas, sobre todo al subir o bajar peldaños.

-El gran problema al que me enfrenté no fue en cuanto a código, sino un problema con las físicas. El juego se podría decir que carece completamente de físicas (toda comprobación de si es posible o no un movimiento se realiza mediante rayos o raycast), excepto a la hora de introducir obstáculos movibles. Estos obstáculos hacían que el cubo constantemente fuera empujado o desplazado en cualquier dirección de forma no discreta, y no hubiera habido problema alguno si el juego no requiriera ser discretizado para poder resolver algunos puzles (hay una gran cantidad de casillas de las mismas dimensiones que el tamaño del cubo, a las que no se podía acceder en cuanto el cubo se saliera lo más mínimo de su rango dentro de las casillas). Y no servía colocar colliders para que las plataformas empujaran al cubo porque si se usan las físicas de Unity se han de usar para todo el juego, y el resto del juego carece de estas físicas como hemos mencionado. Lo que originó el tener que depender de unas físicas distintas a las de Unity, y tener que estar corrigiendo posiciones constantemente al entrar y salir de plataformas movibles, y tener que prescindir de algunas plataformas que originaban algún efecto complicado de solucionar (tener que escalar plataformas mientras se estaban moviendo, por ejemplo).

El juego consta de 6 niveles. Se tomarán los primeros 3 niveles como un tutorial donde los jugadores del experimento se familiarizarán con las mecánicas del juego y sus elementos (en ninguno de estos niveles se medirá la inteligencia). Estos 3 niveles corresponden a los primeros 3 niveles del juego original. Los otros niveles restantes son los que van a formar parte del experimento y, por lo tanto, se medirán los resultados obtenidos en ellos. Estos 3 últimos niveles corresponden a los niveles 23, 24 y 25 del juego original.

# Implementación

**El juego consta de 8 escenas:**

* La primera escena, que será la primera que nos encontremos nada más ejecutar el juego, es el menú principal donde se podrá escoger entre jugar los niveles del tutorial o dar paso a los niveles del experimento. En caso de escoger la segunda opción, aparecerá otra pantalla que estará en la misma escena, donde el usuario deberá de añadir un código para poder comenzar el experimento.

En esta escena se cargarán por primera y única vez el GameManager y el SessionManager, explicados más adelante.

* La segunda escena es un menú de información que aparecerá nada más acabar cualquier nivel, ya sea del tutorial o del experimento. En él aparecerán los resultados que el jugador ha conseguido en ese nivel: el tiempo, el número de veces que ha muerto y el número de objetos cogidos con respecto a todos los que contenía el nivel.
* Las otras 6 escenas corresponden a los 6 niveles de juego. Las escenas de juego se componen de:

-Player: objeto que controlará el jugador.

-Cámara y luces: la cámara seguirá al player en todo momento.

-Terreno: todos los demás objetos que componen el nivel. A su vez, se divide en objetos de suelo, objetos de pared, ítems (que son los objetos que va recogiendo el player) y deathzones (el terreno está dividido en zonas donde si el jugador muere reaparece en un sitio en concreto –checkpoints-, y dependiendo de la complejidad y extensión del nivel habrá más o menos de estos puntos).

-LevelManager: controlador de todos los eventos que ocurren en el nivel para informar al GameManager y al SessionManager.

**En cuanto a objetos o elementos del juego, se dispone de los siguientes:**

* Jugador: cubo en 3D que representa a la entidad manejable con el jugador. Cambia de color constantemente siguiendo un patrón para que llame directamente la atención del usuario. El jugador puede avanzar en 4 direcciones con un movimiento de rotación del cubo con las teclas WASD o las flechas del teclado. El jugador también puede “escalar” o subir obstáculos si tienen exactamente la misma altura que el jugador y no tienen otro objeto encima. Los movimientos del player son todos discretos –el jugador avanza una posición exactamente igual a su anchura hacia la dirección indicada- salvo cuando es arrastrado o empujado por plataformas.
* Paredes y suelo: elementos estáticos en todo momento que no realizan otra acción más allá de notificar al player de que están ahí para que el jugador no choque ni caiga al vacío. No obstante, hay un tipo especial de paredes del que hay que hacer una distinción especial para que bloquee el movimiento del jugador si está siendo arrastrado por una plataforma.
* Plataformas: objetos de dimensiones variadas que pueden ser tanto de suelo como de pared. Pueden arrastrar y empujar al jugador acabando siempre su posición final en una posición discreta.
* Activadores de plataformas: la mayoría de las plataformas se activan y desactivan automáticamente pero otras no lo hacen hasta que el jugador acciona un activador.
* Activadores de velocidad: tienen la misma apariencia que los activadores de plataformas pero en lugar de activar lo que hacen es empujar al player un número de casillas determinado en una dirección.
* Doble activador: parecido al activador de plataformas, pero dispone de 2 objetos que se activan y desactivan a la par (en cuanto uno se activa el otro se queda en posición para que pueda ser activado y al revés), de forma que haya que ir alternando entre los dos para que algunas plataformas avancen por pasos.
* Suelo quebradizo: suelo que se rompe y desaparece al poco tiempo de que pase el player por él.
* Trigger de MiniController: la apariencia es un marco pintando en el suelo que cambia de color y se agranda y se reduce constantemente. Cuando el jugador acaba en esta casilla, se reduce a un cuadrado una tercera parte de su tamaño original, y obtiene habilidades nuevas para poder avanzar en algunos niveles.
* Items: son cubos pequeños que flotan en el suelo y también cambian de color constantemente para que el jugador los vea resaltados. Desaparecen cuando los coge el player y suma 1 al contador de items del nivel.
* Plataforma fin de nivel: es una plataforma azul de 3x3 que representa el final del nivel, el jugador ha de acabar justo en su centro para darlo por concluido.

**Implementación de Scripts:**

Managers:

* PlayerController: gestiona el input del usuario para el control del player y realiza la acción que corresponda si es posible. Le gestión de físicas para saber si el jugador choca con un obstáculo o no se realiza mediante RayCasts en el método CanMove(). No sólo tiene que lanzar Raycasts sobre la dirección en la que se quiere avanzar, también hay que tener en cuenta que:

-Puede haber un obstáculo que impida hacer movimiento normal pero permita ser escalado si es justo de 1 unidad de altura.

-No puede avanzar si el jugador tiene un obstáculo justo encima suya, ya que no tiene “hueco” para realizar la rotación.

-Por consiguiente, no puede avanzar a una casilla vacía que justo encima tenga un obstáculo.

-Al igual que el punto anterior, no puede subir a un escalón libre si justo encima de donde quedaría el player hay un obstáculo.

-Tampoco puede escalar un peldaño si justo encima hay un objeto distinto a un activador o item.

-También se lanzan constantemente Raycast hacia abajo para ver si el player está sobre suelo o sobre vacío en el que tenga que caer.

El jugador dispone de varios estados en los que puede estar. Pero únicamente puede estar en uno en cada instante. Los estados son los siguientes:

-Parado: para realizar cualquier acción, el jugador tiene que estar en este estado. También es el único estado donde se lanzan Raycast para ver si pisa o no suelo para evitar que caiga cuando esté realizando movimientos ya que partiría la discretización de casillas y atravesaría paredes y obstáculos en ocasiones.

-MovW, movA, movS, movD: representan que el jugador está realizando un movimento normal hacia una de las 4 direcciones posibles. Una vez haya terminado de rotar los 90 grados pasará al estado parado.

-SubeW, subeA, subeS, suveD: al igual que los 4 anteriores, pero para cuando está subiendo escalones. El movimiento es distinto ya que se rota el cubo con respecto a la arista superior, no inferior, de la dirección indicada. Además, este movimiento es de 180 grados.

-Cayendo: en este estado se simula que el jugador cae por efecto de la gravedad manualmente, al no haber físicas como tal. El jugador cae a una velocidad constante en Y hasta chocar contra un objeto que no sea Trigger o al llegar a un deathzone porque se cae fuera del nivel. Mientras está cayendo, el jugador no puede realizar ningún movimiento.

-Fin: estado especial que sólo tiene efecto cuando el cubo llega al objeto de 3x3 de fin de nivel, y se simula un efecto de que “flota” en el aire mientras gira.

El PlayerController también es el encargado de ajustar la posición en los 3 ejes del cubo cuando sea necesario, normalmente al acabar un movimiento o terminar de ser empujado por una plataforma, para asegurar que siempre se acaba en una casilla con un valor discreto y no acumular errores de posición.

* PlayerMiniController: controla el movimiento del player cuando se convierte en un cubo pequeño. Tiene prácticamente las mismas funciones y métodos que el PlayerController salvo algunas excepciones:

-El jugador no sólo sube peldaños sino que los puede escalar en el eje Y, es la principal ventaja de este modo para poder llegar a zonas que no haría el PlayerController. Por lo que incluye 4 estados más (escalaW, escalaA, escalaS y escalaD).

-El jugador ahora no puede pasar al estado cayendo nada más pasar al estado parado, ya que eso originaría que cayera constantemente al intentar trepar paredes. Hay que dejar un tiempo de unos 200 milisegundos en los que el jugador esté sin pulsar ninguna tecla para que el jugador comience a caer.

-Los Raycasts del método CanMove() cambian totalmente, ya que se reduce el alcance a un tercio del original, además hay raycasts que ya no son necesarios (no va a ocurrir el caso de que haya un obstáculo justo encima del jugador e impida que se mueva).

-También se incluyen los métodos ReduceCubo() y AlargaCubo() que hacen el efecto de que se hace pequeño el player o vuelve a su estado original después de ser pequeño. Tiene también sus propios estados ya que no interesa que el player pueda realizar movimientos mientras ocurre este efecto.

Lo podemos encontrar en los niveles 5 y 6.

* GameManager: gestor de todas las variables que se necesitan entre escenas o que necesitan ser guardadas, como son el número de niveles, los items recogidos en cada uno de los niveles, las muertes y los tiempos. Todo ello cuando ocurre se lo comunica al SessionManager mediante eventos. También se encarga del cambio de escenas y del controlar si se está jugando un tutorial o no, ya que en modo tutorial no envía eventos al SessionManager, y para controlar los tiempos que se puede estar jugando tutorial o experimento.

Se crea 1 vez en la primera escena de todas y ya no se destruye hasta que se salga de la aplicación.

* LevelManager: controla los sucesos que van ocurriendo en el nivel para comunicarselos al GameManager. También controla el tiempo actual del nivel mientras se está jugando, y cuándo se ha acabado un nivel.
* SessionManager: fiesta

Objetos:

* Activador: activa el obstáculo que se le indique cuando un jugador colisione con su trigger.
* ActivadorVel: desplaza al jugador X casillas cuando colisione con su trigger.
* DobleActivador: requiere de un enlace con otro objeto doble activador. Uno de ellos va a comenzar activado y el otro desactivado. Cada uno puede activar una serie de obstáculos especiales llamados Activables por pasos, explicados más abajo. Cuando el jugador colisione con el trigger del que puede ser activado, éste comprueba que todos sus activables pueden ser movidos (es decir, que ninguno está en movimiento en ese momento) y los activará, pasará a estar desactivado y llamará al otro activador para ponerlo en estado de poder activarse y así repetir el proceso.
* Activable por pasos: Objeto que contiene un array de posiciones que indican los varios recorridos que va a realizar cada vez que el jugador active su DobleActivador que corresponda. En principio, sólo se mueven en X o Z, y no a la vez por cada uno de los pasos, así que se tratan por separado para evitar complicaciones y comprobaciones innecesarias. Se encuentra sólo en el nivel 6.
* Item: Objeto que simplemente gira en el aire y espera a que el player colisione con su trigger para desaparecer y notificárselo al LevelManager.
* Obstaculo: Objeto que se mueve linealmente en una o varias direcciones, y vuelve a su posición original pasado un tiempo que se le indique. Puede activarse con un activador, o estar ya activo desde el comienzo del nivel.
* Pared: para paredes que puedan bloquear el movimiento de un personaje empujado por plataformas. Deben ir a parte de las paredes normales ya que se necesita un collider trigger para que colisione, mientras que las normales disponen de un collider sin trigger para los raycasts.
* Platform: trigger que hace “hijo” al player cuando entra en contacto y le mueve en la dirección en la que se esté moviendo. Al llegar al final del recorrido, si es un trigger lateral, le deshereda para que no le arrastre a la vuelta como si fuera un imán.
* SueloQuebradizo: hace que el objeto que lo contenga caiga medio segundo después de que el player haya colisionado con su trigger.
* TrayectoriaRectangulo: obstáculos especiales que en lugar de moverse en una sola dirección, describen una trayectoria rectangular constante. Se paran al llegar a las esquinas o a la mitad del recorrido para facilitar el movimiento al player. Se pueden encontrar a mitad del nivel 5.
* TriggerMini: trigger que desactiva el PlayerController y activa el PlayerMiniController o al revés. Según el juego original, un mismo triggerMini puede o encoger o agrandar, pero nunca las dos cosas.

Otros:

* CambiarTrayectoria: trigger especial que cambia el movimiento de un obstáculo normal. Sólo lo encontramos en un obstáculo 1x1 a mitad del nivel 6.
* Camera: componente de la cámara que hace que siga el player. Intentamos evitar el que siga al player en el eje Y salvo cuando caiga o escale, ya que constantemente cambia su posición sutilmente en este eje al hacer un movimiento normal hacia cualquier casilla y originaba un efecto de rebote constante en la cámara que podía llegar a ser molesto.
* ChangeColor: componente que se le añade a un objeto para que cambie su color constantemente siguiendo un patrón, y lo haga más vistoso entre los demás objetos. Se usa para el Player, los ítems, los bordes de las plataformas y el triggerMini.
* Deathzone: trigger que comunica al Levelmanager cuándo el player cae sobre él y muere. A parte de restaurarlo al checkpoint que corresponda, también restaura algunos objetos necesarios para poder avanzar (obstáculos, suelos quebradizos y activadores dobles).
* FinEffect: efecto de que flota el player al llegar al fin de nivel, y trigger que comunica al LevelManager.
* MenuInfo: muestra por pantalla los resultados de la partida en el menú de fin de nivel.

**Referencias y enlaces**

Página de información del juego: <https://en.wikipedia.org/wiki/Edge_(video_game)>

Página oficial de Unity: <https://unity.com/es>

Youtube (referencias de vídeos de gameplay): <https://www.youtube.com/>

Enlace de descarga del juego para Android (versión gratuita-demo): <https://play.google.com/store/apps/details?id=net.mobigame.edge.demo&hl=es>

Enlace de descarga del juego para Android (versión de pago): <https://play.google.com/store/apps/details?id=net.mobigame.edge&hl=es>

Enlaces de vídeos de Youtube:

- Trailer (review): <https://www.youtube.com/watch?v=_qmSqiSj61U>

- Niveles 1-10: <https://www.youtube.com/watch?v=Bvyzrr9PeUI&t=195s>

- Niveles 23-25 que corresponden a los 3-5 de este proyecto: <https://www.youtube.com/watch?v=6ZXVhhyHJn8&list=WL&index=5&t=147s>

Enlace de perfil de Github: <https://github.com/alex97ortega>

Enlace del TFG del juego en Github: <https://github.com/alex97ortega/Edge>