

Mack contra los robots del espacio

Julio Armillas Sánchez

Grado Multimedia

Área de videojuegos

Equipo docente:

Profesor colaborador: Joel Servitja Feu

Profesor: Joan Arnedo moreno

10 de Junio de 2018

© (Julio Armillas Sánchez)

Reservados todos los derechos. Está prohibido la reproducción total o parcial de esta obra por cualquier medio o procedimiento, comprendidos la impresión, la reprografía, el microfilme, el tratamiento informático o cualquier otro sistema, así como la distribución de ejemplares mediante alquiler y préstamo, sin la autorización escrita del autor o de los límites que autorice la Ley de Propiedad Intelectual.

FICHA DEL TRABAJO FINAL

Título del trabajo:	<i>Mack contra los robots del espacio</i>
Nombre del autor:	<i>Julio Armillas Sánchez</i>
Nombre del consultor/a:	<i>Joel Servitja Feu</i>
Nombre del PRA:	<i>Joan Arnedo Moreno</i>
Fecha de entrega (mm/aaaa):	06/2018
Titulación:	<i>Grado Multimedia</i>
Área del Trabajo Final:	<i>Videojuegos</i>
Idioma del trabajo:	<i>Castellano</i>
Palabras clave	<i>Videojuego, plataformas, PC</i>
Resumen del Trabajo (máximo 250 palabras): <i>Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.</i>	
<p>En el mercado actual se incrementa cada día la oferta y demanda de videojuegos, en una sed de satisfacer nuevas sensaciones, retos y desarrollar nuevas habilidades por parte del usuario. Esto impulsa a programadores, diseñadores, profesionales de sonido y otras competencias a unir fuerzas o a ampliar conocimientos en otros campos para sumergirse en este “nuevo” mercado.</p> <p>Unity como herramienta gratuita, pero a su vez potente, ha sido la elegida para desarrollar este proyecto por su facilidad de uso y a su vez por su gran potencial incorporando una interfaz amena y programación mediante C#. También se ha interactuando con otras herramientas como Blender, al igual que la anterior gratuita, para el desarrollo 3D y Photoshop utilizado entre otros para editar texturas en el videojuego.</p> <p>El resultado del TFG ha sido un videojuego de plataformas que consta de 7 niveles, los cuales incrementan su dificultad y fuerzan al usuario a desarrollar la agilidad mental a la vez que aporta diversión. Las dificultades en las que se encontrará Mack (nuestro héroe), producirán quizás frustración, pero tras ello la capacidad de pararse y mirar el entorno para que ante dicha situación pueda terminar con éxito el nivel en el tiempo dado.</p>	

Abstract (in English, 250 words or less):

Nowadays the videogames trade is constantly growing, as people want to satisfy their entertainment, face new challenges and improve their capacities. All of this brings developers, designers, sound professionals and others together, with the opportunity to share knowledge or try new methods as they dive in this “new” market.

Unity is not only a free tool but a powerful one. It has been chosen with the goal to develop this project because it is handy and at the same time has a great potential. It has a smooth interface and C# developing.

Other great tools have been used, like Blender which is also free, for 3D design purpose and Photoshop to create videogames textures.

The result of this Final Degree Project is a platform videogame that consists in 7 levels which gradually increase in difficulty. Because of this the user will need to develop his mental agility while at the same time having fun.

The difficulties that Mack (our hero) is going to face might be frustrating for the user, but will enhance the capacity to stop and look at their environment to find the way and successfully finish the level on time.

Índice

1. Introducción	1
1.1 Contexto y justificación del Trabajo.....	1
1.2 Objetivos del Trabajo	1
1.3 Enfoque y método seguido.....	2
1.4 Planificación del Trabajo	2
1.5 Breve resumen de productos obtenidos.....	4
1.6 Breve descripción de los otros capítulos de la memoria	4
2. Diseño y aspecto visual.....	5
2.1 Diseño de los personajes.	5
2.2 Diseño del escenario.....	5
2.3 Jerarquía y progreso del juego.....	7
3. Scripts y comportamientos.	9
3.1 Game, Sus funciones más relevantes y scripts relacionados.	9
3.1.1 Pausa();	9
3.1.2 Disparo();	10
3.1.3 PlayerMuere();	11
3.1.4 Instant Player();.....	12
3.1.5 SetGanarVidas();	12
3.1.6 SetGanarTiempo();	12
3.1.7 SetGanar();	13
3.1.8 SetScore ();.....	14
3.1.9 Morir();	14
3.2 Botones.	14
3.3 GameHistoryMenu.	15
3.4 Level1.....	15
3.5 Player1.	16
3.6 SeguimientoCamara.....	17
3.7 KillEnemy.	17
3.8 Enemigo.	18
3.9 Erase.	19
3.10 MetaFinish.....	19
3.11 TablaHorizontal	20
3.12 TimeController.....	21
3.13 Txt (control de textos).....	22
5. Audio.	22
5.1 Música de fondo durante la partida.	22
5.2 Efectos sonoros.....	23
6. Conclusiones.....	24
7. Glosario	25
8. Bibliografía	26

1. Introducción

1.1 Contexto y justificación del Trabajo

El videojuego para el usuario es un método de ocio, de encontrar una ocupación divertida y a su vez de forma consciente o inconsciente de adquirir nuevos conocimientos o habilidades.

Estos días el Videojuego está ampliando horizontes mediante tecnologías que son relativamente nuevas y se está aplicando a distintos campos como turismo, lenguajes, aprendizaje infantil y adulto en diferentes áreas. Puede ser individual o social, online u offline. Y con infinidad de variantes.

Mack contra los robots del espacio pretende ser un juego con posibilidad de ampliación online en cuanto a las puntuaciones, y para otras plataformas. Pero básicamente offline para su uso aunque por ahora doméstico ahora domestico mediante un PC o Mac permita sin necesidad de internet desarrollar la habilidad evitar peligros del entorno dentro de un juego de plataformas mientras aporta diversión.

1.2 Objetivos del Trabajo

- Desarrollar un videojuego mediante una herramienta con posibilidad de compilado multiplataforma.
- Desarrollar dotes creativas fabricando un entorno que antes no existía.
- Poner en práctica los conocimientos adquiridos para la creación de gráficos y programación.
- Potenciar la habilidad de investigación para desarrollar las técnicas necesarias para la creación del videojuego y la unión entre ellas para su correcta funcionabilidad.
- Conseguir crear al usuario un reto que superar y con ello una sensación, la interactividad juego-usuario así como frustración-recompensa.

1.3 Enfoque y método seguido

Las posibilidades en la creación de un videojuego terminan donde termina la imaginación de su creador, aunque delimitadas por el tiempo y las leyes de que se dispone, por ejemplo, no se puede hacer, así como así un juego de Superman o Supermario, ambos tienen copyright por lo que se debe optar por una historia nueva con personajes nuevos.

Se ha decidido crear un juego basado en robots, así al disparar no se eliminan vidas aunque así se le nombren, cuando un robot muere no muere un ser vivo ni hace alusión a ello de cara no dañar la percepción de los usuarios más jóvenes.

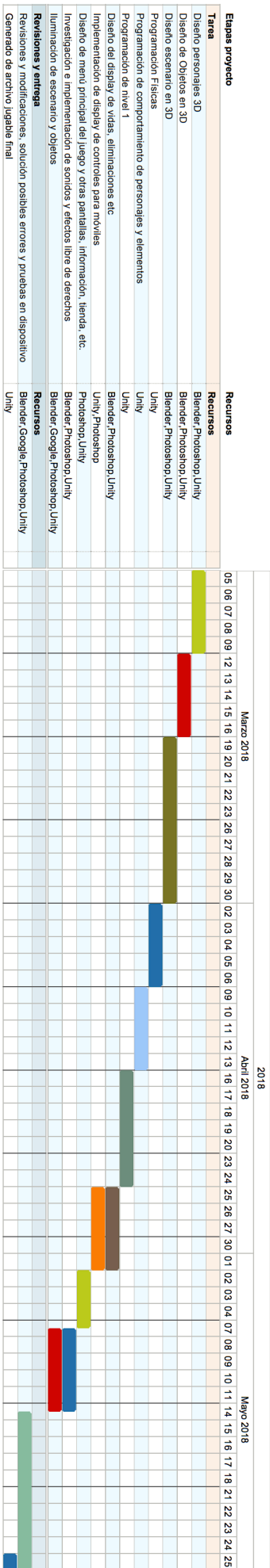
En cuanto a su composición es sumamente difícil sentarse delante del ordenador y ponerse a programar sin referencia alguna, Unity ayuda en este aspecto, teniendo un índice completo y modular, desarrollado con ejemplos que permiten no solo entender sino muchas veces copiar, pegar y adaptar al propósito del entorno que estamos creando.

En este caso se ha servido de los ejemplos de Unity y a veces no encontrando la solución se ha buscado en internet, casi siempre hay alguien que le pasa lo mismo o algo parecido y cuyas respuestas sirven para terminar de encajar las piezas que es montar un videojuego, tras intentarlo todo han surgido ideas del profesor que han ayudado adaptándolas al proyecto en cuestión a terminar de solucionarlo.

1.4 Planificación del Trabajo

La planificación seguida, tras ciertas mejoras sugeridas por el profesor en la primera PEC, ha sido la indicada en siguiente diagrama de Gantt.

Diagrama de Gantt.



1.5 Breve resumen de productos obtenidos

- GDD del videojuego.
- Ejecutable del juego "Mack contra los robots del espacio"
- Todos los componentes creados para el videojuego, Fotografía de fondo, efectos de audio (a excepción de la música de fondo que ha sido descargada), texturas de personajes y escenario, scripts.
- Video presentación y explicación de proyecto.

1.6 Breve descripción de los otros capítulos de la memoria

En el capítulo 2 se mostrará la parte del proyecto orientada al diseño de los personajes y del escenario, así como en la jerarquía del juego y sus pantallas fuera del juego principal.

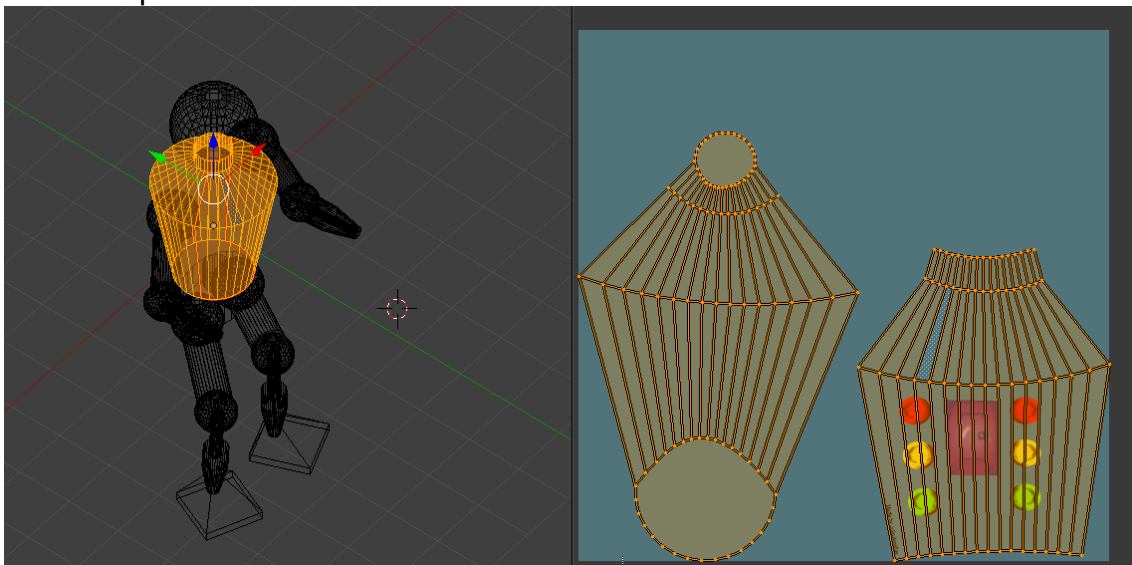
En el capítulo 3 entraremos más detalladamente en el proceso lógico del juego, así como sus comportamientos y la causalidad (acción-reacción) ante la interacción con el usuario.

2. Diseño y aspecto visual

2.1 Diseño de los personajes.

Para el desarrollo 3D se ha utilizado Blender, éste programa además de gratuito y grande, es muy eficaz y aporta gran variedad de atajos de teclado que permiten acceder a sus herramientas rápidamente.

Se ha utilizado para realizar los robots en low poly. Tras tener unas figuras básicas, estas se han seccionado para después preparar sus texturas en Photoshop.

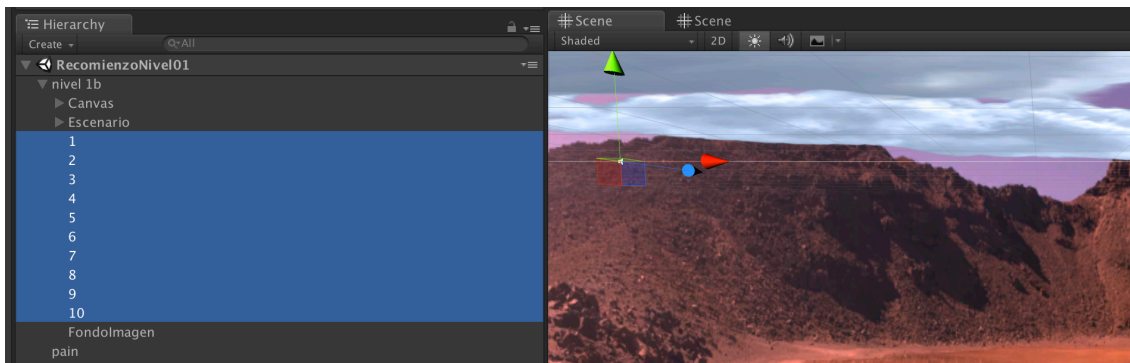


Una vez terminado con Blender, se genera un archivo Fbx, que Unity será capaz de interpretar.

De esta manera el personaje y textura encajaran tal y como se habían diseñado sin necesidad de invertir tiempo modificando las coordenadas y el tamaño de la textura.

2.2 Diseño del escenario.

Se ha optado por utilizar una misma escena para todos los niveles del modo historia, de esta manera reducimos a un mismo espacio de trabajo y evitamos errores futuros como modificar algo en una escena, pero no en otra.



La escena contiene 10 gameObjects invisibles distribuidos a la misma distancia el uno del otro y a la misma altura para que, al comenzar el juego, cargar en cada de una de esas posiciones los distintos prefabs de terreno previamente ordenados en función del nivel actual.

Vemos en como mediante la función “ActualizarLevel()” se crean los gameObject a los que se les asigna la posición en el escenario.

Después mediante un switch case según el nivel actual asigno los prefabs (que se habían cargado con anterioridad) en un orden u otro a un nuevo gameObject y una vez ordenados se les asigna su posición

```
public void ActualizarLevel(){ //Al iniciar nivel el script del boton nos dice el nivel de la pantalla
    currentLevel= PlayerPrefs.GetInt ("CURRENTLEVEL");

    GameObject b1 = GameObject.FindGameObjectWithTag("b1");
    GameObject b2 = GameObject.FindGameObjectWithTag("b2");
    GameObject b3 = GameObject.FindGameObjectWithTag("b3");
    GameObject b4 = GameObject.FindGameObjectWithTag("b4");
    GameObject b5 = GameObject.FindGameObjectWithTag("b5");
    GameObject b6 = GameObject.FindGameObjectWithTag("b6");
    GameObject b7 = GameObject.FindGameObjectWithTag("b7");
    GameObject b8 = GameObject.FindGameObjectWithTag("b8");
    GameObject b9 = GameObject.FindGameObjectWithTag("b9");
    GameObject b10 = GameObject.FindGameObjectWithTag("b10");

    switch (currentLevel) {
    case 1:

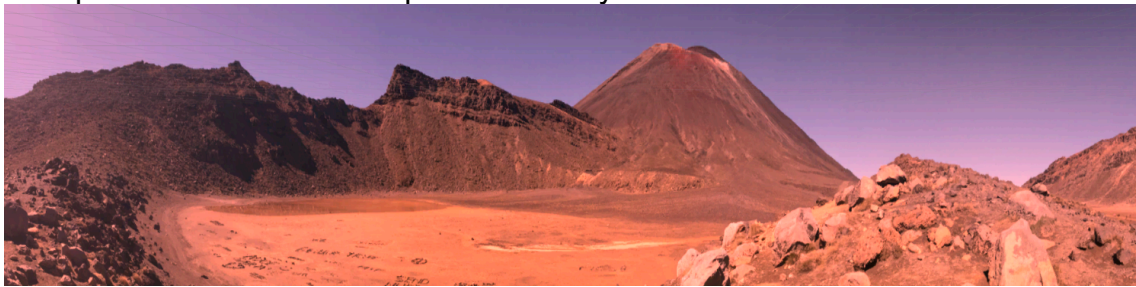
        d1=c1;
        d2=c3;
        d3=c3;
        d4=c5;
        d5=c3;
        d6=c5;
        d7=c3;
        d8=c5;
        d9=c3;
        d10=c2;

        break;
    case 2:

        d1=c1;
        d2=c3;
        d3=c6;
        d4=c3;
        d5=c3;
        d6=c6;
        d7=c3;
        d8=c6;
        d9=c3;
        d10=c2;
        break;
    case 3:

        d1 = Instantiate (d1, b1.transform.position, b1.transform.rotation) as GameObject;
        d2 = Instantiate (d2, b2.transform.position, b2.transform.rotation) as GameObject;
        d3 = Instantiate (d3, b3.transform.position, b3.transform.rotation) as GameObject;
        d4 = Instantiate (d4, b4.transform.position, b4.transform.rotation) as GameObject;
        d5 = Instantiate (d5, b5.transform.position, b5.transform.rotation) as GameObject;
        d6 = Instantiate (d6, b6.transform.position, b6.transform.rotation) as GameObject;
        d7 = Instantiate (d7, b7.transform.position, b7.transform.rotation) as GameObject;
        d8 = Instantiate (d8, b8.transform.position, b8.transform.rotation) as GameObject;
        d9 = Instantiate (d9, b9.transform.position, b9.transform.rotation) as GameObject;
        d10 = Instantiate (d10, b10.transform.position, b10.transform.rotation) as GameObject;
        d1.SetActive(true);
        d2.SetActive(true);
        d3.SetActive(true);
        d4.SetActive(true);
        d5.SetActive(true);
        d6.SetActive(true);
        d7.SetActive(true);
        d8.SetActive(true);
        d9.SetActive(true);
        d10.SetActive(true);
    }
}
```

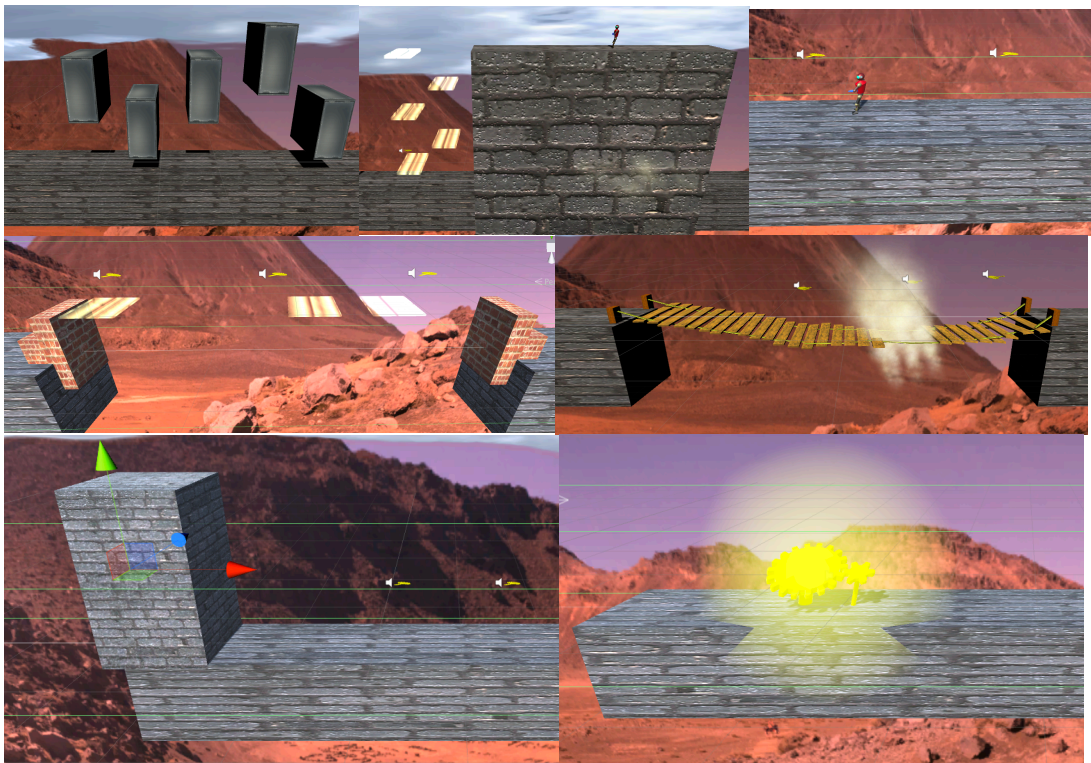
Los escenarios tienen en común la textura del suelo y la imagen de fondo La montaña del Tongariro (Nueva Zelanda) que una vez modificada la intensidad y tono parece un fondo de un planeta árido y seco.



Los prefabs de terreno 7 en total incrementan la dificultad del juego a cada nivel avanzado, 2 de ellos son fijos uno para el inicio y otro para el final.

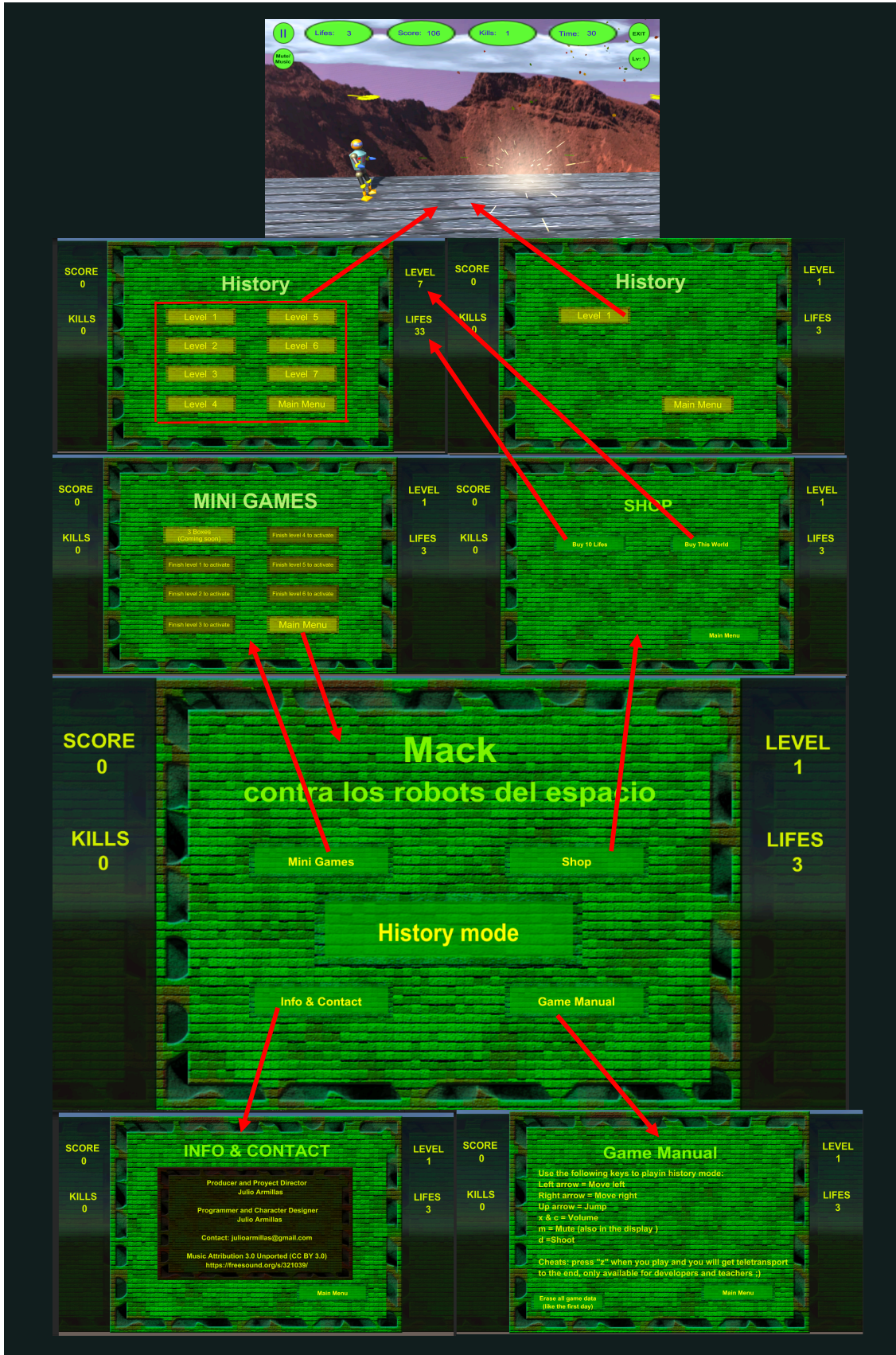
Habrà elementos como pequeños engranajes repartidos por el escenario, estos otorgarán puntos y tiempo al personaje una vez alcanzado un máximo de tiempo se reestablecerá el contador de tiempo, pero a cambio se ganará una vida.

Más dificultades como tablas en movimiento, puentes rotos, cajas de metal que aplastan al que este debajo y enemigos que disparan (a mayor velocidad según el nivel actual) al detectar a nuestro héroe, sacarán a la luz las habilidades del usuario.



2.3 Jerarquía y progreso del juego.

Veamos la distribución de escenas:



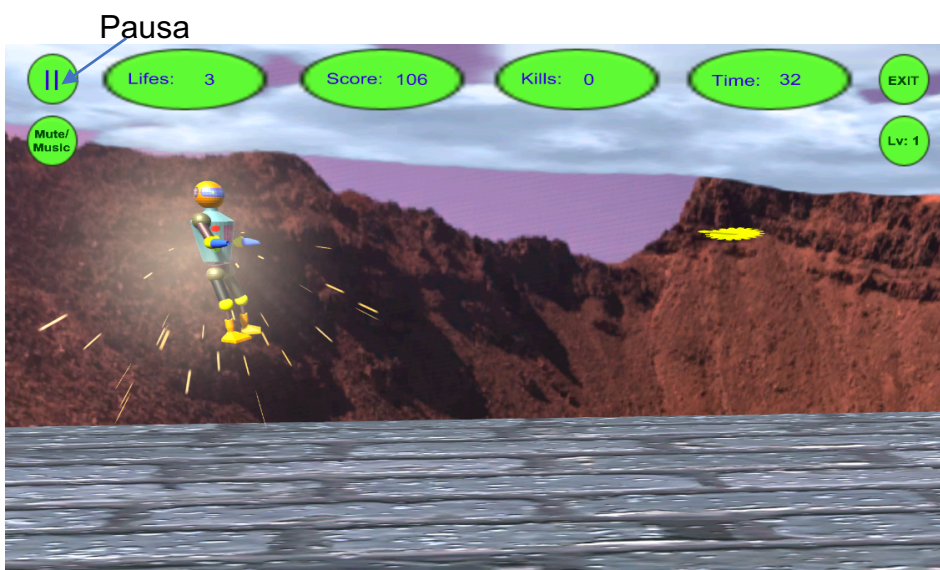
Cada nivel superado habilitará la posibilidad de acceder al siguiente. Los puntos totales son la suma de los obtenidos en cada nivel.

Si el usuario decide acceder de nuevo un nivel ya superado para mejorar la puntuación, solo se sumará si es mejor que la obtenida para ese en concreto.

3. Scripts y comportamientos.

3.1 Game, Sus funciones más relevantes y scripts relacionados.

3.1.1 Pausa();



El botón de pausa aparece en la pantalla del juego una vez iniciado el modo historia, pulsar este botón o bien la letra “P” ejecutará un código en el script del botón o en el evento al pulsar la tecla P de Game que accionan un cambio en la variable “run” en el script de “Game”, esto detendrá el contador, y parará cualquier movimiento.

Script Pausa

```
public void ClickButton(){
    if (game.run) {

        game.run = false;

        Time.timeScale = 0;

    } else if (!game.run) {
        game.run = true;
        Time.timeScale = 1;
    }
}
```

Script Game

```
if (Input.GetKeyDown (KeyCode.P))
    if (!estamuerto) {
        Pausa ();
    }
}
```

El evento Pausa() en el script “game” tiene el mismo contenido que el evento “ClickButton()”, se ha duplicado pero perfectamente se podría simplificar código llamado a la función pausa escribiendo “game.Pausa()”.

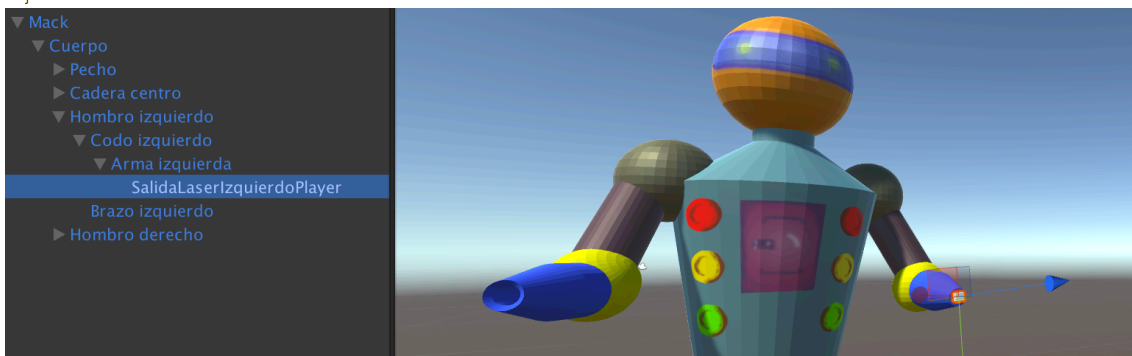
3.1.2 Disparo();

Cuando jugamos en el modo historia podemos disparar un laser, para que esto ocurra le decimos a Game que si detecta que se ha pulsado la tecla “D” accione la función Disparo().

Dentro de dicha función reservamos espacio en memoria para el objeto “laserIzquierdoPlayer” este contendrá una copia de los valores del objeto en juego llamado “SalidaLaserIzquierdoPlayer” y es en esa posición, y rotación donde creamos un nuevo laser como nuevo objeto en el juego. Este objeto tendrá una duración de 3 segundos y se destruirá.

```
if (Input.GetKeyDown (KeyCode.D)) {
    Disparo();
}

public void Disparo(){
    GameObject laserIzquierdoPlayer = GameObject.FindGameObjectWithTag("SalidaLaserIzquierdoPlayer");
    bulletPlayer = (GameObject)Instantiate (laserPlayer, laserIzquierdoPlayer.transform.position, laserIzquierdoPlayer.transform.rotation) as GameObject;
    Destroy(bulletPlayer, 3f);
}
```



Al ser creado el láser, Un script que tiene asociado tomará el mando donde ejecutará un sonido al comenzar y su posición variará según el tiempo transcurrido.


```

void Start () {
    audio.Play();
}

void Update () {
    transform.Translate(Vector3.left*Time.deltaTime*15);
}

```

El objeto laser de nuestro héroe tendrá asociado un tag específico para que el enemigo detecte si ha tenido colisión con este.

3.1.3 PlayerMuere();

En caso de entrar en contacto algún objeto (como puede ser enemigo y laser enemigo, bloque invisible en la profundidad de una caída, una caja metálica en movimiento, etc) que contengan el tag "Pain" activarán el script Pain añadido al Player, este Script y la detección en Game de que la cuenta atrás está a 0 activarán la función de Game "PlayerMuere()".

Una vez ejecutada, esta función detendrá el tiempo de juego, restará una vida del contador de vidas y se actualizará en la memoria del dispositivo mediante "player prefs", de esta manera si una vez perdido una vida se cierra el juego, ya habrá quedado almacenada con antelación.

```

public void PlayerMuere(){
    run = false;
    estamuerto = true;

    vidas--;
    PlayerPrefs.SetInt("TOTALLIFES",vidas);
    player.GetComponent<Player1>().SetMove(true);
    panel.SetActive (true);

    if(vidas > 0) {
        Destroy (player);
    } else if(vidas<=0){
        vidas=0;
        PlayerPrefs.SetInt("TOTALLIFES",vidas);
        SetPerder(true);
        Destroy (player);
        panelGO.SetActive(true);
    }
}

```

Acto seguido comprobamos si tiene más vidas, si las tiene le daremos la opción al usuario de recomenzar desde el principio o de salir al menú de historia, Si no le quedan más vidas saldrá un panel de "Game Over" que le permitira obtener 3 vidas más o salir del nivel con 0 vidas.



Si decide reiniciar se destruirá el objeto player y se ejecutará InstantPlayer()

3.1.4 Instant Player();

La función “InstantPlayer()” ubicada en el Script “Game”, cada vez que iniciamos el nivel, bien sea por primera vez o porque se resucite al player y se vuelva a jugar el nivel, tenemos un objeto fijo en la escena con tag Start que es donde ubicaremos al player cada vez que se ejecute esta función. El sistema de instanciado es del mismo tipo que el que usabamos anteriormente para explicar el disparo.

```
public void InstanPlayer(){
    run = true;

    GameObject span = GameObject.FindGameObjectWithTag("Start");

    player = Instantiate (player, span.transform.position, span.transform.rotation) as GameObject;
    jugador=player.GetComponent<Player1>();
    jugador.enabled = true;
    GameObject.FindGameObjectWithTag("MainCamera").GetComponent<SeguimientoCamara>().setPlayer(player.transform);

    panel.SetActive (false);
    panelGO.SetActive (false);

    tiempo = 35f;
    jugador.setIzq ();
    ActualLevel ();
}
}
```

Cada vez que se activa, estamos indicándole al nivel que debe reiniciarse por lo que se resetea el tiempo y si hay algún panel activo, lo desactiva y llama a la función “ActualLevel()” la cual hemos mencionado cuando hablábamos de cómo se forma el escenario.

3.1.5 SetGanarVidas();

```
public void SetGanarVidas(){// Añadir 1 vida de forma automatica
    vidas++;
}
```

Función que añade una vida al contador actual, llamada si coge mas de cierto número de monedas, gana una vida, pero no se guardará en caso de que el usuario decida abandonar la partida sin finalizar el nivel.

Con esto evitamos que se empiece y termine precipitadamente un nivel con la intención única de ganar vidas pero sin el riesgo de intentar finalizarlo.

3.1.6 SetGanarTiempo();

```
public void SetGanarTiempo(float temp){// ganar tiempo de forma automatica
    tiempo+=temp;
}
```

Cada vez que cogemos monedas llamamos a esta función para incrementar el tiempo de partida.

3.1.7 SetGanar();

Cuando el player atraviesa el engranaje grande de meta se ejecuta SetGanar(), aquí le indicamos al juego que ya se ha jugado una partida, activamos el panel con las opciones de salir del nivel o repetir como el anterior cuando moria y le quedaban vidas pero esta vez actualizamos en “player prefs” el nivel alcanzado (en caso de que no lo hayamos alcanzado anteriormente), para que se active el botón del siguiente nivel en el modo historia.

Guardamos los datos de esta partida (mediante SetInt) en los datos de player prefs en el almacén de datos correspondiente al nivel actual, y solo si se han mejorado por ello leemos (mediante GetInt) y comparamos con los datos previos.

Una vez hecho esto ya podemos sumar los datos globales que son los que se mostrarán en todas las pantallas de menú.

```
public void SetGanar(){
    yaJugado = 1;
    panel.SetActive (true) ;
    Pausa ();
    currentLevel = PlayerPrefs.GetInt ("CURRENTLEVEL");
    totalLevels = PlayerPrefs.GetInt ("TOTALLEVELS");
    //SI EL NIVEL ACTUAL ES MAYOR, ACTUALIZO

    if (currentLevel >= totalLevels) {
        totalLevels=currentLevel+1;

        PlayerPrefs.SetInt ("TOTALLEVELS", totalLevels);
    }

    switch (currentLevel) {
    case 1:

        EnemyKillsTemp = PlayerPrefs.GetInt ("KILLSLEVEL01");
        if (EnemyKillsTemp < EnemyKills) {
            PlayerPrefs.SetInt ("KILLSLEVEL01", EnemyKills);
        }
        scoreTemp = PlayerPrefs.GetInt ("SCORELEVEL01");
        if (scoreTemp < score) {
            PlayerPrefs.SetInt ("SCORELEVEL01", score);
        }
        break;
    case 2:
        EnemyKillsTemp = PlayerPrefs.GetInt ("KILLSLEVEL02");
        if (EnemyKillsTemp < EnemyKills) {
            PlayerPrefs.SetInt ("KILLSLEVEL02", EnemyKills);
        }
        scoreTemp = PlayerPrefs.GetInt ("SCORELEVEL02");
        if (scoreTemp < score) {
            PlayerPrefs.SetInt ("SCORELEVEL02", score);
        }
        break;
    case 3:

        EnemyKillsTemp = PlayerPrefs.GetInt ("KILLSLEVEL01");
        totalKills += EnemyKillsTemp;
        EnemyKillsTemp = PlayerPrefs.GetInt ("KILLSLEVEL02");
        totalKills += EnemyKillsTemp;
        EnemyKillsTemp = PlayerPrefs.GetInt ("KILLSLEVEL03");
        totalKills += EnemyKillsTemp;
        EnemyKillsTemp = PlayerPrefs.GetInt ("KILLSLEVEL04");
        totalKills += EnemyKillsTemp;
        EnemyKillsTemp = PlayerPrefs.GetInt ("KILLSLEVEL05");
        totalKills += EnemyKillsTemp;
        EnemyKillsTemp = PlayerPrefs.GetInt ("KILLSLEVEL06");
        totalKills += EnemyKillsTemp;
        EnemyKillsTemp = PlayerPrefs.GetInt ("KILLSLEVEL07");
        totalKills += EnemyKillsTemp;
        PlayerPrefs.SetInt("TOTALKILLS",totalKills);
        EnemyKills=0;
        //PlayerPrefs.SetInt("KILLS",EnemyKills);

        score = 0;
        scoreTemp = PlayerPrefs.GetInt ("SCORELEVEL01");
        score += scoreTemp;
        scoreTemp = PlayerPrefs.GetInt ("SCORELEVEL02");
        score += scoreTemp;
        scoreTemp = PlayerPrefs.GetInt ("SCORELEVEL03");
        score += scoreTemp;
        scoreTemp = PlayerPrefs.GetInt ("SCORELEVEL04");
        score += scoreTemp;
        scoreTemp = PlayerPrefs.GetInt ("SCORELEVEL05");
        score += scoreTemp;
        scoreTemp = PlayerPrefs.GetInt ("SCORELEVEL06");
        score += scoreTemp;
        scoreTemp = PlayerPrefs.GetInt ("SCORELEVEL07");
        score += scoreTemp;

        PlayerPrefs.SetInt("TOTALSCORE",score);

        score = 0;
        PlayerPrefs.SetInt("SCORE",score);
    }
}
```


3.1.8 SetScore ();

Mediante distintos objetos como pueden ser monedas o al eliminar enemigos, podemos enviar segun la acción diferente cantidad de puntos al contador de puntos, este se incrementa en “SetScore” y lo guarda.

```
public void SetScore(int import){
    score = PlayerPrefs.GetInt ("SCORE");
    score += import;
    PlayerPrefs.SetInt("SCORE", score);
}
```

3.1.9 Morir();

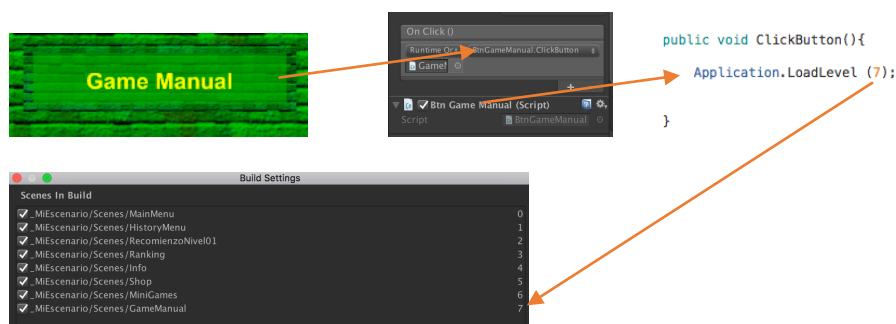
La función “Morir()” suma muertes enemigas en el contador y las guarda.

```
public void Morir(){//suma muertes enemigas
    EnemyKills++;
    PlayerPrefs.SetInt("KILLS", EnemyKills);
}
```

3.2 Botones.

Los botones como objeto llevan asociado un script y el evento de al hacer clic con el ratón en ellos se ejecuta dicho script.

Por ejemplo, el botón de navegación “Game Manual” al pulsarlo se genera la acción asociada al hacer clic, carga en el juego la escena con número 7, el juego va al listado de escenas y carga la que está en dicha posición.



3.3 GameHistoryMenu.

Aplicado al menú de modo Historia, este script lee cual es el nivel máximo que se ha superado y en función de ello mediante un switch case, activa o desactiva los botones de los distintos niveles.

```
totalLevels = PlayerPrefs.GetInt ("TOTALLEVELS");
try{
    switch (totalLevels){
        case 1:
            GameObject.FindGameObjectWithTag("1").SetActive(true);
            GameObject.FindGameObjectWithTag("2").SetActive(false);
            GameObject.FindGameObjectWithTag("3").SetActive(false);
            GameObject.FindGameObjectWithTag("4").SetActive(false);
            GameObject.FindGameObjectWithTag("5").SetActive(false);
            GameObject.FindGameObjectWithTag("6").SetActive(false);
            GameObject.FindGameObjectWithTag("7").SetActive(false);

            break;
        case 2:
            GameObject.FindGameObjectWithTag("1").SetActive(true);
            GameObject.FindGameObjectWithTag("2").SetActive(true);
            GameObject.FindGameObjectWithTag("3").SetActive(false);
            GameObject.FindGameObjectWithTag("4").SetActive(false);
            GameObject.FindGameObjectWithTag("5").SetActive(false);
            GameObject.FindGameObjectWithTag("6").SetActive(false);
            GameObject.FindGameObjectWithTag("7").SetActive(false);
            break;
        case 3:
```

3.4 Level1.

Cada botón de nivel llevará asociado un script como éste pero cambiando el valor de "currentLevel" por el valor del nivel que se ha clicado, de esta manera cargara siempre la escena almacenada en la posición 2.

Al iniciar el juego en modo Historia, éste sabrá que bloques colocar según la actualización en playerPrefs, no se está actualizando directamente la función del nivel actual en el script de Game porque aún no ha sido cargado por lo que no encontraría sus funciones.

```
public void ClickButton(){
    currentLevel = 1;
    PlayerPrefs.SetInt ("CURRENTLEVEL", currentLevel);
    SceneManager.LoadScene(2);
```

3.5 Player1.

Player1 es uno de los Scripts más importantes del juego, éste se encarga del desplazamiento de nuestro robot protagonista, en función de si pulsamos las flechas del teclado, irá a la izquierda, a la derecha, y saltará.

Tienen un efecto de continuidad, si se mantienen sigue hacia esa dirección pero si la sueltan este parara, no es apretar una vez.

De esta manera cuando saltamos podemos controlar un salto corto o un salto más largo y desde este script controlaremos también si esta saltando para después recoger ese valor desde el script "SeguimientoCamara()" y actuar diferente

Mediante "Invoke("upFin", 1.0);" indicamos si saltamos se llama a la función upFin con un retraso de 1 segundo y cambia la fuerza del salto, así cuando empieza a saltar lo hace con más fuerza pero aunque se mantenga el botón de salto, limitamos que siga "flotando" nuestro personaje.

```
void FixedUpdate () { // es llamado una vez por frame

    if (!mover){

        if (Input.GetKey (KeyCode.UpArrow)) {
            //si pulsa tecla hacia arriba, salto
            up();
            Invoke("upFin", 1.0f);
            //PlayerPrefs.SetInt("SALTANDO",1);
        }else{
            saltando=false;
        }
        if (Input.GetKeyUp (KeyCode.UpArrow)) {
            //si pulsa tecla hacia arriba, salto
            anim.SetBool("Saltando",false);
        }
        if (Input.GetKey (KeyCode.LeftArrow)) {
            izquierda();
        }
        if (Input.GetKeyDown (KeyCode.LeftArrow)) {
            anim.SetBool("Corriendo",false);
        }
        if (Input.GetKey (KeyCode.RightArrow)) {
            derecha();
        }
        if (Input.GetKeyDown (KeyCode.RightArrow)) {
            anim.SetBool("Corriendo",false);
        }
    }
}

public void izquierda(){
    anim.SetBool("Corriendo",true);

    if (saltando) {
        transform.Translate (Vector3.left * velocidad / 2 * Time.deltaTime / 2, Space.World);
        PlayerPrefs.SetInt ("SALTANDO", 1);
    }
    if (!saltando) {
        transform.Translate (Vector3.left * velocidad / 2 * Time.deltaTime, Space.World);
        PlayerPrefs.SetInt ("SALTANDO", 0);
    }
    if (izq) {
        transform.Rotate (new Vector3 (0, 180f, 0));
    }
    izq = false;
    PlayerPrefs.SetInt("IZQDER",0);
}

public void derecha(){
    if (saltando) {
        transform.Translate (Vector3.right * velocidad / 2 * Time.deltaTime / 2, Space.World);
        PlayerPrefs.SetInt ("SALTANDO", 1);
    }
    if (!saltando) {
        transform.Translate (Vector3.right * velocidad / 2 * Time.deltaTime, Space.World);
        PlayerPrefs.SetInt ("SALTANDO", 0);
    }
    if (izq == false) {
        transform.Rotate (new Vector3 (0, 180f, 0));
    }
    izq = true;
    PlayerPrefs.SetInt("IZQDER",1);
}

public void up() {
    anim.SetBool("Saltando",true);
    transform.Translate (Vector3.up * salto * ((Time.deltaTime)/15) , Space.World);
    PlayerPrefs.SetInt("SALTANDO",1);
}

    saltando=true;
}

public void upFin() {
    estaSaltando = PlayerPrefs.GetInt ("SALTANDO");
    if (estaSaltando == 1) {
        transform.Translate (Vector3.up * salto * (-1)*((Time.deltaTime)/55) , Space.World);
    }
    anim.SetBool("Saltando",true);
    PlayerPrefs.SetInt("SALTANDO",1);
}

    saltando=true;
}
}
```

3.6 SeguimientoCamara

El script SeguimientoCamara al comenzar busca la posición del player mediante la función “LocalizaJugador()” durante el juego en cada frame va recolocando la camara según la posición x del jugador.

A su vez pregunta si el jugador está cerca del suelo y también si está saltando, mientras no se aleje de cierta distancia del suelo, la posición de la coordenada “y” no varía.

De ese modo si salta vemos lo que hay debajo, pero por otro lado si se aleja a cierta distancia del suelo, la cámara debe recoger la posición de su coordenada “y” de lo contrario al subir demasiado lo perderíamos de vista

```
void start() {
    LocalizaJugador();
}

void LateUpdate () {
    suelo = 0;
    suelo= PlayerPrefs.GetInt ("CAMARASUELO");
    salto = PlayerPrefs.GetInt ("SALTANDO");
    izqDer = PlayerPrefs.GetInt ("IZQDER");

    Vector3 temp = posCamera;
    temp.x = Player1.position.x+7;
    temp.z=-11.81f;

    if (suelo == 1) {
        temp.y = Player1.position.y;

        transform.position = temp + posCamera;
    }
    if (suelo == 0) {
        temp.y = altura;
        transform.position = temp;
    }
}

}

public void LocalizaJugador(){
    Player1 = GameObject.FindGameObjectWithTag ("Player").transform;
}

public void setPlayer(Transform player){
    this.Player1 = player;
}
}
```

3.7 KillEnemy.

El script killEnemy situado en el enemigo, detectara la entrada de una colisión por parte de un objeto, si este se llama “PiePlayer”, tag que hemos asignado al pie del player y al laser del player. Marcará en el Script llamado Enemigo la función setTocado con el parametro tocado= true.

```
Game game;
Enemigo enemigo;
void start () {
    game = GameObject.FindGameObjectWithTag ("Game").GetComponent<Game>();
    enemigo = GameObject.FindGameObjectWithTag ("Enemigo").GetComponent<Enemigo>();
}
//detecta que colisiona con la cabeza del enemigo
void OnTriggerEnter(Collider obj){
    if(obj.transform.tag.Equals("PiePlayer")){
        transform.GetComponentInParent<Enemigo>().SetTocado(true);
    }
}
```

3.8 Enemigo.

Cuando se detecta un toque, se ejecuta una función que por un lado ejecuta la función del “game” “morir” de la que ya hemos hablado en el script de game. Acto seguido activa un efecto de una explosión mediante partículas propias de Unity y destruye el objeto del enemigo.

Mas adelante habrá un contador tras detectar como verdadera la variable booleana “tocado” para ejecutar un contador de toques, según la dificultad del enemigo que no lo mate si no traspasa el numero asignado a ese enemigo.

```
Game game;

void Start () {
    positionInitial= transform.position;
    desde.x=positionInitial.x+range/2;
    hasta.x=positionInitial.x-range/2;
    game = GameObject.FindGameObjectWithTag ("Game").GetComponent<Game>();
}

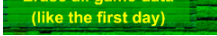
public void SetTocado(bool golpeado){
    tocado = golpeado;
}

void matame(){
    game.Morir();
    Invoke("Explosion", 0.0f);
    Destroy(transform.gameObject,0f);
    tocado=false;
}

void Update () {
    if(dir){
        transform.Translate(Vector3.right*Time.deltaTime*3);
        if(transform.position.x<hasta.x) {
            dir=false;
        }
    }
    if (!dir){
        transform.Translate(Vector3.left*Time.deltaTime*3);
        if(transform.position.x>desde.x){
            dir=true;
        }
    }
    if (tocado) |
        matame ();
}

public void Explosion(){
    Instantiate(Particula,transform.position,transform.rotation);
}
```

3.9 Erase.

El Script Erase, mediante un evento al hacer clic en el botón  que está situado en el menú "Game Manual", modificará todos los datos almacenados en "PlayerPrefs" para que el juego se encuentre como el primer día.

```
public int score=0;
public int totalScore=0;
public int kills=0;
public int totalKills=0;
public int currentLevel=1;
public int totalLevels=1;
public int totalLives=3;
public int yaJugado=0;

public void ClickButton(){

    PlayerPrefs.SetInt("SCORE",score);
    PlayerPrefs.SetInt("TOTALSCORE",totalScore);
    PlayerPrefs.SetInt("TOTALKILLS",kills);
    PlayerPrefs.SetInt("KILLS",kills);
    PlayerPrefs.SetInt("CURRENTLEVEL",currentLevel);
    PlayerPrefs.SetInt("TOTALLEVELS",totalLevels);
    PlayerPrefs.SetInt("TOTALLIFES",totalLives);
    PlayerPrefs.SetInt("YAJUGADO",yaJugado);
```

3.10 MetaFinish.

El Script "MetaFinish" está asignado al trigger del colider del engranaje grande que hay al final de cada nivel, éste detectará si entra en colisión con Player y si es así detendrá el tiempo, y llamará a la función de Game, llamada SetGanar(), ya mencionada anteriormente.

```
public float velocidad = 0;
public GameObject GameScene;
Game game;

void Awake () {
    game = GameObject.FindGameObjectWithTag ("Game").GetComponent<Game>();
}

// Update is called once per frame
void Update () {
    transform.Rotate (velocidad * Time.deltaTime, 0, 0);
}

void OnTriggerEnter(Collider obj){
    if (obj.transform.tag.Equals ("Player")) {

        game.SetGanar();
        Destroy(transform.gameObject);
    }
}
```



3.11 TablaHorizontal

Las tablas que están en movimiento tienen el Script tabla horizontal, otras de ellas tienen otro igual, pero cambiando la dirección, utilizado también para el movimiento de las cajas metálicas pero modificando las coordenadas “y” en lugar de las coordenadas en “x”. Además, se mantendrán siempre dentro del rango indicado.

Solo en el caso de las tablas, (las cajas no) contienen las funciones “OntriggerStay()” y “OntriggerExit()” con la asignación o eliminación respectivamente, de asignar al objeto que entra en colisión la posición como posición hija de las tablas.

Esto quiere decir que cuando el Player entra en colisión con las plataformas o se posa sobre ellas, adquiere la dirección y velocidad de estas, siendo arrastrado por ellas, de otra manera se quedaría en el lugar y caería.

```

public float range=20;

public Vector3 posTabla= new Vector3(0f,0f,0f);

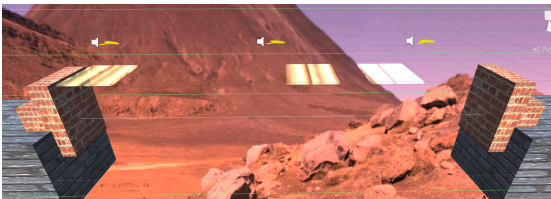
void Start () {
    positionInitial= transform.position;
    desde.x=positionInitial.x+range/2;
    hasta.x=positionInitial.x-range/2;
}

void OnTriggerEnter(Collider other)
{
    other.transform.parent = transform;
}

void OnTriggerExit(Collider other)
{
    other.transform.parent = null;
}

void Update () {
    if(dir){
        transform.Translate(Vector3.left*Time.deltaTime*3);
        if(transform.position.x<hasta.x) {
            dir=false;
        }
    }else{
        transform.Translate(Vector3.right*Time.deltaTime*3);
        if(transform.position.x>desde.x){
            dir=true;
        }
    }
}
}

```



3.12 TimeController

Utilizamos el script TimeController para que el tiempo corra según el tiempo real, de otra manera iría mas rápido o más lento según la potencia del ordenador o dispositivo que ejecute el juego.

```

public class TimeController : MonoBehaviour {
    public static float deltaTime;
    private static float _lastframetime;
    void Start () {
        _lastframetime = Time.realtimeSinceStartup;
    }

    void Update () {
        deltaTime = Time.realtimeSinceStartup - _lastframetime;
        _lastframetime = Time.realtimeSinceStartup;
    }
}

```


3.13 Txt (control de textos).

Todos los textos que varían durante el juego, son modificados mediante scripts únicos para ellos así pues el contador kills del juego en modo historia recogería el valor mediante la lectura recibida en el parámetro temporal SetKills(temp).

Lo asignaría a una variable con la que puede trabajar desde este ámbito y la convierte para ser mostrada como texto.



```
Text txt;
private int currentKills=0;

// Use this for initialization
void Start () {
    txt = gameObject.GetComponent<Text>();
    txt.text=" " + currentKills;
}
public void SetKills(int temp){
    currentKills=temp;
}

// Update is called once per frame
void Update () {
    txt.text=" " + currentKills;
}
}
```

5. Audio.

5.1 Música de fondo durante la partida.

<https://freesound.org/people/LittleRobotSoundFactory/sounds/321039/>

Attribution 3.0 Unported (CC BY 3.0), mencionada en el juego en el apartado "Info&Contact"

Elegí esta música por que le da dinamismo y acción al juego, es muy adecuada para este juego y acaba siendo adictiva, además es de libre uso con mención.

5.2 Efectos sonoros.

Sonido del láser, Sonido de la moneda al cogerla, sonido de salto

Creación propia mediante el programa gratuito Bfxr

link de descarga: <https://www.bfxr.net/>

Esta aplicación gratuita ayuda a evitar los clásicos problemas de derechos de autor, ayudando a generar muchos tipos de sonidos y adaptarlos a las necesidades del juego.

6. Conclusiones

Como conclusión principal se podría decir que crear un videojuego no es tarea fácil y el tiempo nunca juega a tu favor, pero ha sido una grata experiencia que va a permitir seguir desarrollando y optimizando el juego para una posterior publicación para plataformas móviles

Inicialmente se había marcado como objetivo crear un juego de un nivel para Smartphones, si bien se han creado 7 niveles, Hubo ciertos problemas en la creación de los paneles táctiles en la pantalla para móviles, por lo que el fichero generado y la orientación de cara al TFG ha cambiado a plataforma de sobremesa.

La planificación y metodología se han seguido en medida de lo posible, los comentarios del profesor aparte de ayudar a solucionar algún problema, también han aportado ideas nuevas, como la modificación en el salto para que no parezca que flote, ahora funciona mejor. Así como la corrección en el desplazamiento de las tablas.

Por limitación de tiempo no se ha extendido más el proyecto, ni tampoco era este el objetivo inicial, pero va a ser continuado y perfeccionado a nivel gráfico y en su extensión, más niveles, mini juegos desbloqueados conforme se avanza en el modo historia, el Juego en sí será gratuito, pero se ofrecerá la oportunidad de desbloquear objetos, niveles y más vidas mediante compras.

Se ha aprendido mucho durante el desarrollo del mismo a nivel de diseño, programación, y temporalización del trabajo, además cada vez que se juegue a un Videojuego ahora se va a poder intuir como está controlada cada acción. Pero si algo hay seguro es que siempre se continuara aprendiendo.

7. Glosario

G.D.D. (Game Design Document): Documento del diseño del videojuego.

Player Prefs: Variable utilizada por Unity para almacenar datos en el dispositivo, la información permanecerá hasta que sea borrada.

Script: Grupo de código con un mismo propósito o propósitos similares.

Función: líneas de código dentro de un script con un propósito directo.

Booleano: variable que solo puede tomar uno entre 2 valores.

Trigger: Ejecuta acción predeterminada de un colider si algo entra en su rango.

Colider: rango de espacio de un objeto, aunque se puede modificar su dimensión de manera independiente al objeto. Se utiliza para detectar colisiones entre objetos.

Switch case: selección entre varias opciones segun resulte una variable dada.

Prefab: Agrupamiento de uno o varios objetos de Unity que al ser llamado el padre los hijos mantienen sus posiciones y comportamientos respecto al padre.

8. Bibliografía

- Web:
<https://docs.unity3d.com/es/current/Manual/UnityManual.html>
múltiples visitas [5 de marzo de 2018 a 10 de Junio de 2018]
- Web:
<https://answers.unity.com/> múltiples visitas [5 de marzo de 2018 a 10 de Junio de 2018]
- Web:
<http://www.unityspain.com/> múltiples visitas [5 de marzo de 2018 a 10 de Junio de 2018]