Desarrollo de un juego 3D en Unity

“Rooms”

CFGS Desenvolupament d'Aplicacions Multiplataforma

Nom estudiants participants

Víctor Gil Pes

Daniel Ruz Vázquez

Curs acadèmic

DAM 2-A

Data Lliurament

30/05/2017

C) Copyright

Resumen del proyecto (màxim 250 paraules):
Este proyecto trata sobre el desarrollo de un pequeño juego en un entorno 3D utilizando el motor gráfico de Unity y el lenguaje de programación C#.

Hablaremos de nuestro diseño inicial, los primeros planteamientos, así como los diferentes niveles, código esencial para su funcionamiento, optimización y demás.
En el proyecto se podrán ver los distintos diagramas diseñados en un inicio, las motivaciones para realizar este proyecto así como una parte mucho más técnica en la que detallamos cómo funcionan algunos de los elementos más icónicos que tiene el juego.

Es notable destacar también que explicaremos un poco cómo utilizamos y nos desenvolvemos con el uso de Unity para la realización y consolidación gráfica de los distintos niveles que hemos diseñado.

Abstract (in English, 250 words or less):
This project is about the development of a small and homecrafted game in a 3D environment using the cross-platform game engine Unity and the programming language C#.

We will explain further our original design, our first approaches to both designing and the whole concept of the game, the levels, essential code for it to work, optimization and more.
In this project you will see the different diagrams that we designed at the very start, our motivations for starting such a project and also a more tricky and technical section where we talk and explain with the required simplicity the “how does this work” of the most iconic elements that our game has.

It is remarkable to highlight that we will also explain with a little bit more of detail how do we use and how do we work with Unity’s usage to build and create all those graphics elements of the different levels of our game.

Paraules clau (entre 4 i 8):
Unity, c#, Visual Studio, GameObject, Script

Índice

1. Introducción

1.1 Contexto y justificación del proyecto

Nuestra motivación principal era la de realizar un juego en 3D ya que había otros proyectos en 2D y nos parecía interesante realizar algo en 3D. Además, la opción de poder aprender otro lenguaje como C# que está en alza junto a Unity nos parecía una buena iniciativa para poder aplicar los conocimientos de estos dos años en algo diferente e interactivo.

1.2 Objetivos del trabajo

- Realizar un juego en 3D
- Aprender otro lenguaje de programación (C#)
- Aprender a utilizar un motor gráfico como Unity
- Descubrir un poco el mundo del desarrollo de videojuego.

1.3 Enfoque y método seguido

Para realizar este trabajo, nos dividimos las tareas en unos aspectos muy claros y básicos en su inicio. Por una parte, realizar un cierto número de niveles para dar un acabado robusto al proyecto. Por otra, realizar desde cero un proyecto decente en 3D que implique al usuario para resolver la mayoría de sus pantallas.

1.4 Planificación del proyecto

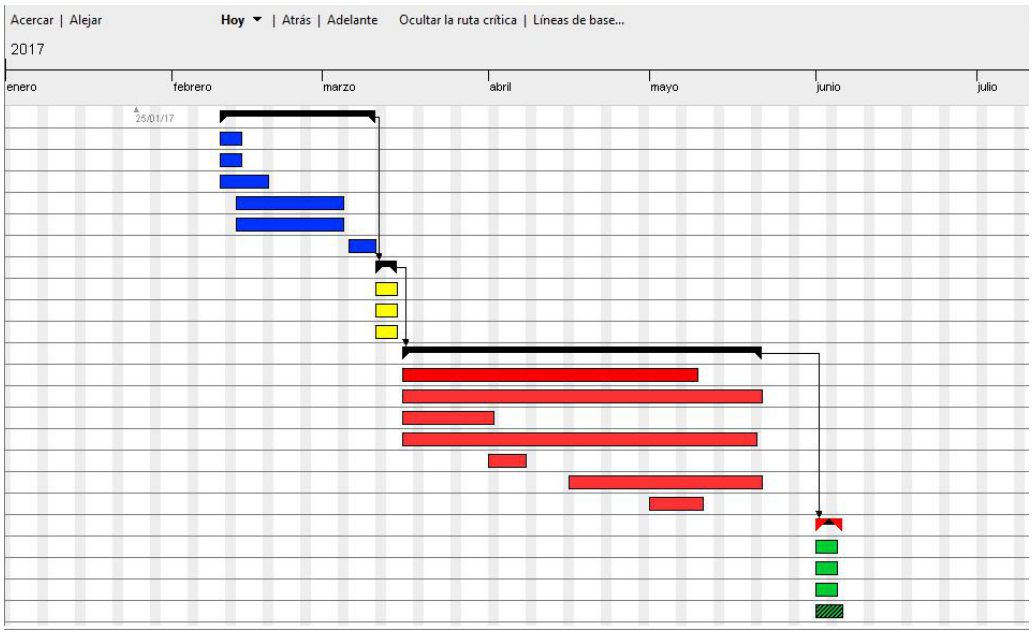
Para realizar este proyecto necesitábamos tres herramientas fundamentales: Windows, Visual Studio *(de ahora en adelante VS)* y Unity. Nuestra elección por Windows y VS viene justificada por la robustez que presenta este editor junto a Unity. No obstante, mediante unos sencillos pasos sí que se podría adaptar el proyecto para el resto de sistemas operativos.

Una vez preparadas las herramientas básicas, realizamos una serie de diagramas para tratar de organizar, orientar y preparar la organización de las tareas a realizar para llevar a cabo el proyecto.

Sin embargo, cabe destacar que la complejidad en la que ha derivado el proyecto y el funcionamiento específico de Unity han hecho que todos estos diagramas, que en un principio diseñamos con una cierta viabilidad, han quedado totalmente eclipsados y obsoletos para que puedan representar fielmente cada uno de los niveles que componen nuestro juego.

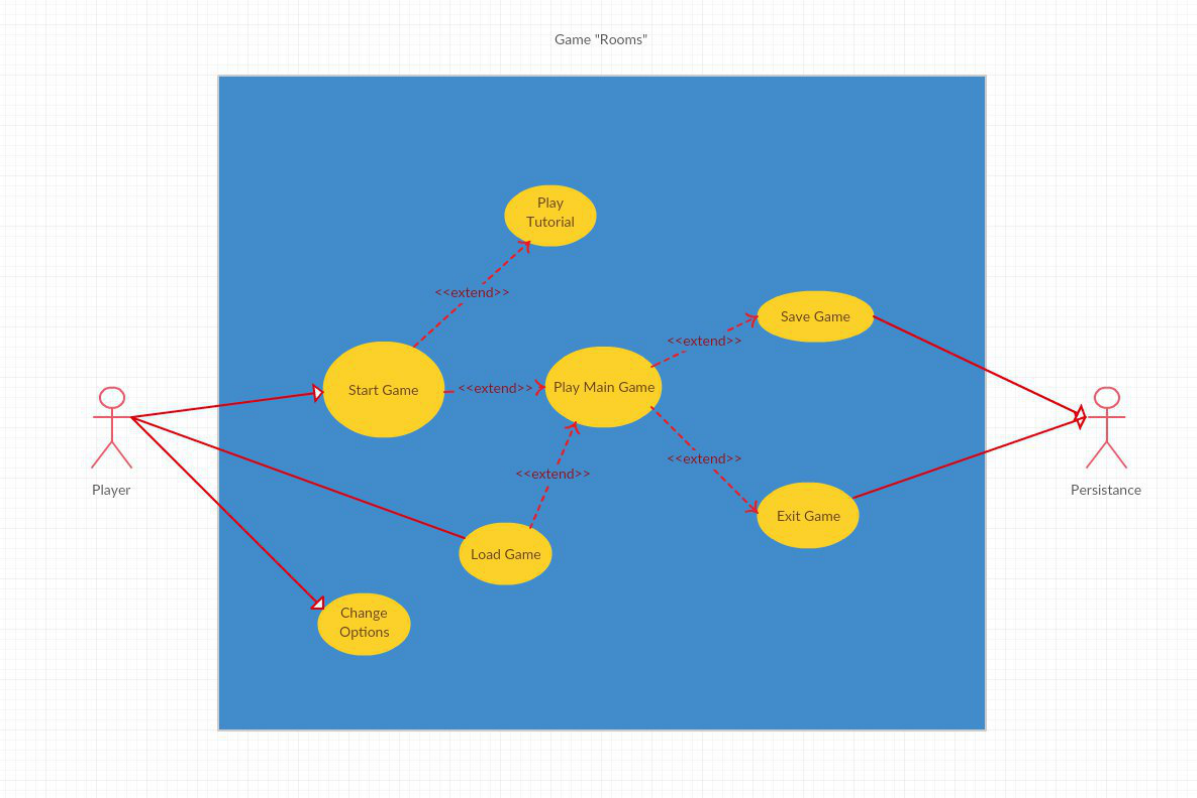
A continuación adjuntamos nuestro diagrama de Gantt sobre la planificación del proyecto.

Fase Inicial	10/02/17	10/03/17
Elaboración del informe p...	10/02/17	13/02/17
Decisión del tipo de juego	10/02/17	13/02/17
Preparación de requisitos	10/02/17	18/02/17
Aprendizaje C#	13/02/17	4/03/17
Aprendizaje motor Unity	13/02/17	4/03/17
Elaboración 'Entrega 1' (PI...	6/03/17	10/03/17
Segunda fase	11/03/17	14/03/17
Elaboración diagrama de ...	11/03/17	14/03/17
Elaboración diagrama cas...	11/03/17	14/03/17
Desarrollo primeros nivele...	11/03/17	14/03/17
Tercera fase	16/03/17	21/05/17
Desarrollo juego	16/03/17	9/05/17
Aplicación de recursos grá...	16/03/17	21/05/17
Programación de Scripts b...	16/03/17	1/04/17
Diseño y programación de...	16/03/17	20/05/17
Elaboración 'Entrega 2'	1/04/17	7/04/17
Pruebas de errores	16/04/17	21/05/17
Elaboración 'Entrega 3'	1/05/17	10/05/17
Fase final	1/06/17	5/06/17
Bugfixing	1/06/17	4/06/17
Beta testing	1/06/17	4/06/17
Polishing	1/06/17	4/06/17
Preparar exposición	1/06/17	5/06/17
Elaboración 'Memoria final'	1/06/17	5/06/17



2. Diseño

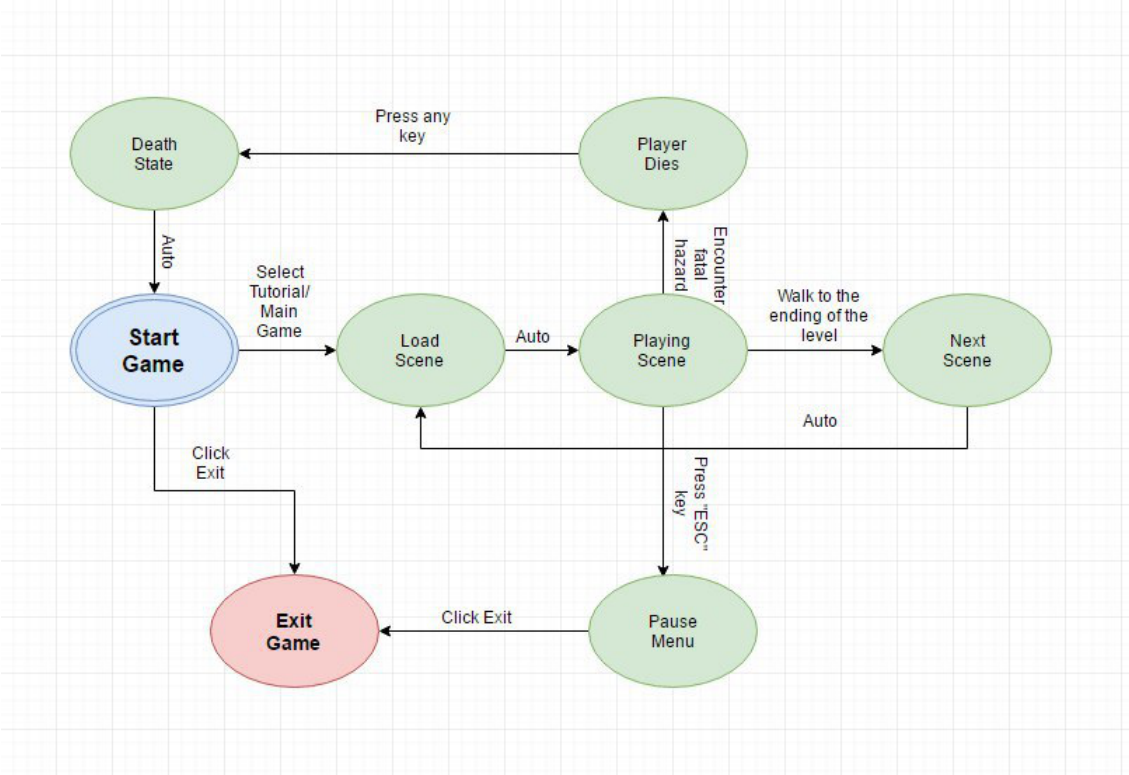
2.1.- Casos de uso



Dado a la gran cantidad de clases que disponemos en nuestro proyecto, hemos decidido hacer un diagrama de clases mostrando el funcionamiento básico y general del juego. Cabe destacar que este diseño es muy primitivo ya que el funcionamiento de Unity nos limitaba el hacer un diagrama estándar.

Como se ve en la imagen, el jugador al iniciar el juego puede escoger entre jugar un tutorial o empezar una partida. Luego, el usuario cuando esté jugando puede guardar la partida o salir de la misma. También puede escoger unas opciones personalizadas por si desease cambiar los controles.

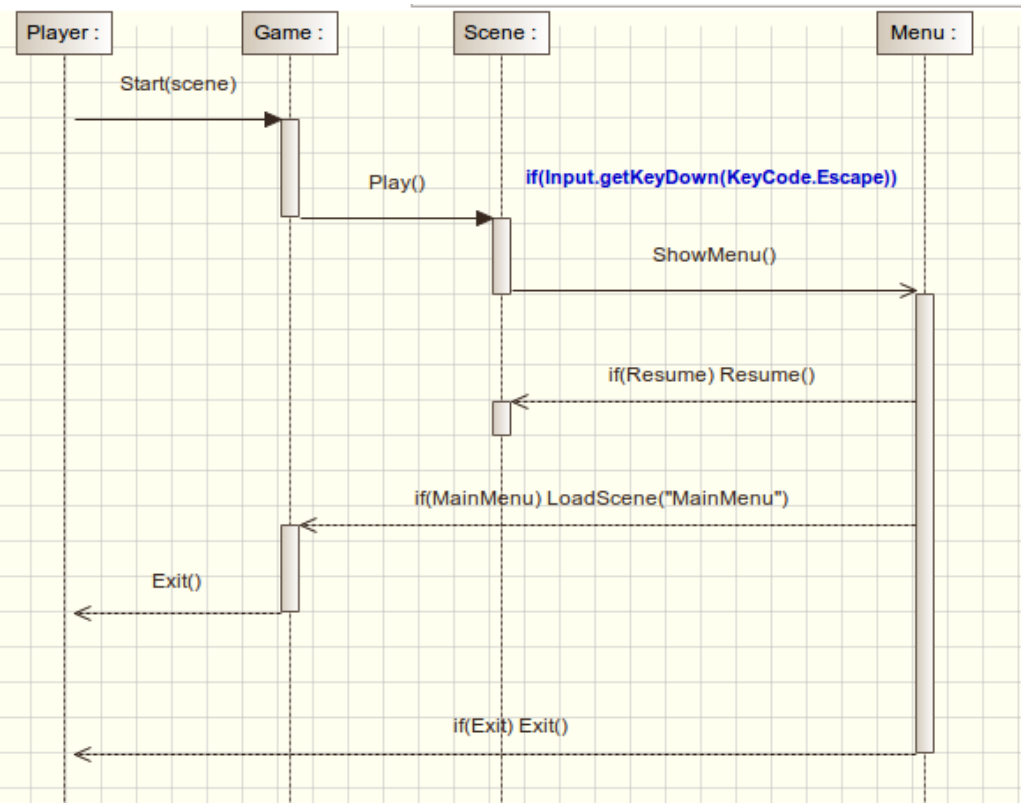
2.2.- Diagrama de estado



Este pequeño diagrama intenta definir de una manera sencilla y algo simple el funcionamiento típico de cualquier nivel desde que se inicia el juego hasta que se cierra éste. Debemos destacar que hay ciertos elementos del juego que influirían en este diagrama debido al estado actual en el que se encuentran, pero para dar una visión más general y no diseñar distintos diagramas para diferentes niveles hemos optado por quedarnos con este.

Para ello podemos ver cómo sería el ciclo común de nuestro juego, iniciándonos en *Start Game* y viendo cómo el transcurso normal trataría de cargar el nivel, jugarlo, pausar o morir o pasar a la siguiente pantalla. Tratamos de definir en cada rama las acciones que desencadenan cada estado.

2.3.- Diagrama de secuencia



En este diagrama de secuencia se muestra el funcionamiento general de nuestro juego. Siempre se sigue la misma mecánica. El jugador inicia la partida y empieza el juego (Start). Luego, mientras el jugador se encuentra en la escena resolviendo los puzzles, puede pausar la partida presionando la tecla escape. Entonces, se muestra un pequeño menú con tres botones:

- Resume: cuando el jugador clique ese botón, se reanuda la partida.
- Main Menu: vuelve al menú principal.
- Exit: el jugador sale del juego.

2.4 Cambio de diseño

Una vez explicados los diagramas en los dos apartados anteriores queremos justificar su desuso en toda la posterioridad del proyecto. Hemos optado por tomar esta decisión debido al funcionamiento del motor gráfico de Unity, utilizando las clases (que en general siguen una estructura similar a todos los lenguajes orientados a objetos) pero que en este caso no las emplea de manera secuencial ni de la forma programática más típica. Es por ello que, hacer casos de uso para cada una de las casi cincuenta clases con las que cuenta este proyecto era prácticamente imposible y una pérdida valiosa de tiempo, ya que muchas de las acciones de todas estas clases son prácticamente anecdóticas en la mayoría de niveles.

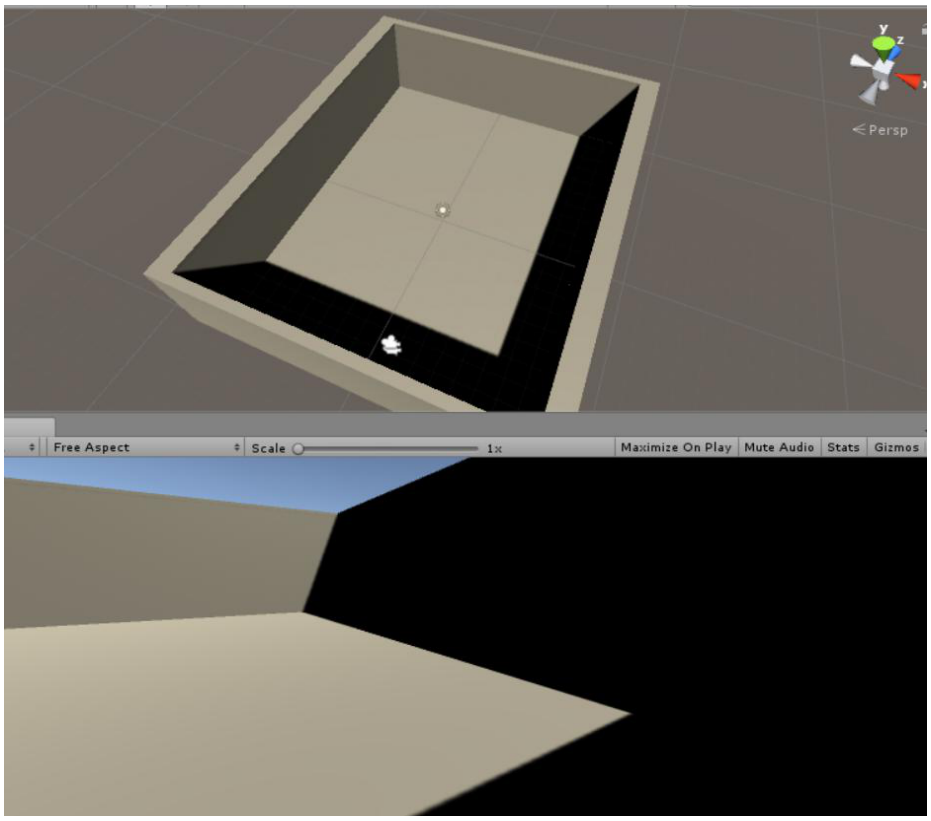
Por lo tanto, los diagramas presentados en los apartados anteriores únicamente reflejan un estado muy generalista del comportamiento del proyecto en su totalidad y, como hemos comentado, hacer diagramas para tratar de incorporar tal cantidad de elementos de una manera legible y coherente no es una tarea baladí.

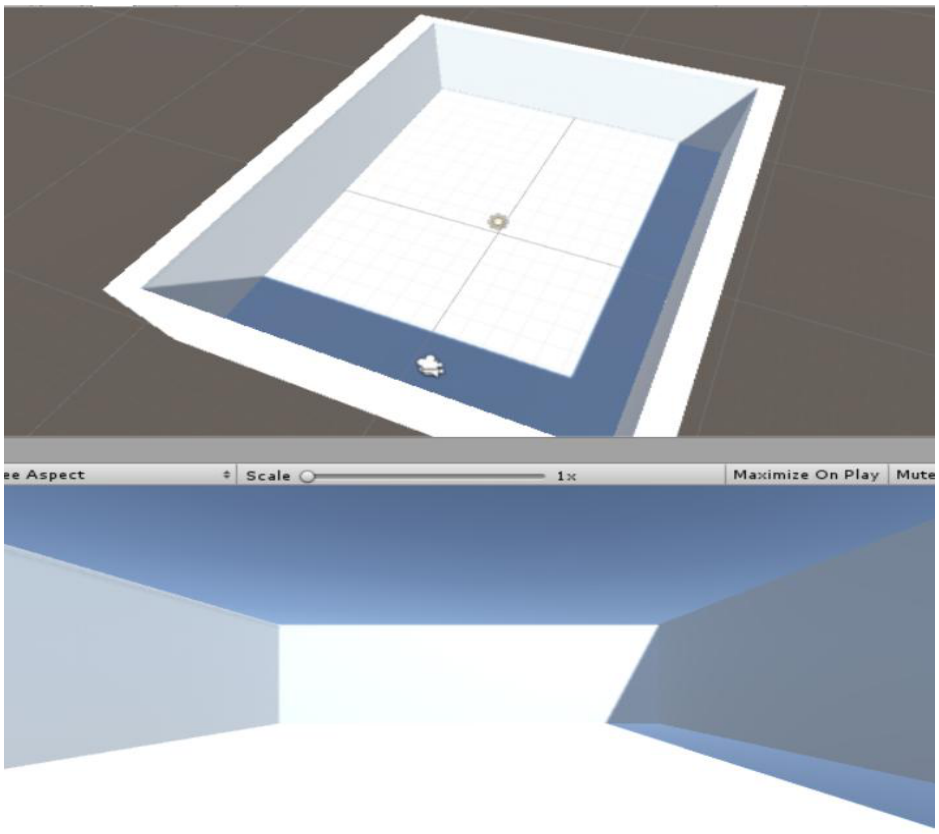
3.- Desarrollo

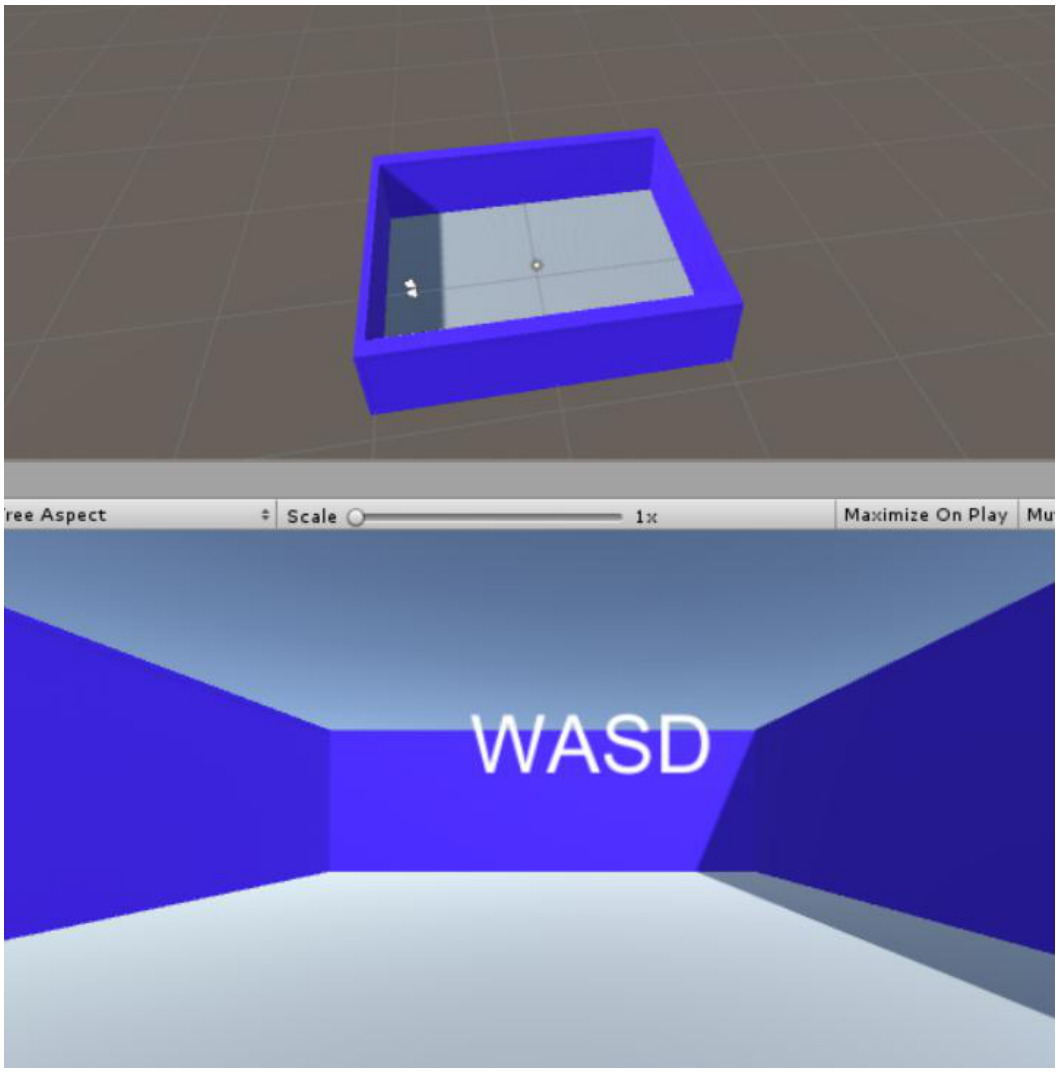
Dado que no teníamos conocimientos de C# ni de Unity, primero tuvimos que realizar pequeños proyectos para aprender las principales funcionalidades de Unity, así como adaptarnos al tipo de programación que supone C#.

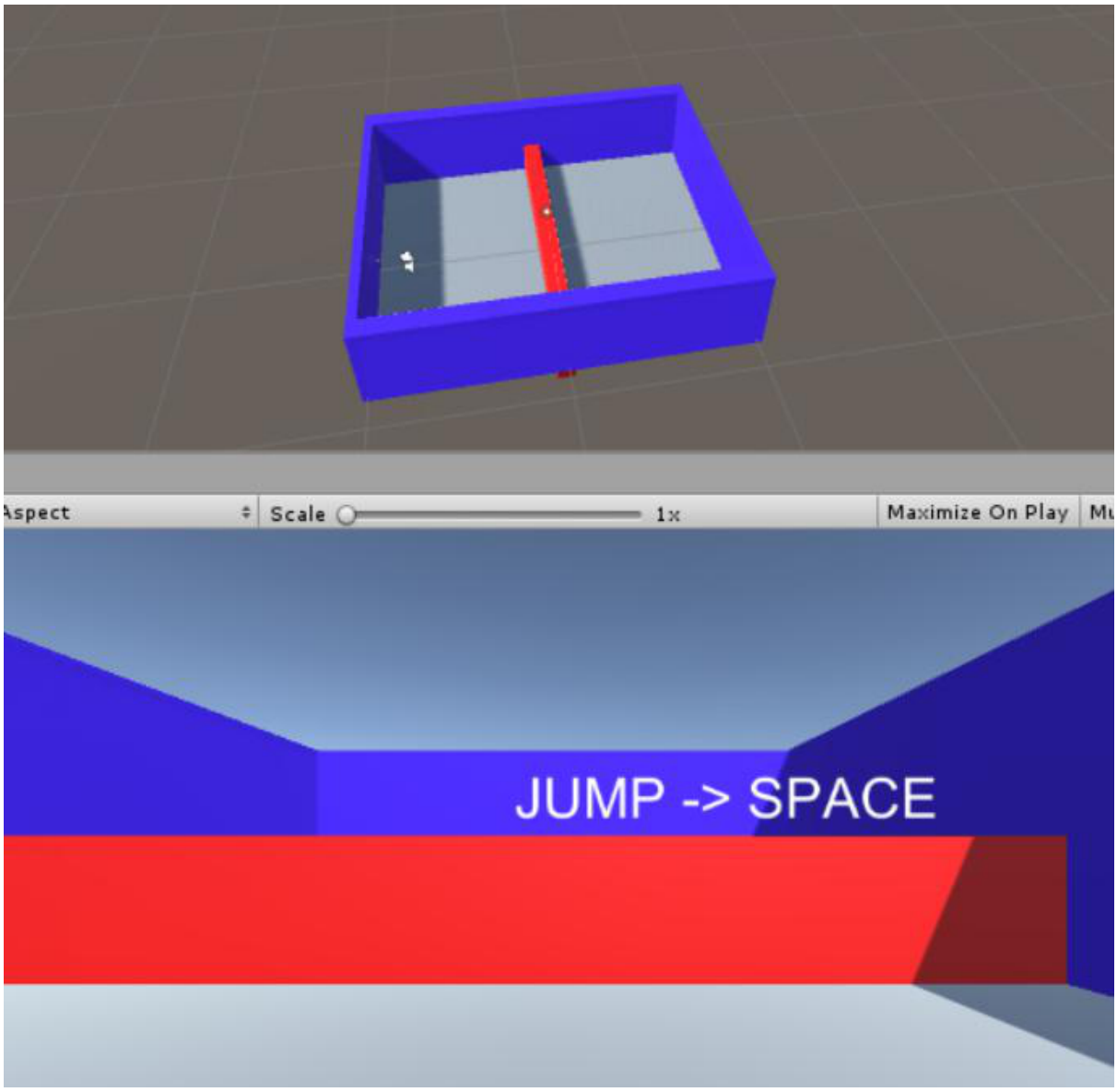
3.1 Desarrollo inicial

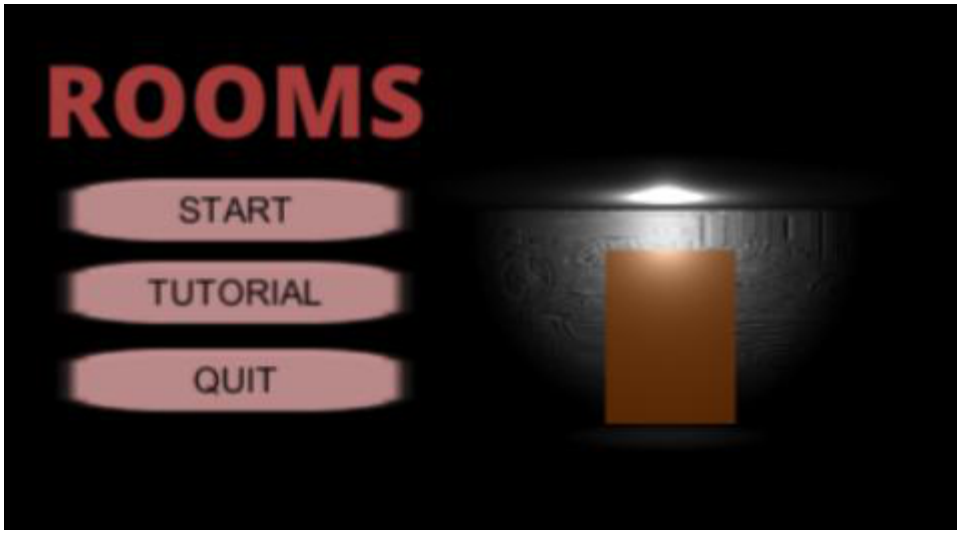
Para acabar de familiarizarnos con este nuevo entorno, pensamos que la mejor manera de poder acabar de aprender este mundo, decidimos hacer unos pequeños niveles de tutorial para que el usuario aprenda los controles básicos del juego. Este tutorial consiste en tres escenas básicas en las que el jugador debe de encontrar la manera de abrir una puerta.


Persp
Free Aspect
Scale
1x
Maximize On Play
Mute Audio
Stats
Gizmos


Scale
1x
Maximize On Play


ree Aspect
Scale
1x
Maximize On Play
Mu
WASD


Aspect
Scale
1x
Maximize On Play
JUMP -> SPACE


ROOMS
START
TUTORIAL
QUIT

En estas imágenes podemos observar el diseño inicial de los primeros niveles que desarrollamos.

Tomando de ejemplo la primera imágen, hay un jugador principal que tiene un script que le permite el movimiento. Dicho personaje se puede ver en la imágen con este icono  . Simplemente consiste en la cámara principal de la escena que puede ser controlada tanto para moverse por la sala o para rotar la imagen.

Como aún no habíamos creado la puerta, hicimos un apaño que simulaba que el jugador abría la puerta y pasaba a otro nivel. Para ello, hicimos un script que cuando el personaje principal tocaba esa pared, el nivel cambiaba. Este script controlaba que cuando el usuario entraba a una zona determinada (cerca de la pared), la escena cambiaba:

```
void OnTriggerEnter(Collider other){
      SceneManager.LoadScene("Tutorial 2");
}
```

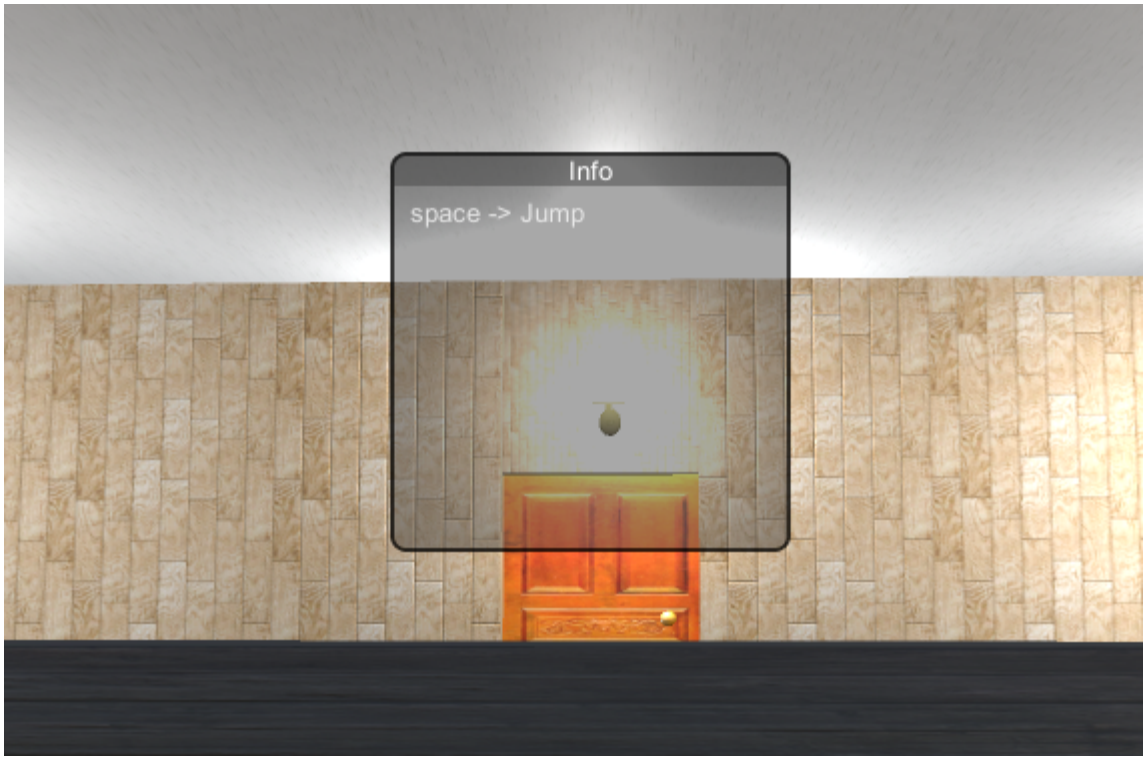
OnTriggerEnter es un método que hemos utilizado mucho a lo largo del proyecto ya que nos ayuda a que ocurran diferentes eventos en el juego cuando el usuario esté en una zona determinada.

3.2 Primeros niveles

Los primeros niveles que el jugador puede realizar es el tutorial explicado previamente. En él, se pretende que el usuario aprenda los controles básicos del juego y se familiarice con él mostrándole ciertas ayudas.

En esas tres imágenes se ve el diseño final de los primeros niveles. Todos tienen la misma funcionalidad: el usuario debe de llegar a la puerta y abrirla.

Para explicar todo el funcionamiento, nos centraremos en el segundo nivel del tutorial: El jugador se encuentra en una pequeña sala con un pequeño muro en medio que le impide llegar a la puerta. Para ello, debe saltar ese muro presionando la tecla espacio. Previamente se le indica al jugador cuál es la tecla para saltar.



Dicho esto, cuando el jugador se acerca al muro solamente debe de presionar esa tecla y saltar. Sin embargo, si toca el muro, la escena se reinicia y el personaje principal vuelve a su posición inicial.

Cuando el jugador consigue saltar sin tocar el muro, lo único que le queda por hacer es ir a la puerta y abrirla. Para ello, le sale un cartel informativo como el de la imágen anterior que le indica cómo abrir la puerta.

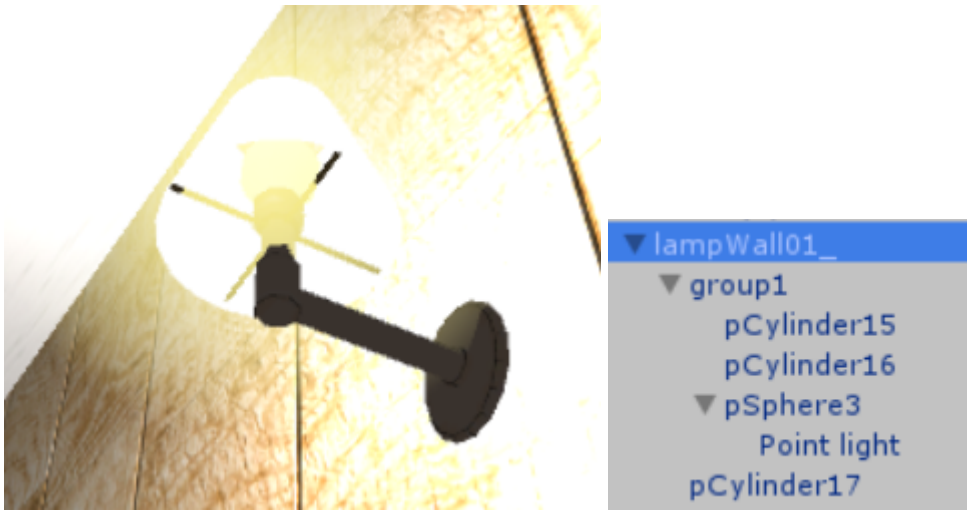
Entonces, el usuario presiona la tecla ‘e’, la puerta se abre y carga el siguiente nivel.

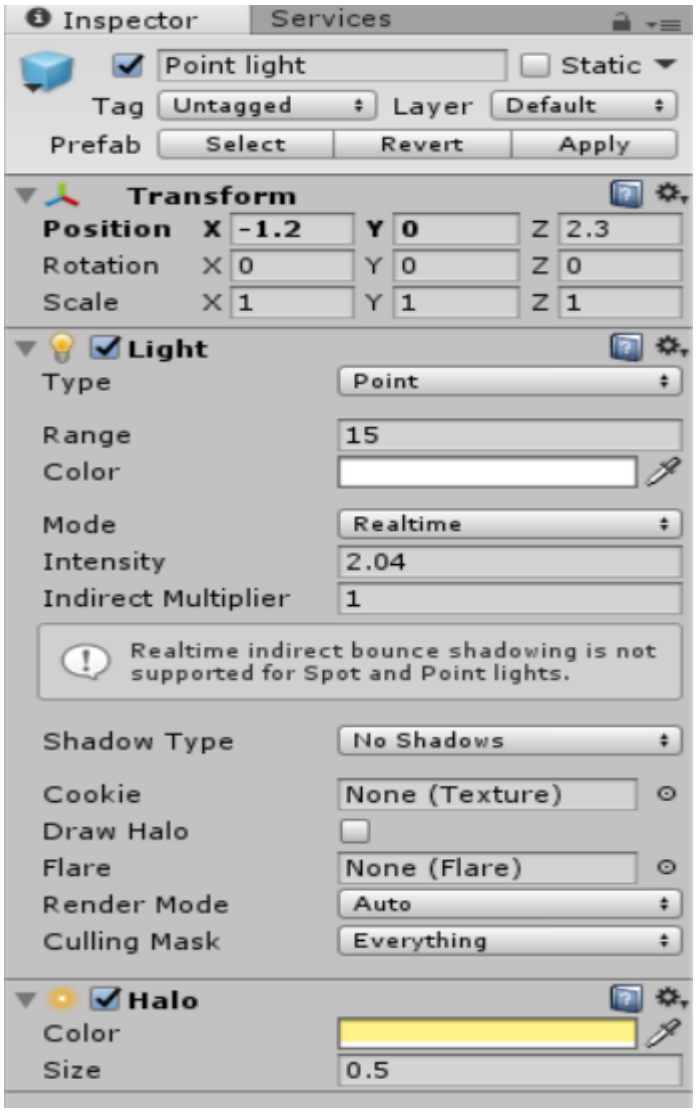
Centrándonos en el funcionamiento general del nivel, hay diferentes elementos que interactúan con el jugador:

- El muro: tiene una estructura similar a la pared del diseño inicial que se ha explicado en el anterior punto. Tiene un trigger que cuando el jugador toca el muro, se reinicia la escena
- La puerta: para explicar el funcionamiento de la puerta hay que diferenciar diferentes aspectos. Hay un *`Empty GameObject`* que hace de controlador de la puerta. Dicho objeto tiene un *`Box Collider`* que simplemente es una pequeña zona invisible delante de la puerta que actúa de trigger. Es decir, cuando el

jugador entra a esta zona sale el cartel informativo para abrir la puerta, y cuando éste sale, el cartel desaparece. Mientras el jugador esté dentro de la zona, podrá abrir la puerta presionando la tecla correcta. Luego está la propia puerta, que consiste en un cubo al que le hemos puesto la imágen de una puerta.

En cuanto a la iluminación, decidimos crear una lámpara y añadirle luz:





Es un objeto con dos componentes:

- Light: la luz que tendrá el objeto. Es del tipo *`Point`* (un punto que emite luz en una zona determinada), con una intensidad y un rango de iluminación determinado.
- Halo: el color del resplandor de la luz.

3.3 Niveles finales

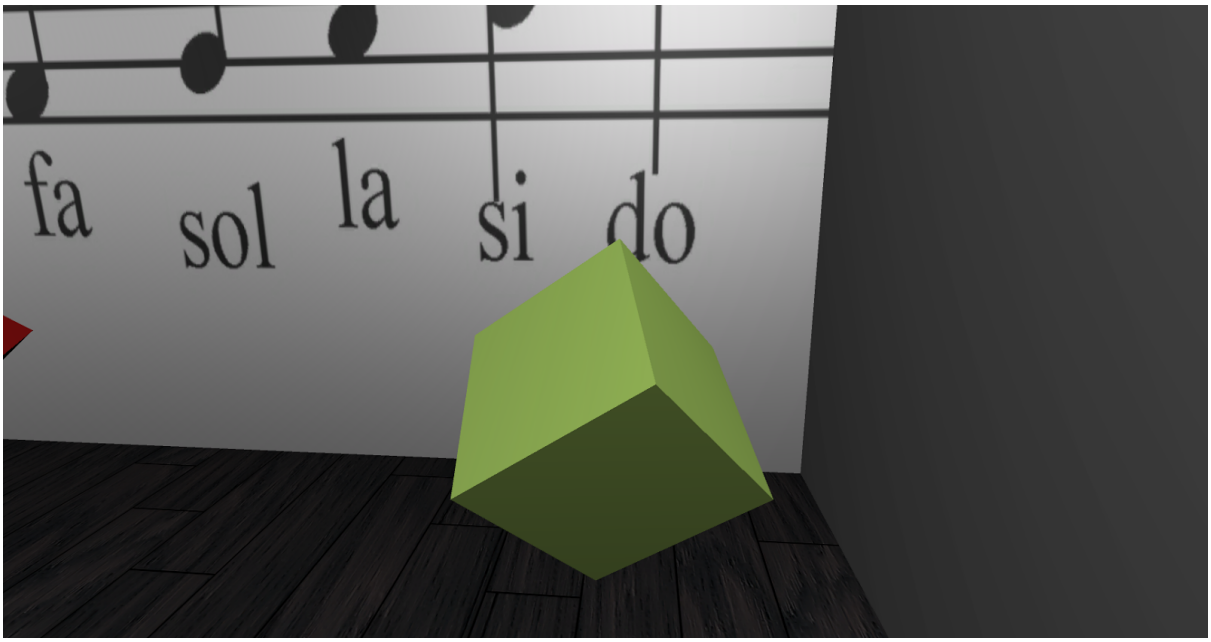
Nuestros últimos niveles implementados en el juego ofrecen la mayor complejidad y creatividad de todo el proyecto. Aún siendo más ambiciosos en cuanto a la programación de éstos, resultaron ser de los niveles que más rápido hemos conseguido desarrollar, denotando en ello nuestra familiarización con el motor gráfico y nuestra fluidez al manejar C# y los scripts asociados a Unity.

Debemos destacar también que estos niveles son bastante más simples en cuanto a diseño gráfico que los demás, un diseño que nos ha llevado muchísimas horas para no tener ni un solo componente de programación. Este hecho también agudizó la presteza con la que terminamos dichos niveles.

Nuestros niveles finales (que irónicamente son los dos primeros que el jugador deberá superar para continuar hasta el final) son un puzzle relacionado con una ordenación de colores y otro empleando elementos musicales simples. No obstante, empleamos distintos elementos más avanzados que nos ofrece Unity para dar un acabado y una eficiencia mucho más elevados que los realizados en un primer momento.

En el nivel de colores, podemos simplificar y reutilizar un simple script para 6 objetos distintos y emplear un sistema de Eventos que nos facilita Unity para llamar de manera casi instantánea las diferentes funciones de comprobación y validación de cada uno de los colores sin realizar declaraciones redundantes ni ejecuciones de código innecesarias. Es por ello que podríamos decir que el nivel de los colores es, por ahora, el nivel más eficiente de todo el proyecto.

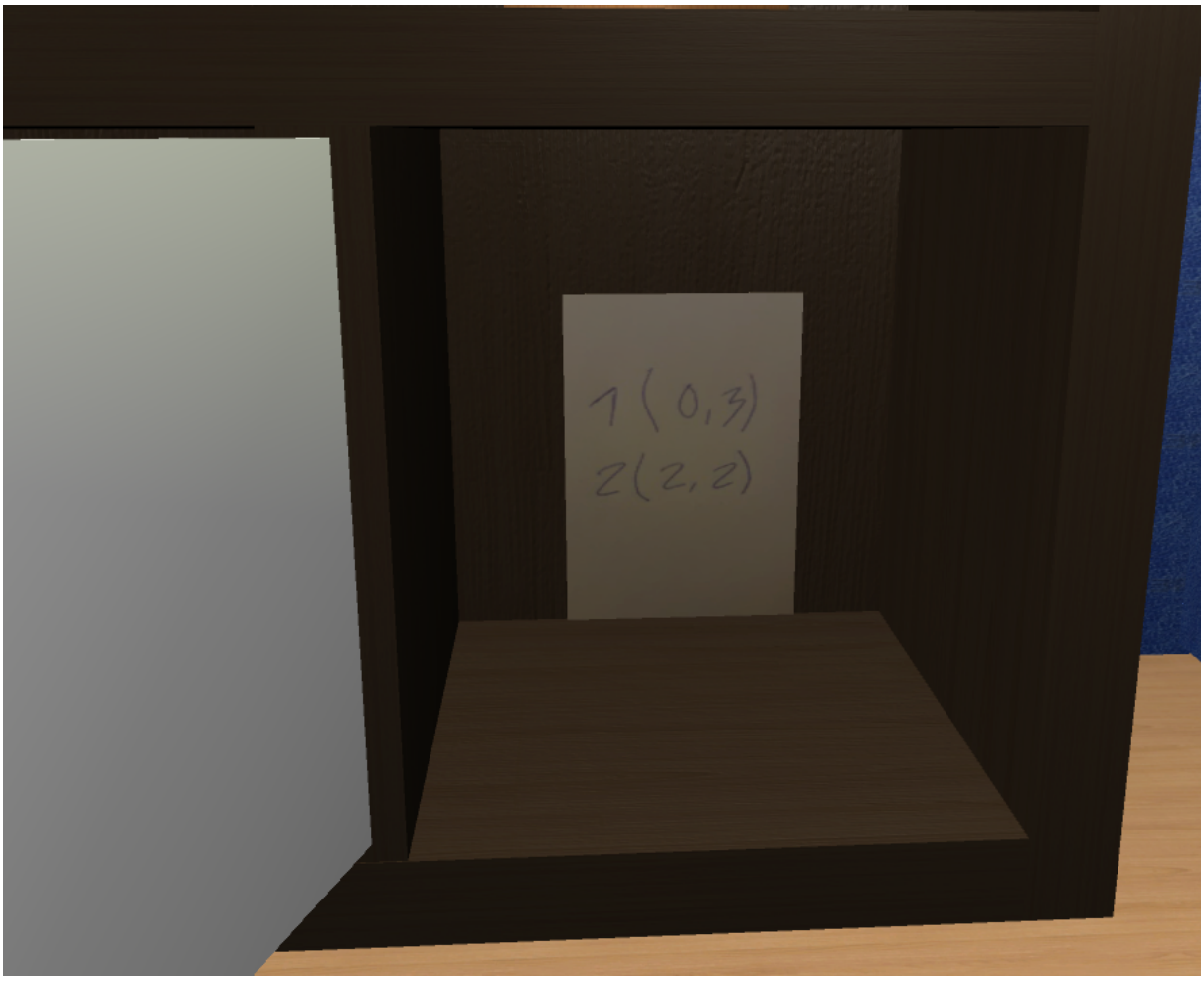
En el nivel musical, se utilizan dos scripts: uno común para todos los cubos y otro que controla si se han tocado las notas musicales en el orden correcto. Estos cubos tienen dos procesos: uno que hace el movimiento ascendente y descendente y otro para la rotación. Luego, tienen un trigger que, cuando el jugador está en él y pulsa la tecla ‘e’, si es el cubo que contiene la nota musical correcta, cambia de color. En caso de ser la nota errónea, se reinician los cubos al color original. Una vez las notas hayan sido tocadas en el orden correcto, la puerta queda desbloqueada.



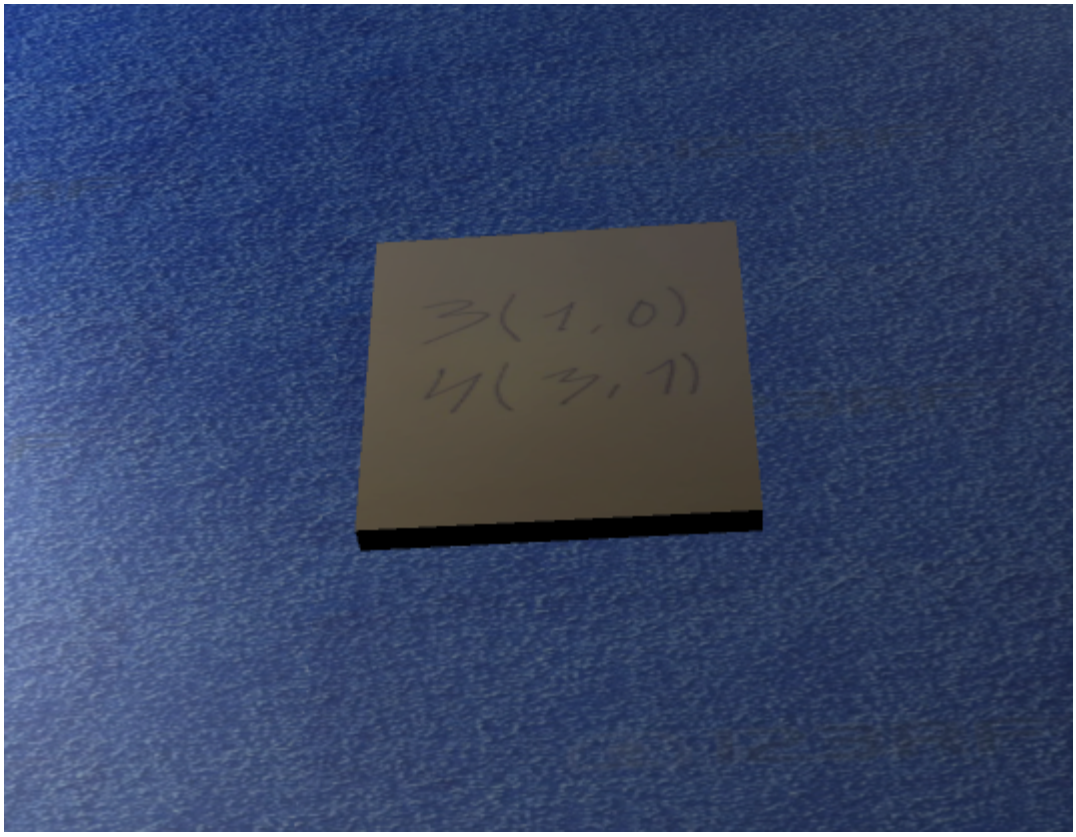
En el último nivel, el jugador se encontrará una habitación como en los anteriores niveles. Tendrá que abrir la puerta que está bloqueada mediante un código el cual deberá encontrar los números correctos que están escondidos en la habitación.

En la estantería hay dos cajones, uno en el que no hay nada y el otro está cerrado con llave. Para abrirlo, el usuario debe de encontrar su llave.

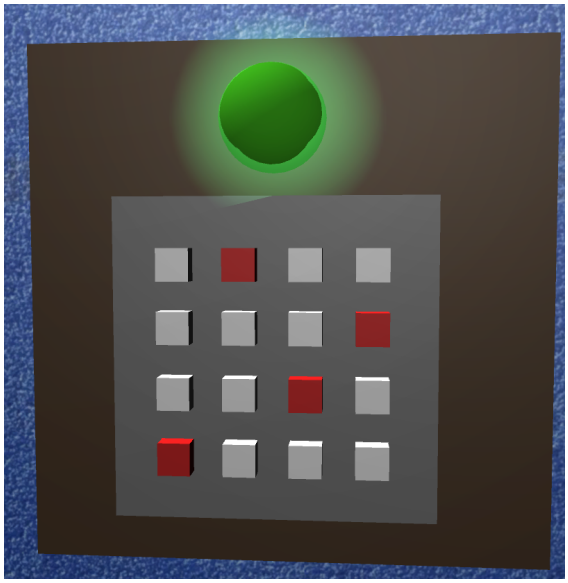
En la habitación, además, hay una cama y una mesita de noche donde se encuentra la llave del cajón de la estantería. Entonces, cuando el jugador abra ese cajón, verá la primera combinación para salir de la habitación.



Por último, el jugador debe de buscar un cuadro y descolgarlo presionando la tecla ‘e’, ya que detrás de dicho cuadro se encuentran las dos últimas coordenadas para desbloquear la puerta.


3(1,0)
4(3,1)

Finalmente, se introduce la contraseña correcta en el panel y la puerta ya podrá abrirse pero, a diferencia de los anteriores niveles, la escena no cambia ya que, una vez se abre la puerta, el jugador verá que está encerrado en un gran laberinto en el que deberá esquivar diferentes trampas para poder escapar.

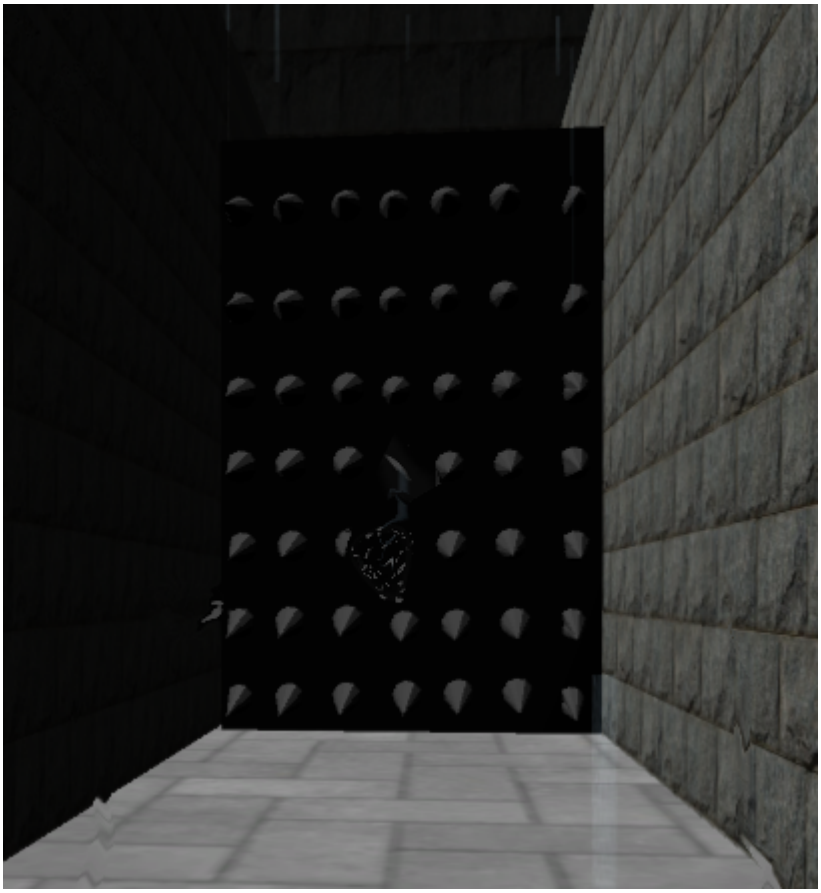


Una vez el jugador haya salido de la habitación deberá llegar al final de un laberinto, cosa que en un principio no será fácil.

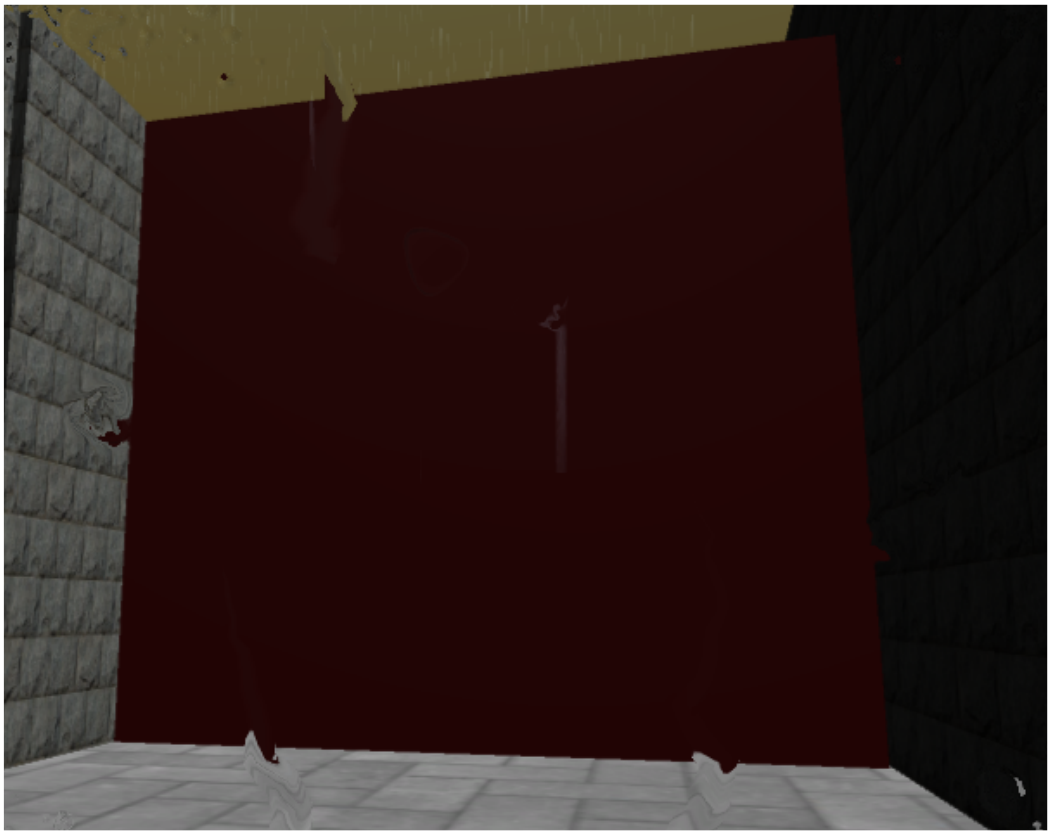
Al diseñar y plantear este nivel hemos decidido no perpetrar un laberinto típico de libro, con múltiples caminos y ya está, pues consideramos que sería algo bastante aburrido y tedioso. En su defecto, hemos llenado distintos pasillos con trampas que acabarán con la vida del jugador, por lo que escoger sabiamente será la clave para superar el reto del laberinto.

Aún así, consideramos que la propia muerte del personaje es suficiente penalización, por lo que hemos implementado una serie de *checkpoints* para que, en caso de escoger erróneamente, el jugador pueda volver a un estado anterior sin tener que empezar el nivel o el juego desde el principio.

Por una parte, si el jugador entra por un pasillo que no toca, se verá atrapado por una trampa que eventualmente acabará con él.

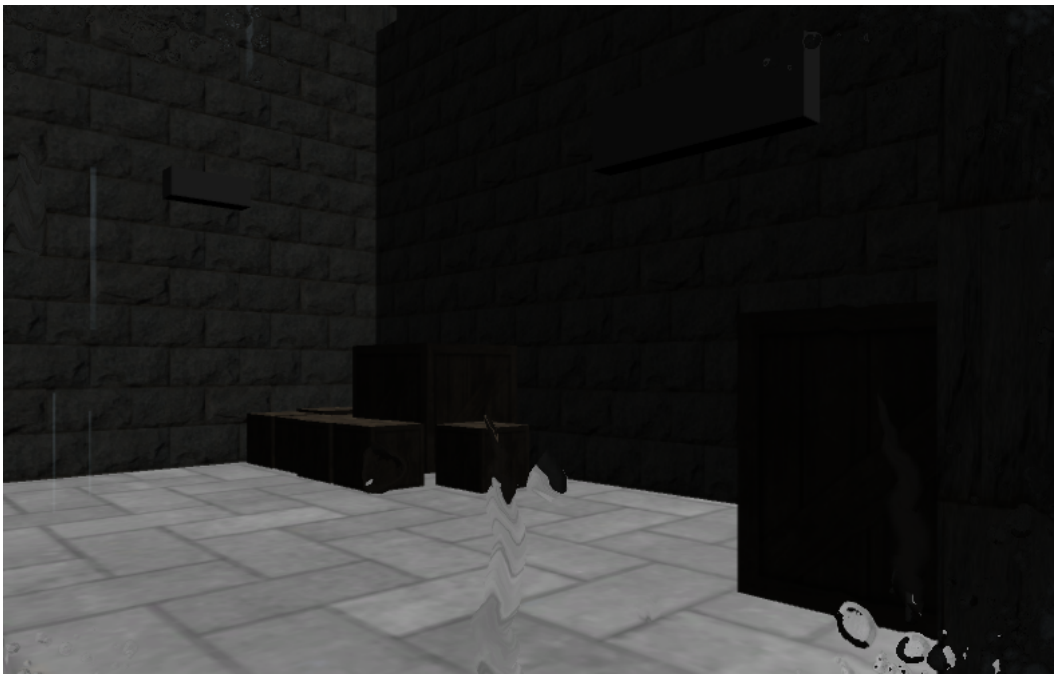


Si escoge otro pasillo, eventualmente llegará a una pared roja que le impedirá el paso. Es recomendable la investigación en este tipo de casos para avanzar y encontrar elementos que permitan superar las trampas. Para poder superar este muro rojo, el jugador encontrará una palanca que, accionándola, abrirá el camino al resto del laberinto.

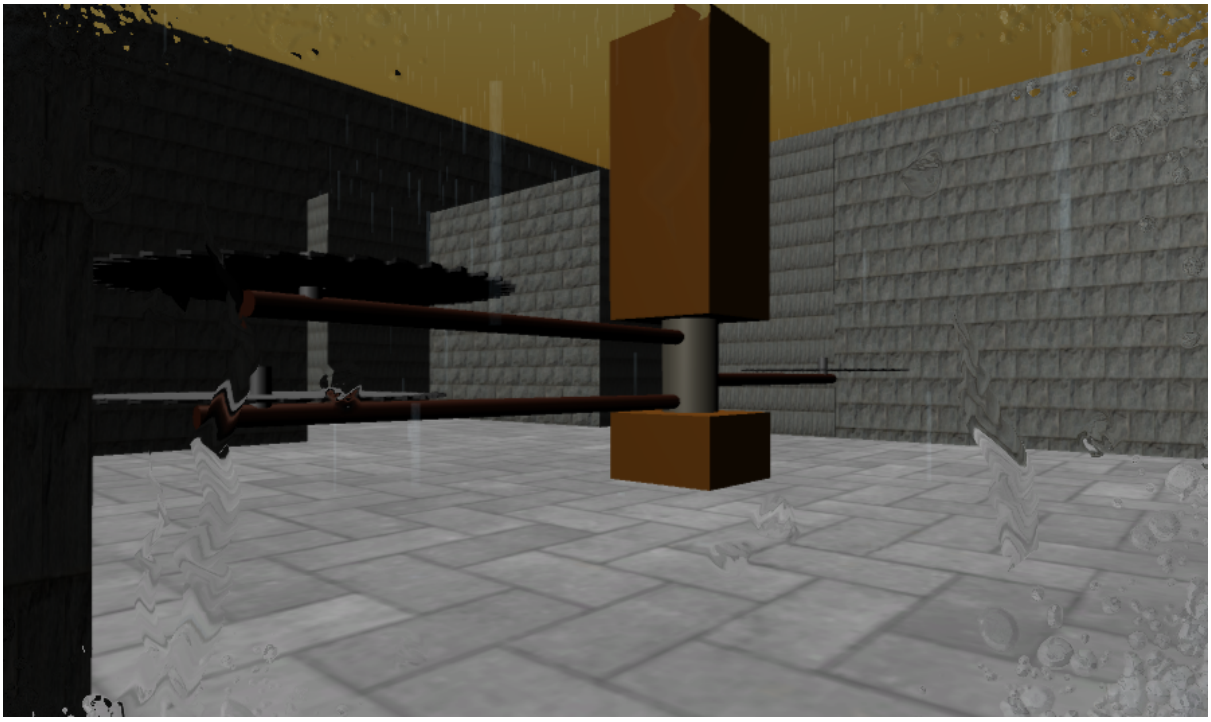


Si se sigue avanzando, el jugador encontrará elementos nocivos que se mueven de forma periódica. Saltar antes de ser alcanzado es recomendable para superar esta trampa.

Para los más hábiles también se ha preparado este pequeño *easter egg* y *skip* que permite superar la totalidad del laberinto si se salta correctamente.



Si el jugador continúa su camino, encontrará eventualmente una serie de sierras giratorias (a distintas velocidades cada una) y un cañón que dispara periódicamente que, si el jugador consigue esquivar adecuadamente, permitirán un transcurso fluido hacia el final del laberinto, pues ya no hay más trampas.



.

3.4 Scripts esenciales

SpearMovement

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SpearMovement : MonoBehaviour
{
    IEnumerator Start()
    {
        Vector3 pointA = transform.position;
        Vector3 pointB = new Vector3(transform.position.x, transform.position.y, transform.position.z + 5.5f);
        while (true)
        {
            yield return StartCoroutine(MoveObject(transform, pointA, pointB, 1.5f));
            yield return StartCoroutine(MoveObject(transform, pointB, pointA, 1.5f));
        }
    }

    IEnumerator MoveObject(Transform thisTransform, Vector3 startPos, Vector3 endPos, float time)
    {
        float i = 0.0f;
        float rate = 1.0f / time;
        while(i < 1.0f)
        {
            i += Time.deltaTime * rate;
            thisTransform.position = Vector3.Lerp(startPos, endPos, i);
            yield return null;
        }
    }
}
```

En este caso, este código lo que hace es mover un par de lanzas horizontalmente y de manera permanente, con una periodicidad sinusoidal típica. Para conseguir que estén siempre en movimiento lo que hacemos es usar dos threads, que en Unity se llaman Coroutines y que sus métodos vienen definidos por el tipo `IEnumerator`. Para no cargar de problemas el juego evitamos usar funciones en `Update`, ya que éste se ejecuta por cada `frame`, por lo que en este caso empleamos dos threads, uno después del otro, que hace que la transición sea suave y sin hacer múltiples llamadas por `frame`, dando errores en el movimiento y `renderizado`. Lo único que hará el código es definir una posición final, una inicial y un tiempo para darle un ratio a la interpolación lineal, que es el método que usaremos para alterar la transformada de las Lanzas. El resultado que obtenemos con este método es el de una eficiencia muy superior a la de

usar un `Update` y además nos permite ejecutar otras funciones en paralelo al `thread`.

`Event System: RedWallManager`

```
public class EventManager : MonoBehaviour {

    public GameObject go;
    public static bool activated;
    private bool enter = false;
    private LeverController lc;
    public delegate void ClickAction();
    public static event ClickAction OnClicked;
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RedWallManager : MonoBehaviour {

    public GameObject redwall;


    private void OnEnable()
    {
        EventManager.OnClicked += OpenPath;
    }

    private void OnDisable()
    {
        EventManager.OnClicked -= OpenPath;
    }

    void OpenPath()
    {
        redwall.SetActive(false);
    }
}
```

El sistema de eventos nos proporciona una manera efectiva de activar y lanzar distintos métodos de objetos de una escena sin instanciar ningún objeto adicional ni hacer malabarismos con el código. Tendremos una clase Manager que normalmente se asocia a la `clase/GameObject` que realizará la acción desencadenante y que ésta lo único que hará es definir el evento. Una vez hecho esto sólo tenemos que llamar a esta nueva función `(ClickAction OnClicked)` y ésta ejecutará todos los métodos asociados con el mismo nombre de las diferentes clases adheridas al evento, en este caso lo único que debemos hacer es añadir el nombre del método que queremos

ejecutar cuando se dé la condición establecida, en la imagen se puede ver en `OnEnable`, en este caso como sólo tenemos un método para este evento sólo llamará al asociado `OpenPath`. Cuando activemos y desactivemos objetos debemos añadirlos y eliminarlos de la pila respectivamente para evitar *memory leaks* y *crashes* del juego y de Unity.

Como vemos en la imagen, cuando en una sección del código de `EventManager` llamamos a `OnClicked` y éste ejecutará todos aquellos métodos que tuviere. Cabe destacar la eficiencia que se obtiene empleando esta 'técnica'.

Fader

```
public static class Initiate {

    //the scene which is going to load, the color of the fader and how long will the transition be (damp)

    public static void Fade(string scene, Color col, float damp)
    {
        //we create a new empty GameObject and we change its behaviour to Fader
        GameObject init = new GameObject();
        init.name = "Fader";
        init.AddComponent<Fader>();
        //new object Fader with the init component
        Fader fader = init.GetComponent<Fader>();
        //and we pass the parameters to its attributes
        fader.fadeDamp = damp;
        fader.fadeScene = scene;
        fader.fadeColor = col;
        fader.start = true;
    }
}
```

```
public class Fader : MonoBehaviour {

    public bool start;
    public float fadeDamp = 0.06f;
    public string fadeScene;
    public float alpha = 0.0f;
    public Color fadeColor;
    public bool isFadeIn = false;
```

```
void OnGUI () {
    if (start) {
        //Components r,g,b define a color in RGB color space and alpha component defines transparency (1 is completel
        GUI.color = new Color(GUI.color.r, GUI.color.g, GUI.color.b, alpha);

        //we use this to modify the texture
        Texture2D myTex;
        myTex = new Texture2D(1, 1);
        myTex.SetPixel(0, 0, fadeColor);
        myTex.Apply();

        //and we draw this texture in the whole scene
        GUI.DrawTexture(new Rect(0, 0, Screen.width, Screen.height), myTex);
        if (isFadeIn)
        {
            alpha = Mathf.Lerp(alpha, -0.1f, fadeDamp * Time.deltaTime);
        }

        else
        {
            alpha = Mathf.Lerp(alpha, 1.1f, fadeDamp * Time.deltaTime);
        }

        if (alpha >= 1 && !isFadeIn)
        {
            SceneManager.LoadScene(fadeScene);
            DontDestroyOnLoad(gameObject);
        }

        else if (alpha <= 0 && isFadeIn)
        {
            Destroy(gameObject);
        }
    }
```

```
void OnLevelWasLoaded(int level) {
    isFadeIn = true;

}
```

`Initiate` es una clase estática que trata la transición entre escenas. Puede ser llamada desde cualquier script que trate el cambio de niveles. Para ello, simplemente debe de llamarse en el código con un *`Initiate.Fader(escena, color de la transición, tiempo de la transición)`*
Entonces, se crea un objeto vacío en el que se le añade un componente del tipo *`Fade`* que se encarga de realizar dicha transición.

Luego, en el método `OnGUI()` de `Fader`, se crea una textura con las características definidas y se añade en toda la pantalla. Entonces, cuando la pantalla se ha vuelto negra por completo, se carga el siguiente nivel y se inicia la transición inversa. Una vez finalizado el evento, el objeto creado en `Initiate` se elimina.

3.5 Bug-Fixing

Durante el proyecto hemos sido víctimas de una serie de bugs que azotaban funciones básicas de nuestro código. Entre ella encontrábamos elementos que quedaban flotando en el aire, el personaje se quedaba atascado entre paredes, o ciertos elementos del juego se activaban cuando no tocaban.

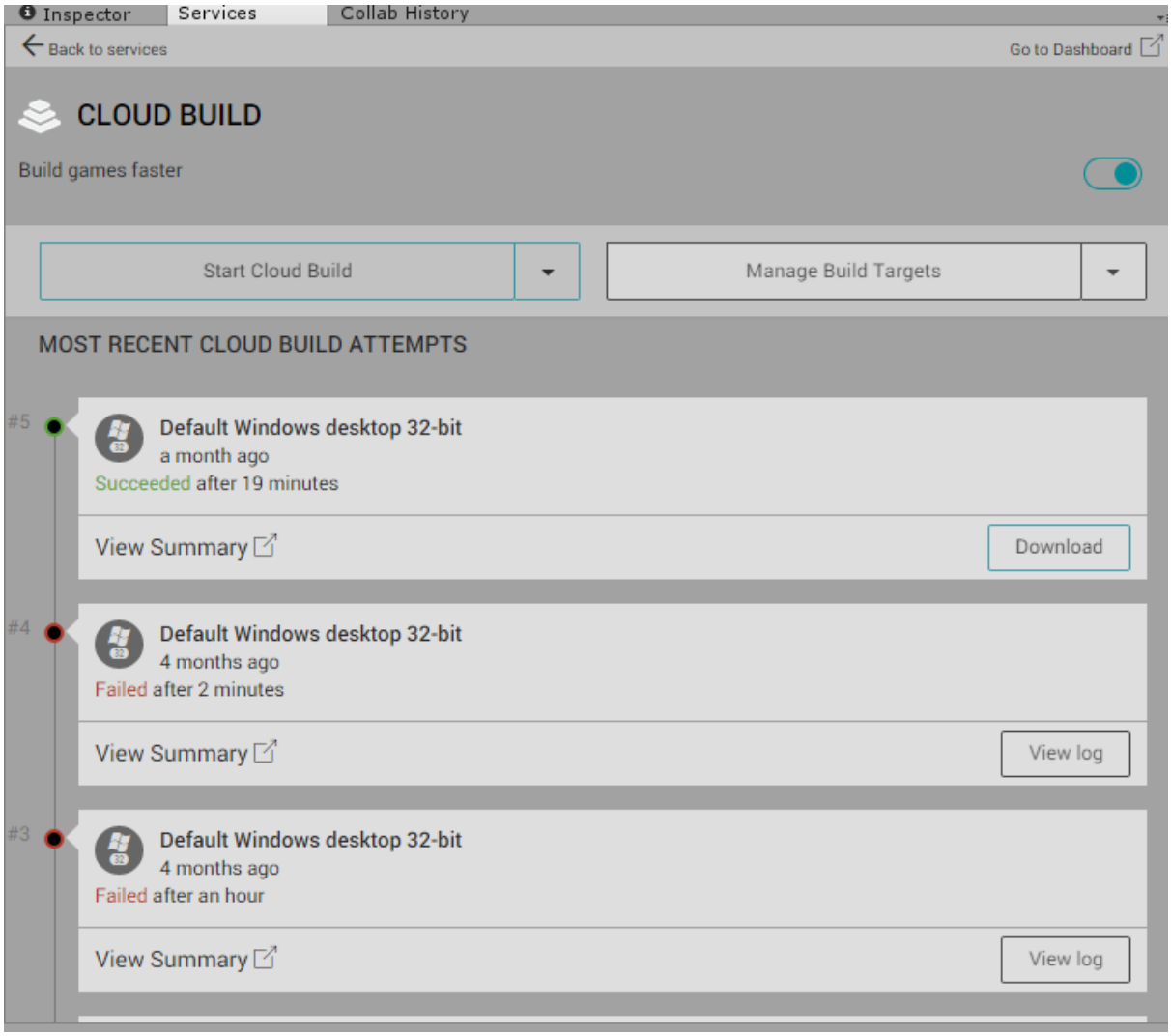
Para solucionar este tipo de problemas realizamos una serie de pasos preventivos para detectar la raíz de estos fallos.
En primer lugar, detectamos el origen del fallo de manera visual, en segundo lugar buscamos las secciones críticas de código y le añadimos `Debug.log("")` y un mensaje para que en tiempo de ejecución nos aparezca en la consola de Unity.
Una vez hecho todo esto, ejecutamos el código y acotamos los elementos que pueden ser conflictivos. Si todo esto no nos funciona, terminamos de comprobar que los *`Components`* de Unity están asignados correctamente o son utilizados por el código como debería.

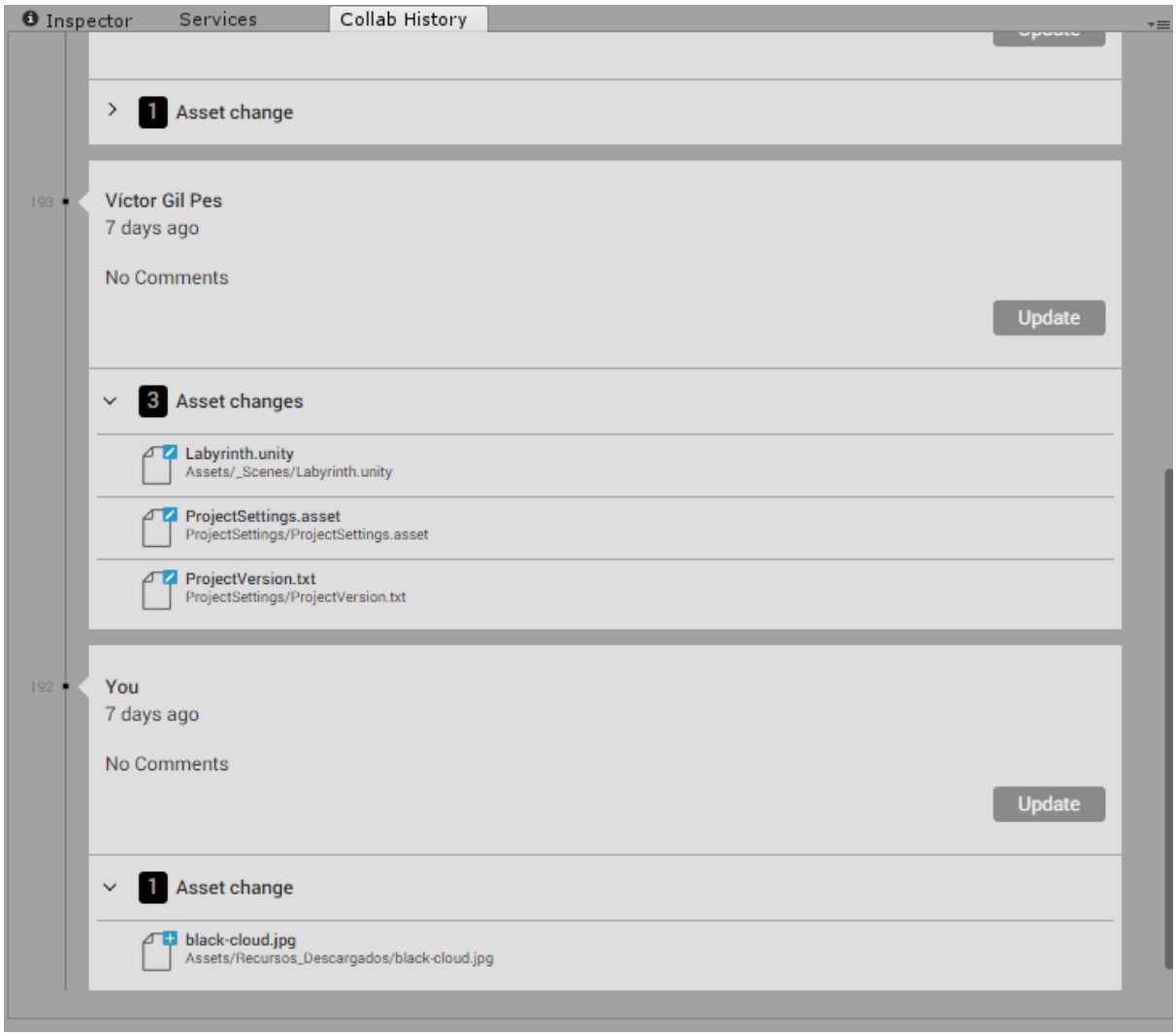
En la mayoría de casos, los bugs que nos hemos encontrado han sido una combinación de fallos en los scripts con una gestión incorrecta de los *`Components`* de los objetos implicados.

4 Gestión del proyecto

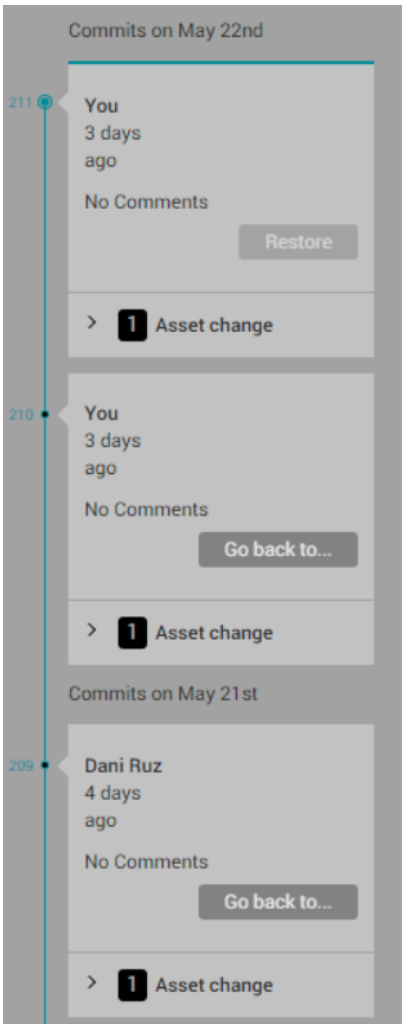
4.1 Unity Cloud Services

Para facilitar nuestro trabajo y poder importar el proyecto desde más de un ordenador, el propio Unity nos ofrece de manera gratuita el sistema de UCS, lo que nos permite subir y bajar los cambios que se hacen en el proyecto a tiempo real, facilitando así el intercambio de scripts y trabajar al momento con el proyecto actualizado. Nos permite también revertir cambios y subidas por si hubiera algún problema con las versiones más nuevas.




Inspector
Services
Collab History
1 Asset change
193
Víctor Gil Pes
7 days ago
No Comments
Update
3 Asset changes
Labyrinth.unity
Assets/_Scenes/Labyrinth.unity
ProjectSettings.asset
ProjectSettings/ProjectSettings.asset
ProjectVersion.txt
ProjectSettings/ProjectVersion.txt
192
You
7 days ago
No Comments
Update
1 Asset change
black-cloud.jpg
Assets/Recursos_Descargados/black-cloud.jpg

4.2 Control de versiones

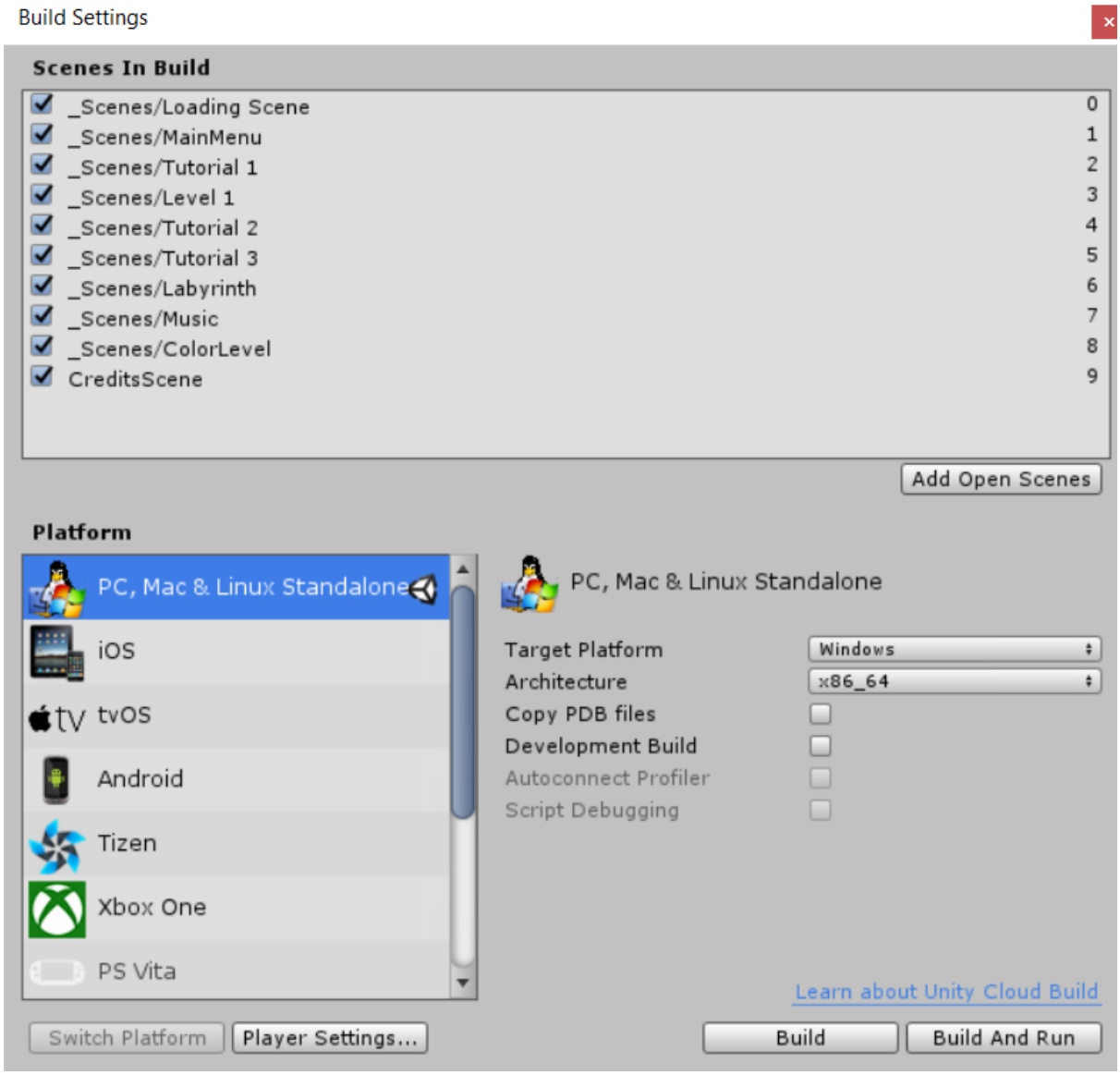


Unity proporciona un servidor para el control de versiones de un proyecto llamado Unity Cloud Services. Es una herramienta muy útil para poder trabajar con diferentes personas en un proyecto que permite tener siempre la última versión del trabajo. Es parecido a Git.

En caso de que dos o más personas hayan desarrollado una misma parte de diferente manera, Unity Cloud permite solucionar estos conflictos mirando las diferencias entre ambas versiones y puedes decidir si la versión que va a prevalecer sea una u otra.

4.3 Configuración del Build/Run

Para ejecutar el juego, puedes hacerlo directamente a través del editor de Unity o haciendo un .exe . Para ello, Unity exporta el proyecto y lo hace. Además, desde la propia web de Unity puedes acceder a la compilación del proyecto.

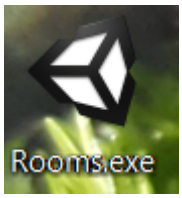


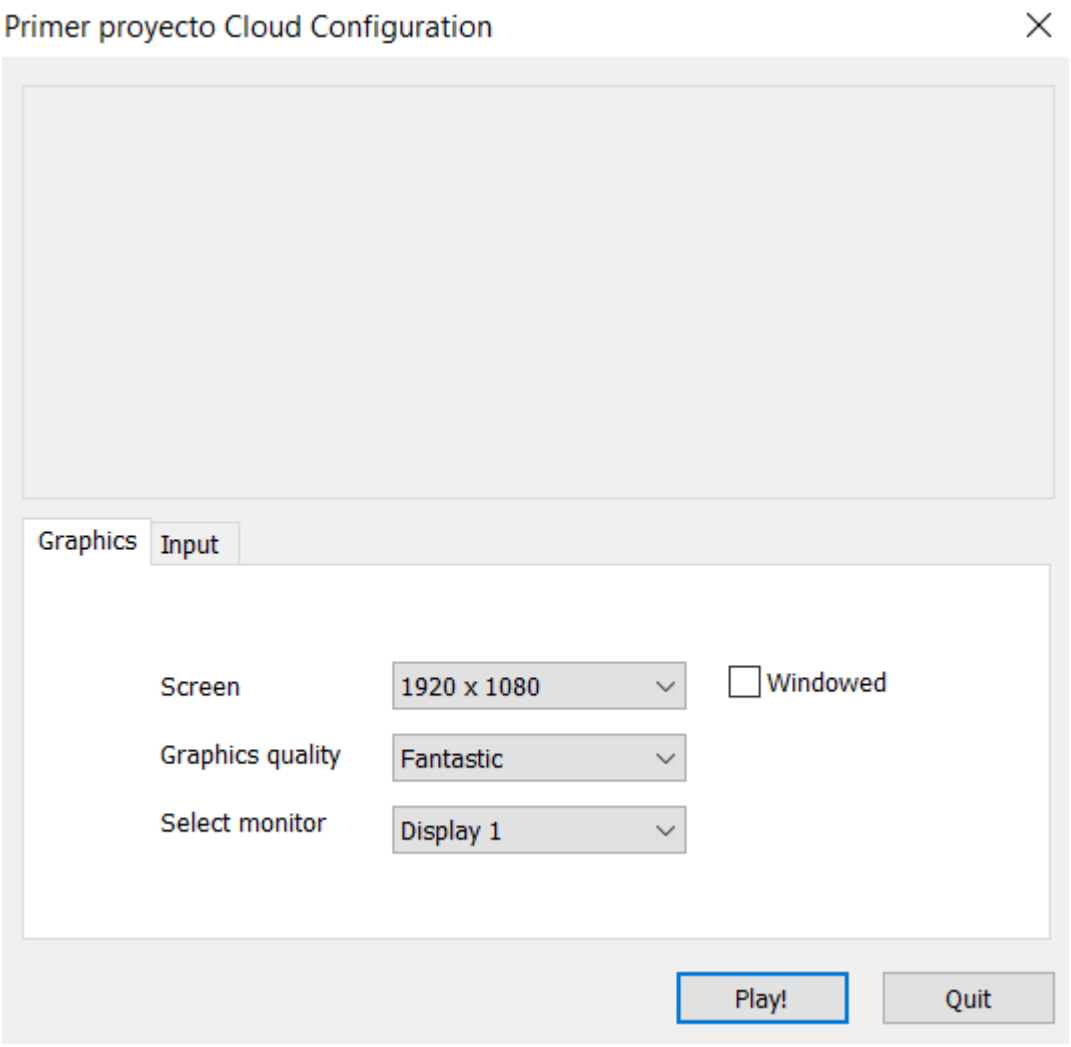
En esta pantalla es donde dices a Unity que escenas va a compilar. Las añades e indicas el sistema operativo que deseas hacer el exe y luego le das a Build And Run. Una vez hecho esto, ya tendrás el exe del proyecto con una carpeta con los archivos necesarios.

Unity también permite hacer el exe en otras plataformas como Android, iOS, etc.

5. Ejecución del proyecto

Una vez hemos generado el ejecutable con el *Build* de Unity lo único que deberemos hacer es colocar tanto este ejecutable como la carpeta *XXX_Data* en la misma localización para permitir la ejecución del juego.





6. Conclusiones

Han sido tres meses de puro trabajo y esfuerzo. Hemos tenido que dedicarle muchas horas en aprender tanto una herramienta como es Unity y un lenguaje que no se ha dado en horas lectivas, en este caso C#. En nuestra humilde opinión creemos que tres (apenas) meses es una cantidad de tiempo minúscula para poder presentar un trabajo en condiciones y de buena calidad, forzando que debamos emplear muchas horas en dar un resultado final similar al de un proyecto con unas aspiraciones temporales mayores.

Empezamos un mes antes a mirar tutoriales y guías para aprender C# y Unity para poder comenzar nuestro proyecto de la mejor manera posible, dedicando bastantes horas a este aprendizaje sumado a las horas lectivas.

Debemos reconocer que tal vez en un principio nos pusimos unas metas que materialmente no eran asumibles para la cantidad de tiempo proporcionada y por ello hemos debido cambiar, adaptar y reducir contenidos que en un primer momento nos hubiera gustado haber incluido en el resultado final en el momento de la entrega. Tal vez hayamos pecado de ambiciosos, pero lo que sí podemos concluir es que, por una parte, se ha disfrutado mucho con todo este largo proceso de aprendizaje, los resultados obtenidos han sido muy satisfactorios y nos enorgullecemos del proyecto presentado.

Nos hemos visto obligados en muchas ocasiones a cambiar la planificación inicial o incluso la planificación que planteamos a medida que avanzaba el proyecto por las cantidad abrumadora de contratiempos, cambios y demás elementos que nos obligaban a replantearnos la dirección general del proyecto y a ejercitar nuestro ingenio para resolver y sobrepasar ciertos obstáculos que aparecían en su momento. Todo esto lo achacamos tanto a nuestra inexperiencia con estas dos tecnologías como a la adaptación a una nueva forma de trabajo como es la que nos fuerza Unity.

Nos hubiera gustado añadir al final ciertos cambios de *QoL* en cuanto a la experiencia final del usuario como también darle un final más *aventuresco* al completar el último nivel, así como haber realizado alguna pantalla con el Terrain Generator de Unity para realizar algún paisaje. Por último también hacer un lavado de cara de los tutoriales para hacerlos más accesibles e intuitivos para el usuario, pero por tiempo no hemos podido incluir todas estas mejoras e innovaciones en el proyecto.

7. Webgrafía

https://3dwarehouse.sketchup.com/

En esta página hemos descargado diferentes diseños que hemos utilizado durante el juego.

https://www.youtube.com/watch?v=Wlkzzk0cDJE
https://www.youtube.com/watch?v=iui5whcpruI&t=15s
https://www.youtube.com/watch?v=8Tknbp0Jx_Q
https://www.youtube.com/watch?v=dJB07ZSiW7k
https://www.youtube.com/watch?v=QT-6u6NLaus
http://answers.unity3d.com/questions/18219/instantiate-prefab-with-c-script.html
https://docs.unity3d.com/Manual/Lighting.html
https://www.youtube.com/watch?v=Ax3hWJtmkPY
https://www.youtube.com/watch?v=m0wgFgUv1zs
https://docs.unity3d.com/Manual/class-Light.html
http://answers.unity3d.com/questions/13944/change-gui-font-size-and-color.html
https://www.youtube.com/watch?v=0Tpm0AyUQQU
https://www.youtube.com/watch?v=yDkZ5QIkfSc
https://forum.unity3d.com/threads/calling-a-function-from-another-script.396635/
https://forum.unity3d.com/threads/how-to-change-text-color-on-gui-label.51791/
http://answers.unity3d.com/questions/850220/how-can-i-get-a-ui-canvas-to-hideappear-on-esc-but.html
https://docs.unity3d.com/ScriptReference/EventSystems.EventSystem.html
https://www.youtube.com/watch?v=xJQXoG3caGc&t=493s
https://docs.unity3d.com/ScriptReference/GUISkin-button.html
https://docs.unity3d.com/ScriptReference/Collider.OnCollisionEnter.html
https://docs.unity3d.com/ScriptReference/Collision-collider.html
https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnCollisionStay.html
http://answers.unity3d.com/questions/671308/access-objects-enum-type.html
http://answers.unity3d.com/questions/138464/how-to-make-a-line-break-in-a-gui-label.html

8. Glosario

- GameObject: es la clase básica de todas las entidades de Unity
- Component: son las clases que forman un GameObject
- Prefab: es un GameObject completo, modulado y con componentes que se almacena de forma permanente para usarlo siempre que quieras.
- Script: es un pequeño programa con código simple e interpretado con Unity que se encargan de controlar todas las acciones del juego.
- BoxCollider: es un componente básico de un GameObject y consiste en una caja invisible alrededor de un objeto y trata sus colisiones. A partir de estas colisiones, se pueden tratar en algún script para realizar diferentes funciones.
- Trigger: es un tipo de formato del BoxCollider. Cuando se indica que un BoxCollider es del tipo Trigger, esta caja invisible se puede atravesar, pero cuando el jugador, objeto o algún elemento entra en el trigger, sale, o se mantiene, se pueden hacer diversas acciones para controlar el desarrollo del juego.

9. Anexo

Plan de trabajo y estado del arte

1.- Introducción

En este plan de trabajo se explica la organización, objetivos y requisitos para elaborar nuestro proyecto de final de grado.

2.Objetivos

El objetivo principal de nuestro proyecto es el de adentrarnos en el complejo mundo del diseño y la programación de los videojuegos, comprendiendo todos y cada uno de los pasos y entendiendo su estructura y funcionamiento básicos.

Sin embargo, en general, otros de nuestros objetivos son:

- Aprender un nuevo lenguaje de programación (C#).
- Aplicar todos los conocimientos adquiridos sobre procesos y programación.
- Aprender a usar el motor gráfico Unity.
- Mejorar la eficiencia y funcionalidad de nuestro código.

3.- Apartados

En este capítulo detallaremos los apartados y subapartados que contendrá nuestro proyecto, en función de los objetivos marcados en el anterior apartado:

1.- Índice

2.- Introducción

3.- C#

- *3.1.- Introducción a C#*
- *3.2.- Algunos ejemplos sobre este lenguaje*

4.- Unity

- *4.1.- Introducción a Unity*
- *4.2.- Conocimiento de Unity*
- *4.3.- Herramientas útiles*

5.- Práctica

- *5.1.- Integrar conocimientos obtenidos a C# y Unity*

6.- Conclusiones

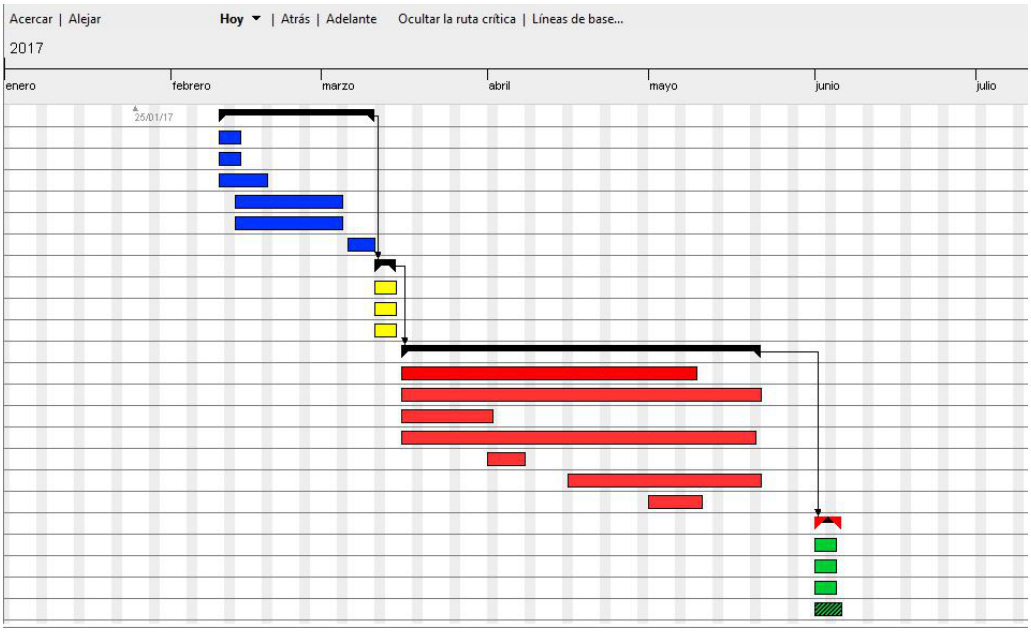
7.- Glosario

8.- Bibliografía

4.- Temporización e hitos

4.1.- Temporización (Diagrama de Gantt)

Fase Inicial	10/02/17	10/03/17
Elaboración del informe p...	10/02/17	13/02/17
Decisión del tipo de juego	10/02/17	13/02/17
Preparación de requisitos	10/02/17	18/02/17
Aprendizaje C#	13/02/17	4/03/17
Aprendizaje motor Unity	13/02/17	4/03/17
Elaboración 'Entrega 1' (PI...	6/03/17	10/03/17
Segunda fase	11/03/17	14/03/17
Elaboración diagrama de ...	11/03/17	14/03/17
Elaboración diagrama cas...	11/03/17	14/03/17
Desarrollo primeros nivele...	11/03/17	14/03/17
Tercera fase	16/03/17	21/05/17
Desarrollo juego	16/03/17	9/05/17
Aplicación de recursos grá...	16/03/17	21/05/17
Programación de Scripts b...	16/03/17	1/04/17
Diseño y programación de...	16/03/17	20/05/17
Elaboración 'Entrega 2'	1/04/17	7/04/17
Pruebas de errores	16/04/17	21/05/17
Elaboración 'Entrega 3'	1/05/17	10/05/17
Fase final	1/06/17	5/06/17
Bugfixing	1/06/17	4/06/17
Beta testing	1/06/17	4/06/17
Polishing	1/06/17	4/06/17
Preparar exposición	1/06/17	5/06/17
Elaboración 'Memoria final'	1/06/17	5/06/17



4.2.- Hitos

Para el buen desarrollo de esta materia, hay una serie de entregas parciales definidos previamente:

1.- Entrega del Plan de Trabajo	10/03/2017
2.- Toma de requisitos	08/04/2017
3.- Implementación	12/05/2017
4.- Entrega de la memoria	03/06/2017
5.- Entrega del código de implementación	06/06/2017

(*)*Estas fechas son estimadas*

5.- Material necesario

5.1 Conocimientos previos

Es necesario e importante tener conocimientos previos de programación orientada a objetos y, en menor medida, algo de física.

5.2 Maquinaria necesaria

Necesitaremos un ordenador capaz de poder mover Unity con cierta fluidez.

5.3 Programas necesarios

-Unity

-Visual Studio

-Frameworks .NET asociados a VS y Unity

6.- Requisitos y usos

6.1.- Requisitos

Los requisitos principales para poder ejecutar y jugar el juego son muy básicos. Necesitaremos una máquina con Windows (ya sea 32 o 64 bits) y al menos 2 GB de RAM para asegurar algo de fluidez, aunque recomendamos de 4 a 6 para garantizar la mayor fluidez posible en cuanto a FPS.

Es posible también adaptar parte del programa para poder ser importado en MacOS y Linux, pero no por ahora.

También es necesario que el usuario cuente con un teclado y ratón.

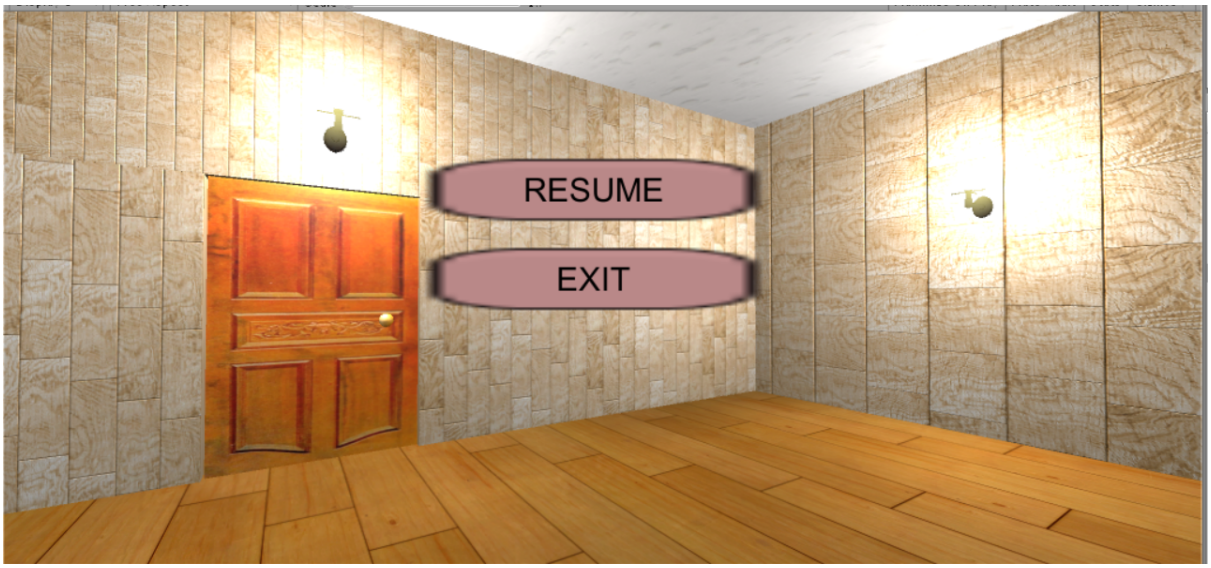
6.2.- Usos

Dado que tratar de describir y mostrar todos los casos de uso de todos los elementos que componen nuestro juego nos costaría una cantidad de tiempo con tendencia al infinito hemos optado por describir de una manera más generalista el flujo y avance del juego y sus estados.

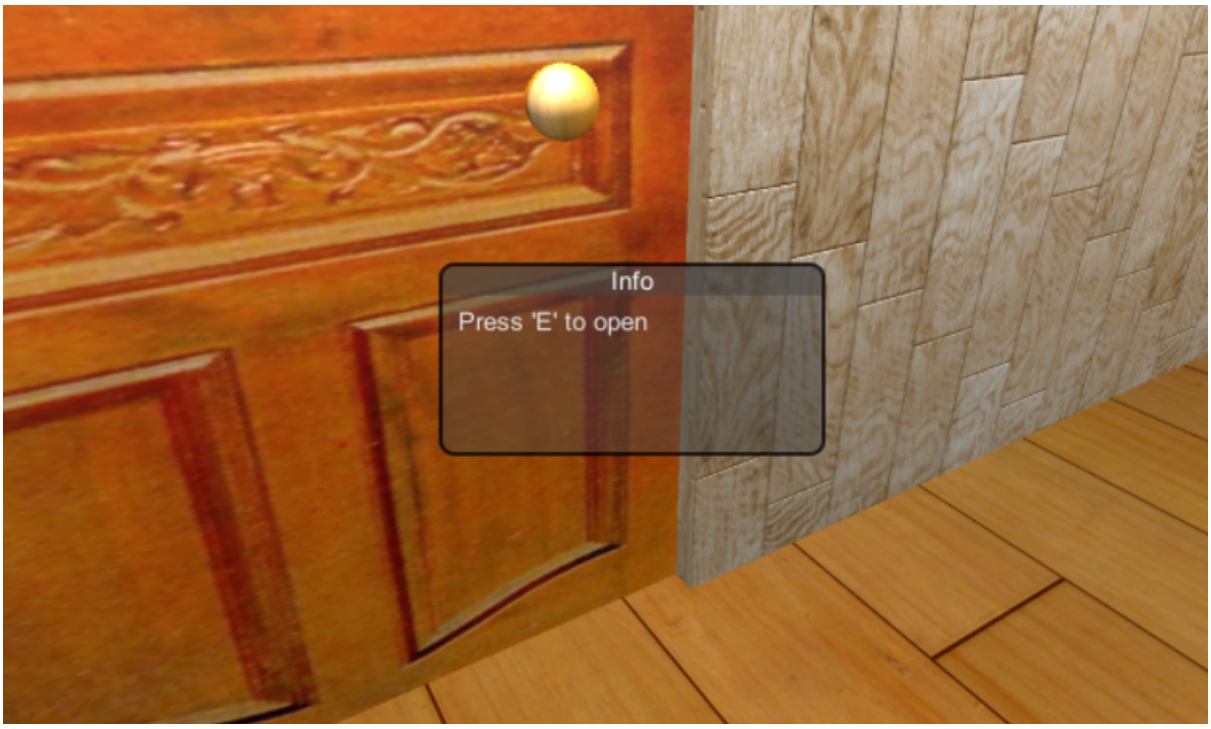
- Menú principal

Al abrir el juego, la primera pantalla que nos aparece es un menú principal. En éste, podemos seleccionar tres opciones: Start, Tutorial y Exit.
Al escoger Start, empezará el juego y tendrás que resolver acertijos pensando o por habilidad.
En Tutorial, hay tres niveles sencillos en los que aprenderás cómo controlar al personaje y podrás familiarizarte con las mecánicas básicas del juego.
Con Exit sales del juego.

Cuando ya estamos jugando, el usuario podrá moverse por una sala como la de la imagen y podrá interactuar con el sistema. Si estamos cansados de jugar o queremos pausarlo, con apretar la tecla escape nos aparecerá un pequeño menú en el que podrás seleccionar o reanudar la partida o salir del juego.



En este caso, la imagen corresponde al primer nivel del tutorial en el que únicamente tendrás que abrir la puerta. Para ello, deberás acercarte a ella y se mostrará un pequeño panel informativo con la tecla que has de presionar para poder salir de la habitación y pasar de nivel.



6.3.- Clases principales

FirstPersonController

Es la clase principal que el usuario maneja e interactúa con ella con el sistema. Simplemente consiste en la cámara de la escena que se mueve por la sala controlada por el usuario.

Triggers&Colliders

Son las clases que interactúan con el usuario y controlan los procesos, eventos, etc. del juego. Un ejemplo rápido sería el de la imagen anterior: la puerta contiene un componente llamado “Box Collider” que consiste en un trigger que detecta al usuario cuando éste entra en él.

Un trigger en Unity es una zona “invisible” que está situada en una zona determinada que se activa cuando un objeto entra dentro. Los triggers tienen tres métodos básicos: OnTriggerEnter, OnTriggerStay y OnTriggerExit. Estos métodos se activan cuando un objeto entra en el trigger, se mantiene en él y cuando sale de la zona.

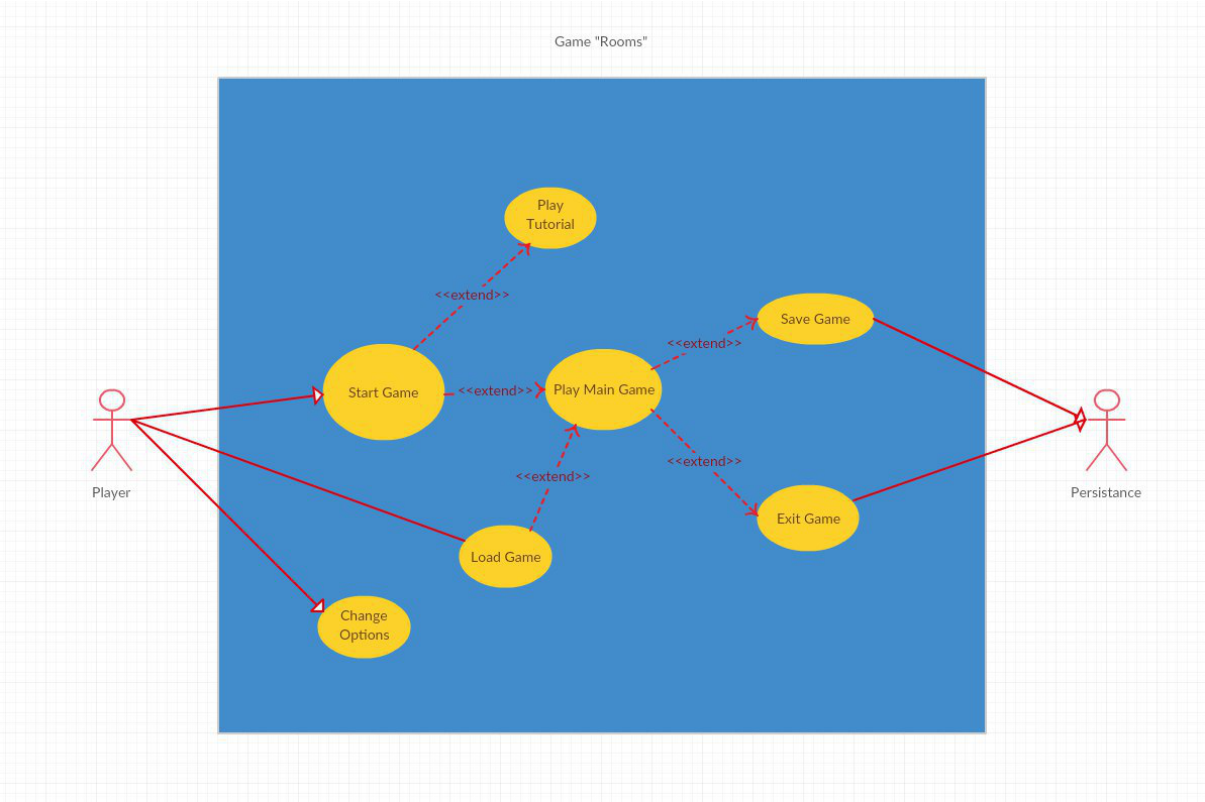
Volviendo al ejemplo anterior, cuando el personaje se acerca a la puerta y entra en el trigger, hay una clase que controla este evento y mientras el usuario está dentro

del trigger, aparece ese panel y espera a que presiones la tecla ‘e’ para que abras la puerta. Si sales del trigger, por mucho que le des a la tecla ‘e’ no podrás abrir la puerta.

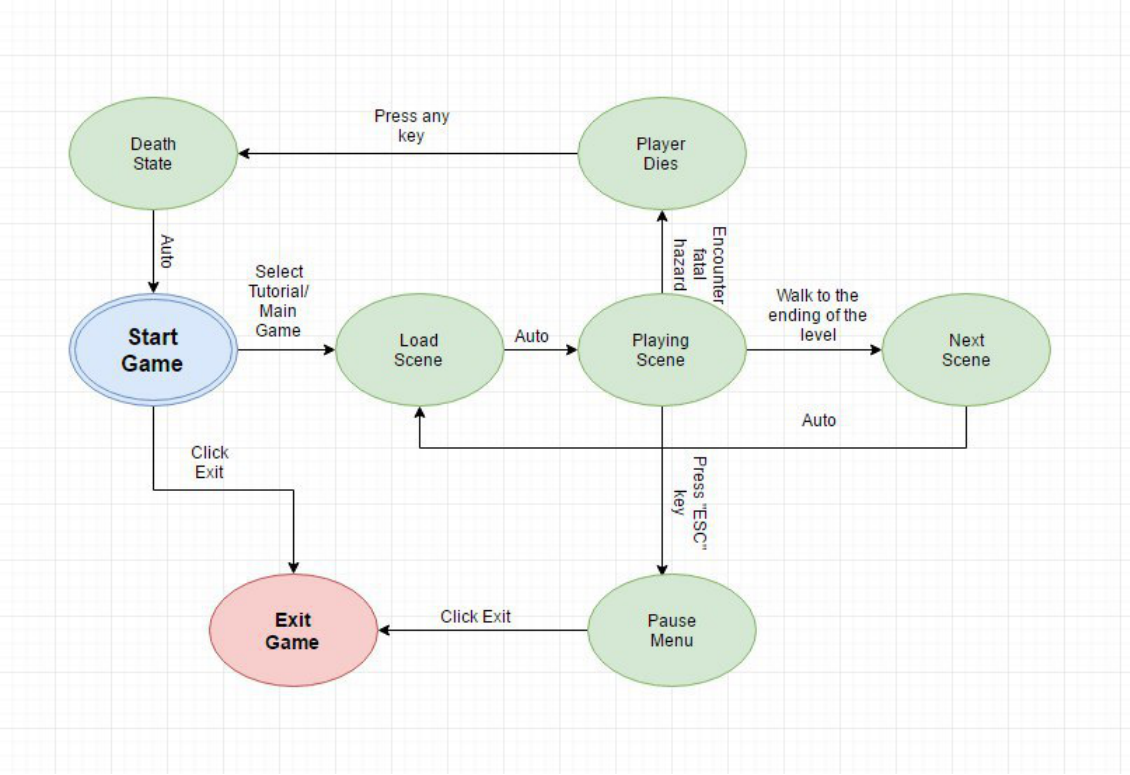
Otro ejemplo de triggers y colliders sería el siguiente:

Como se observa en la imagen, hay un muro en medio de la sala que hay que saltar. Éste muro contiene un trigger que, a diferencia del ejemplo de la puerta que debías entrar dentro para poder interactuar con la puerta, has de evitar entrar en él ya que en caso de acceder en el trigger, se reiniciará el nivel.

7.- Diagramas



En este diagrama de casos de uso se muestra el funcionamiento general de nuestro juego.



En la imagen podemos ver un diagrama de estados del funcionamiento del juego.