Universitat de Girona
Escola Politècnica Superior

Treball Final de Grau/Carrera

Realitzat i defensat a University of Technology and Economics (nom universitat) de Hungria (país)

Estudi: Grau en Enginyeria Informàtica Pla 2010

Títol: GAME DEVELOPMENT IN UNITY3D ENVIRONMENT

Document: Memòria Treball de Fi de Grau

Alumne: Pol Bosch Mestras

Director/Tutor: Dr. László Szirmay-Kalos
Departament: Informàtica i Matemàtica Aplicada

Convocatòria (mes/any): 12/2016

Dr. László Szirmay-Kalos

Budapest University of Technology and Economics

Faculty of Electrical Engineering and Informatics
Department of Control Engineering and Information Technology

Pol Bosch Mestras

GAME DEVELOPMENT IN UNITY3D ENVIRONMENT

SUPERVISOR

Dr. László Szirmay-Kalos

BUDAPEST, 2017

Contents

STUDENT DECLARATION

I, **Pol Bosch**, the undersigned, hereby declare that the present BSc thesis work has been prepared by myself and without any unauthorized help or assistance. Only the specified sources (references, tools, etc.) were used. All parts taken from other sources word by word, or after rephrasing but with identical meaning, were unambiguously identified with explicit reference to the sources utilized.

I authorize the Faculty of Electrical Engineering and Informatics of the Budapest University of Technology and Economics to publish the principal data of the thesis work (author's name, title, abstracts in English and in a second language, year of preparation, supervisor's name, etc.) in a searchable, public, electronic and online database and to publish the full text of the thesis work on the internal network of the university (this may include access by authenticated outside users). I declare that the submitted hardcopy of the thesis work and its electronic version are identical.

Full text of thesis works classified upon the decision of the Dean will be published after a period of three years.

Budapest, 20 March 2017

...……
Pol Bosch

Summary

In October 1958, Pong, was the first video game ever to be developed. It was a very simple tennis game that was the start of the video games that we see nowadays.

Video games have become part of many people's lives because of the hours of entertainment they provide and the level of detail they can have. Thus, the main task of this BSc thesis will be the development of a three-dimensional platform based video game using the modern game engine, Unity [1].

The video game consists of a character trapped in a level with obstacles and collectible items. The objective is to be able to get to the end of the level with the maximum amount of items picked. However, if the character falls down to the sea or steps on a trap, the player loses.

Using the Unity engine, I was able to create the traps and acknowledge when the character made contact with such traps. The same process is used to detect when the player falls down to the sea or picks up a collectable item. The character is able to shoot bullets to break glasses and advance through the level to finish the game.

The game also has the day and night cycle, which is possible thanks to the environment lighting of Unity engine. In the level, there is dusk and dawn, making changes in the lighting of the game.

All these features are possible with the scripts (coded in C#) that help to reach high level of details and the behavior of all the components of the video game.

Resumen

En Octubre de 1958 se desarrolló el primer videojuego, *Pong*. Era un videojuego de tenis muy simple que fue el principio de los videojuegos que tenemos hoy en día.

Los videojuegos se han convertido en una parte de la vida cuotidiana de la gente por la gran cantidad de horas de entretenimiento que proporcionan y el nivel de detalle al que pueden llegar. Entonces, el objetivo principal de esta tesis será el desarrollo de un juego de plataformas en tres dimensiones utilizando el moderno motor gráfico, *Unity*.

El videojuego consiste en un personaje atrapado en un nivel lleno de obstáculos y objetos coleccionables. El objetivo es llegar al final del nivel con el máximo número de objetos recogidos. Sin embargo, si el personaje cae al mar o pisa una trampa, se pierde la partida.

Utilizando el motor gráfico de *Unity*, he sido capaz de crear las trampas y poder capturar el momento en el que el personaje hace contacto con dichas trampas. El mismo proceso es utilizado para detectar cuando el personaje cae al mar o recoge objetos coleccionables. El personaje también es capaz de disparar balas para romper los cristales y para poder llegar al final del nivel.

También hay el ciclo día-noche, que es posible gracias al sistema de partículas de *Unity*. En el nivel amanece y anochece, la cual cosa modifica la iluminación del juego.

1 Introduction

The main task of the BSc thesis is to develop a three dimensional video game using Unity engine.

I have been playing video games since a very young age. I have seen them improve year after year but I never knew how developers were able to create such video games or how they could control every tiny detail in the behavior of the characters involved in them. Being able to know and understand what was going on behind the scenes of a video game was what motivated me to develop one of my own. This was the perfect way to see how video games are created, with my own eyes.

Unity is a very intuitive software and it also has the programming part integrated. With the visual design is possible to create the character, the traps, the level and all the items in the video game. Then, the behavior of these components is defined in the scripts. Every script refers to a component in the game.

This document is organized with the main aspects of the game and how they were developed. There is a section for each aspect and, in it, there is the design process of that part and how it interacted with the rest of the video game.

1.1 Introduction to Unity3D

Unity3D is a game engine developed by Unity Technologies and is used to develop video games for PC, consoles, cell phones and websites.

Unity puts a lot of emphasis on portability, the engine targets the following APIs:

- Direct3D on Windows and Xbox 360: Used to render in three-dimensional graphics in applications where performance is very important. It uses hardware acceleration of the 3D rendering pipeline.
- OpenGL on Mac: Abstract API for drawing 2D and 3D graphics. It is designed to be implemented mostly or entirely in hardware instead of software.
- OpenGL ES on Android and iOS: A subset of the OpenGL API designed for embedded systems. It is a cross-language and multi-platform API, that is why it is the most widely deployed 3D graphics API in history.

With Unity it is possible to make use of texture compression and resolution settings for all the platforms that the game engine supports. It also provides support for bump mapping, reflection mapping, parallax mapping, screen space ambient occlusion (SSAO), dynamic shadows using shadow maps, render-to-texture and full-screen post-processing effects.

Unity's graphics engine's platform diversity provides a shader with many variants and declarative fallback specification which allows Unity to detect the best variant for the current hardware.

In the same project, Unity provides the options to deliver to mobile devices, web browsers, desktops and consoles.

1.2 History of Unity

In Apple's Worldwide Developers Conference in 2005 Unity made it's first appearance to the public, since then, it has been extended to 21 different platforms and its current version in 5.4.3, released November 17th, 2016.

One year after the release, Apple named Unity as the runner-up for its Best Use of Mac OS X Graphic category. This made Unity the first game design tool to ever be nominated for this award.

With the release of Unity 5, a lot of inexperienced developers started producing games and this was a reason to criticize Unity. However, the CEO of Unity believed that this was a side-effect of Unity's success. In Q3 1026, games developed by Unity were downloaded more than 5 billion times and there are more than 2.4 billion unique mobile devices with games developed in Unity. This is why Unity is the leading global software for game industry.

1.3 Main concepts of Unity

The Unity workflow is built around the structure of Components. A component can be thought of as a smaller piece of a larger machine. Each component has its own specific job, and can generally (and optimally) accomplish its task or purpose without the help of any outside sources. Additionally, components rarely belong to a single machine, and can be joined with various systems to accomplish their specific task, but achieve different results when it comes to the bigger picture.

This is because components not only do not care about that bigger picture, but also do not even know it exists.

In the example of a PlayStation controller, it has many buttons, but each one has no idea that there are other buttons that have different behavior. The pressed buttons always do their job independently. The fact that the function of the controller is a one-way street, and its task will never change due to what it is plugged in to. This makes it a successful component, both because it can do its job as a standalone device, but also because it can do its job with multiple devices.

Unity lets you see everything you are working on, and in real time. This means you can test your project, see your project running in a separate window, make edits to your code or game objects, and see those edits reflected live. All of this is made possible by Unity's component-based architecture.

2 Explanation and analysis of the task

The video game consists of one character in a level with obstacles in form of traps or holes in the floor and collectable objects. The game behaves in a physical plausible way. The character can move around the level and it has to avoid the red traps or falling down the sea. If it steps on traps or falls down an explosion will be played and the game will be lost. However, if the character reached the end of the level and picks up the green cube, the player wins. Besides that, there is the day and night cycle that modifies the lighting of the game. However, there are a lot of different concepts involved inside the behavior of the components.

2.1 Physics

Physics are a very important aspect of every video game. With physics it is possible to control the movement and some behavior of the components in the game, as well as making it feel more realistic. In the game, the character can run, turn left, right and jump. To be able to make these moves, forces are applied to the main character.

Unity engine programs on its own the main forces of the game, like gravity. However, in the game I developed, the character can shoot bullets and if these bullets hit glass, the glass will shutter in many pieces making all these pieces fly away from the original position. This is when the physics come into play.

The behavior of this situation is described in the script Break.cs:

```
void OnCollisionEnter(Collision collision){
        if (collision.collider.tag == "Projectile") {
            Destroy (gameObject);
            Instantiate (brokenObject, transform.position, transform.rotation);
            brokenObject.localScale = transform.localScale;
            Vector3 explosionPos = transform.position;
            Collider[] colliders = Physics.OverlapSphere(explosionPos, radius);

            foreach (Collider hit in colliders) {
                if (hit.GetComponent<Rigidbody> ()) {
                    hit.GetComponent<Rigidbody> ().AddExplosionForce (power * collisi
on.relativeVelocity.magnitude, explosionPos, radius, upwards);
                }
            }
        }
    }
```

As seen in the script, every time a bullet (Projectile) collides with a glass, the glass will be destroyed and another object will appear in its place. This new object is the exact same glass but shuttered in many pieces. All these pieces are stored in the array named colliders and then for each piece, a new force is applied making each peace fly in a different direction. This caused some problems because depending on the position of the character, the bullet was going through the glass between consecutive frames and the collision wasn't detected. To fix that, I had to change the collision detection mode of the bullets to be continuous dynamic with this line of code in the script ProjectileShooter.cs:

```
rb.collisionDetectionMode = CollisionDetectionMode.ContinuousDynamic;
```

There is also physics involved in the behavior of the bullets that the character can shoot. The bullets are affected by the gravity so they slow down every frame and they eventually fall down to the floor. The same happens with the main character and all the other components except for the main floor and walls of the level, everything is affected by gravity.

2.2 Camera behavior

The behavior of the camera is something that has to be thought with a lot of details because it is one of the main concepts of the game that can bring a good or bad user experience. The camera is another component of the game, so it follows the physics and the other rules.

On a very early stage of the game, the main camera was set up as a First Person Camera so the player was able to move its vision with the cursor. However, after discussing about it, the last version of the game is played in a Third Person Camera so the player can see the character at all times and can move the camera 360 degrees around the character.

This camera follows the character from behind and when the player presses the right click of the mouse and moves the cursor, the camera moves, always around the main character. This is possible with vectors and quaternions, as shown in the script CameraController.cs:

```
void LateUpdate () {
    if (Input.GetMouseButton (1)) {
        currentAngleX += Input.GetAxis("Mouse X") * speed * Time.deltaTime;
        currentAngleY += Input.GetAxis("Mouse Y") * speed * Time.deltaTime;
    }
```

```
        Quaternion q = Quaternion.Euler(-currentAngleY, currentAngleX, 0);
        Vector3 direction = q * Vector3.forward;
        transform.position = player.position - direction * distance;
        transform.LookAt(player.position);
    }
```

There is also a First Person Camera used when the player presses the key “F”. During this camera mode the player is able to shoot bullets to break the different glasses around the level.

There is a script to switch from the Third Person Camera and the First Person Camera. This is a long script but the main point is the following: VDCameraToggle2.cs:

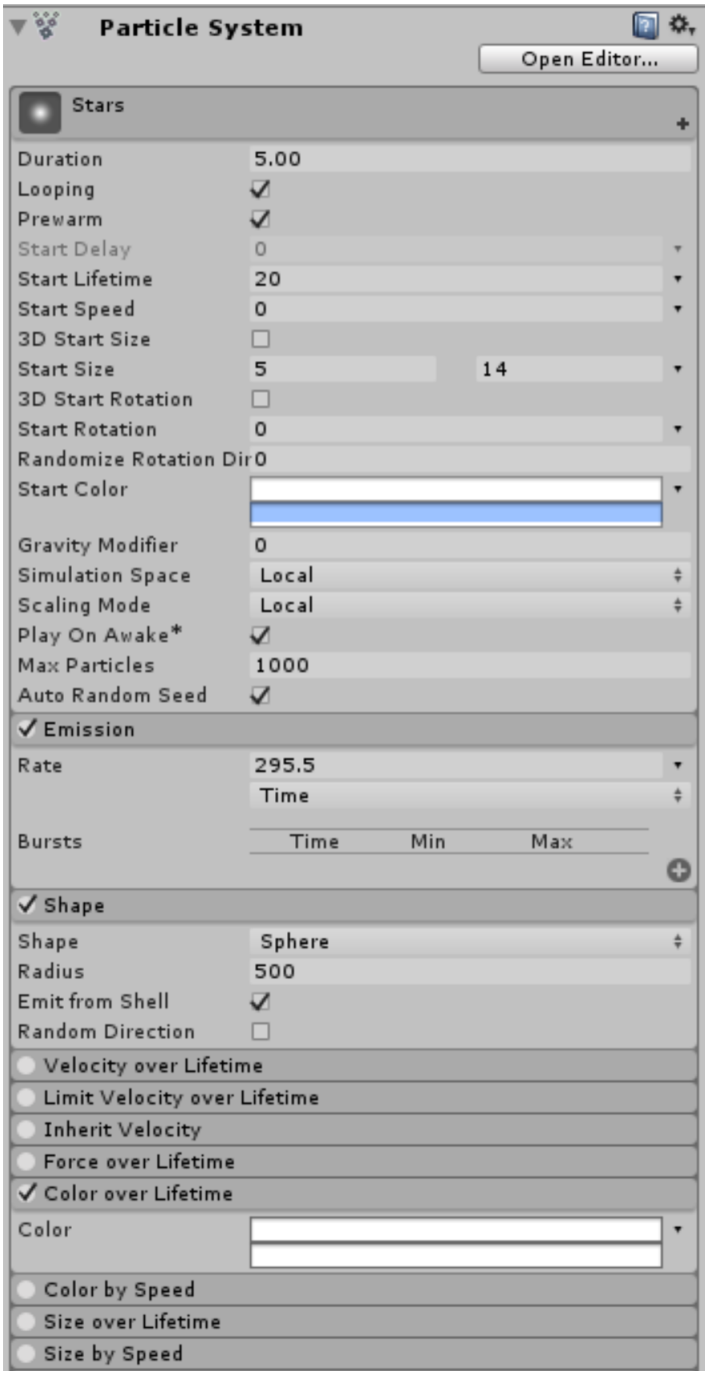
```
if (Input.GetKey(KeyCode.F) && LastKeyInput + KeyDelay <= Time.realtimeSinceStartup)
// Can be used instead of the above
        {
            if (CurrentCamera + 1 < CameraArray.Length)
            {
                // Disable current camera and enable next camera
                CameraArray[CurrentCamera].gameObject.SetActive (false);
                CurrentCamera++;
                CameraArray [CurrentCamera].gameObject.SetActive (true);
            }
            else
            {
                // Disable current camera and enable first camera
                CameraArray[CurrentCamera].gameObject.SetActive(false);
                CurrentCamera = 0;
                CameraArray[CurrentCamera].gameObject.SetActive(true);
            }

            LastKeyInput = Time.realtimeSinceStartup;
        }

        if (CurrentCamera == 1) {
            player.gameObject.GetComponent<Renderer> ().enabled = false;
            player2.gameObject.GetComponent<Renderer> ().enabled = false;
            derrick.GetComponent<Animator> ().speed = 0;
        } else {
            player.gameObject.GetComponent<Renderer> ().enabled = true;
            player2.gameObject.GetComponent<Renderer> ().enabled = true;
            derrick.GetComponent<Animator> ().speed = 1;
        }
```

This part is the one that enables or disables one camera or the other when the switch happens. If the current camera is the First Person Camera, (CurrentCamera == 1), the character renderer is set to false so it becomes invisible and the animator speed becomes 0 to avoid character movement. Using this method, the character cannot move while being on the First Person Camera. When it goes back to the Third Person Camera, the character becomes visible again and can move.

2.3 Particle system

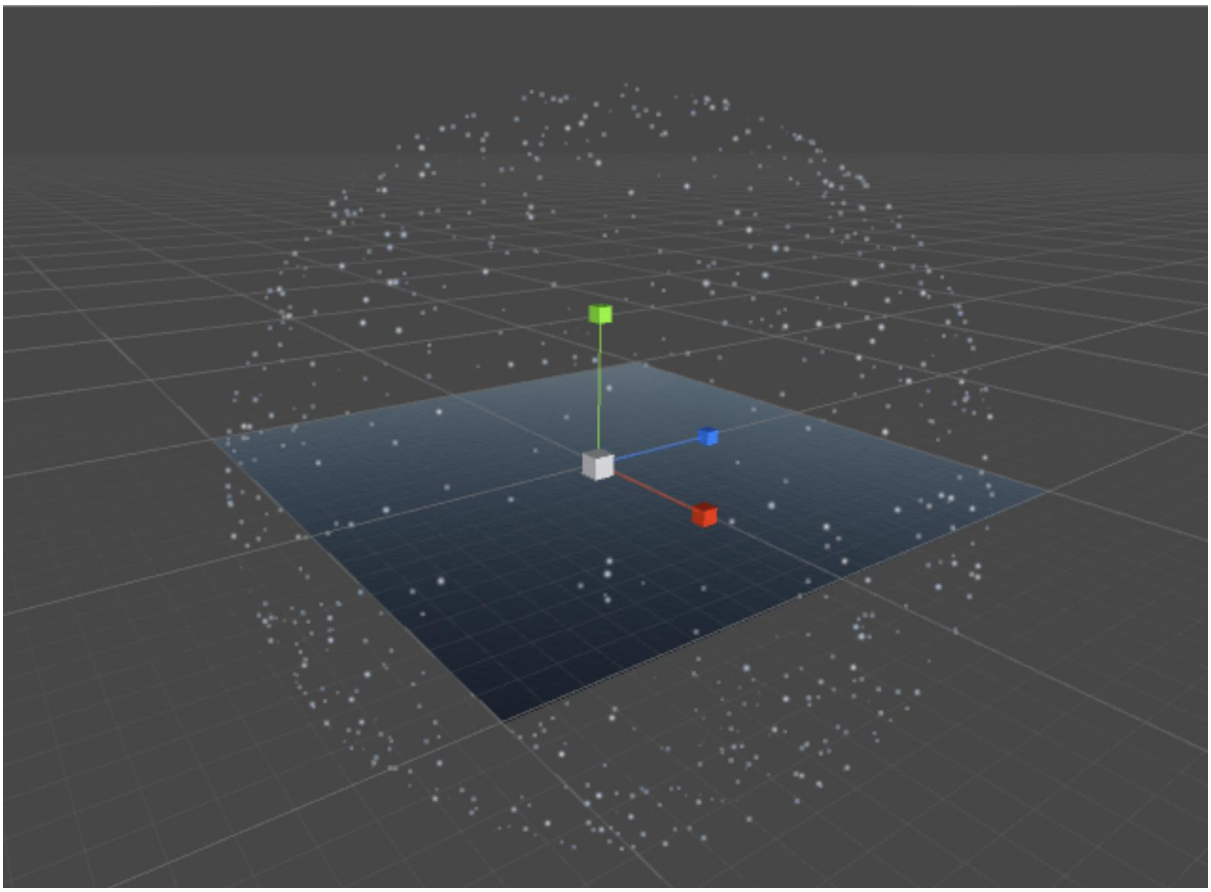
Unity is able to create particles over time using a point as a reference, this is the particle system. While it can be used in different situations, in this game it is used to simulate the stars in the sky.

The stars of the game are many particles centered in a point and they rotate around it. There are many parameters that set up all the details about their behavior but the most important ones, in Unity are the following:



2. 1 figure: Parameters of the particle system

The particle system is set as a sphere with 500 units of radius and the animation loops. Like this, the particles look like stars that move on the sky when the sun goes down. During the day, the stars are barely visible. It is also possible to set the maximum amount of particles as well as the colors.



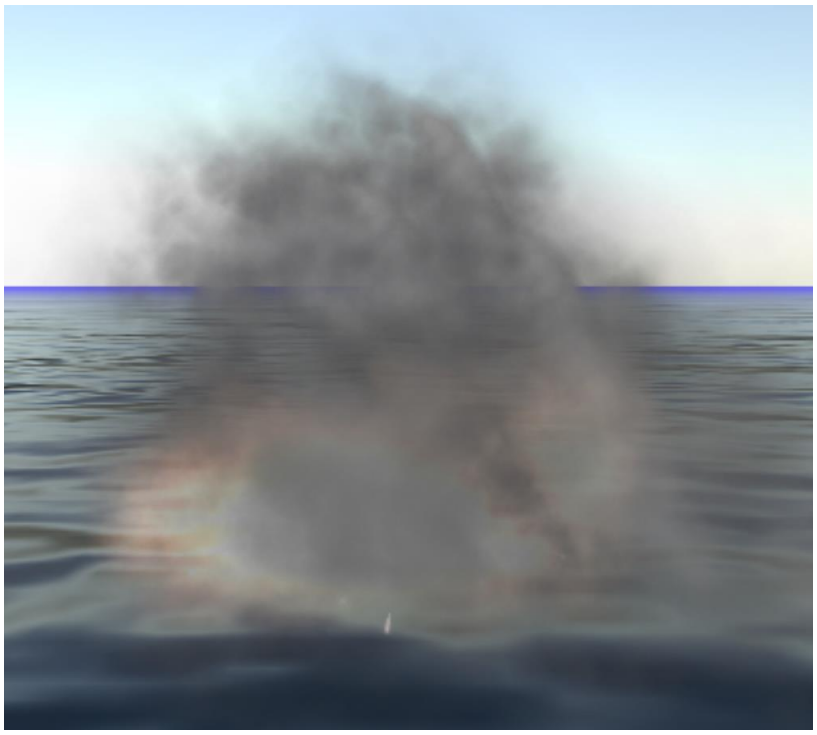
2. 1 figure: Final result of the particle system

2.4 Effects

It is always possible to create many effects in a game, such as explosions, lasers or water. In the game however, there are not many effects implemented because they are not a crucial part of it.

There is one special effect that is played when the character loses. When the character steps on a trap or falls down to the sea, a small explosion is created and the character disappears.

This was possible with the power of prefabs from Unity. There are many explosion prefabs but I chose a small one and added it to the game. Its behavior is designed in the script DetonatorShockwave.cs that has all the parameters about the explosion which in the end, it is a particle system as well.

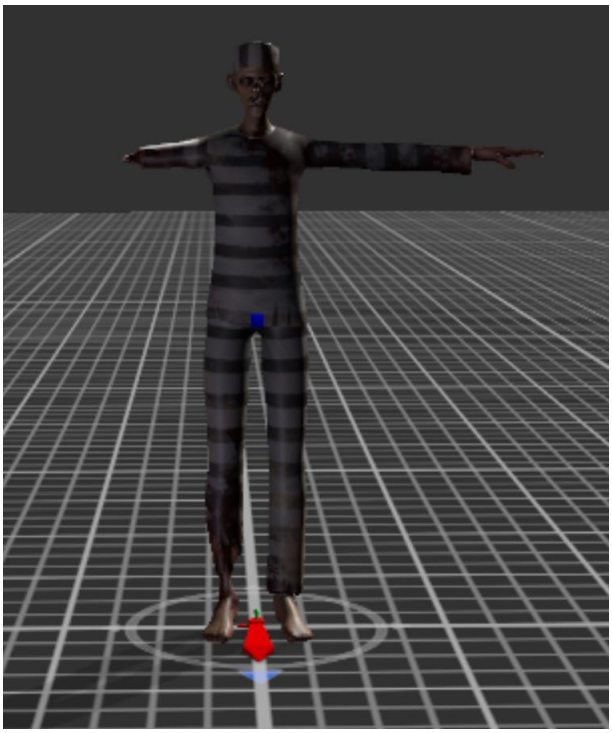


2. 2 figure: Explosion when the character loses

2.5 Animations

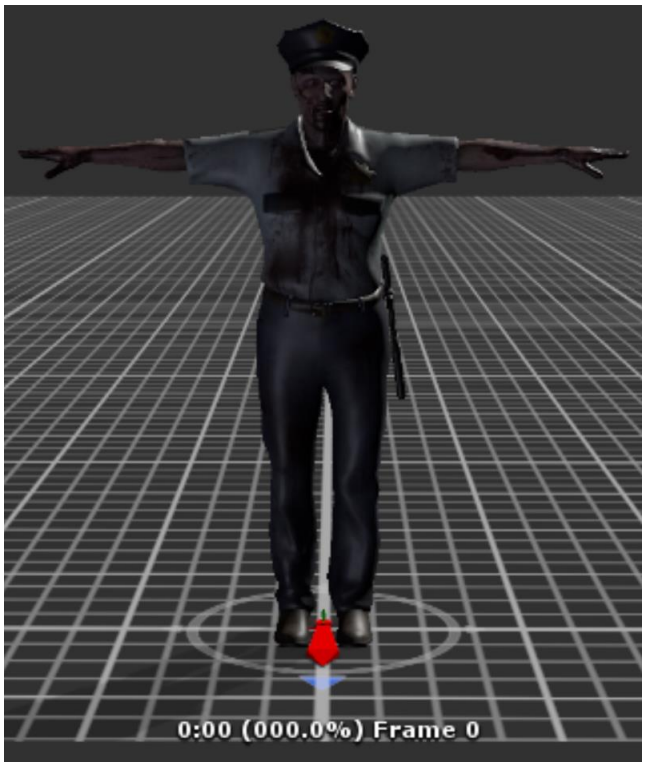
The animations of a game are what make the game smooth. They require a high level of detail and it is very hard to design all the transitions and movements to fit the main character. In the game this was also the hardest part.

As mentioned above, when the character was a sphere there were no animations to be played. Then, using the website www.mixamo.com [16] I imported a new character with the standard pack of animations (Idle, turn right, turn left, walk and jump).



2. 3 figure: First character

The problem with this character was that it is a zombie and when I tried to apply the Humanoid animations to it they did not work because the character was missing its right arm. There were many problems with this and in the end the best solution was to change the character to a real Humanoid and configure all the animations again.



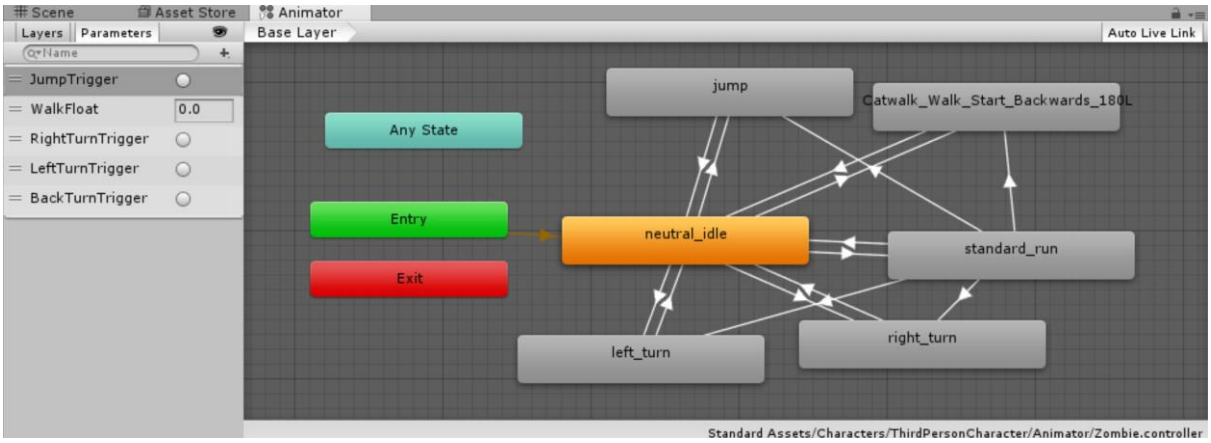
2. 4 figure: Final character

With this second character all I had to do was configure all its animations as Humanoid and with this, the movement of the character could be controlled by the animator.

All the animations were configured in the animator to match the character control. There are two parts of this:

2.5.1 Animator

The animator is a component that manages all the animations from the character. It works like a state machine where the different states are the animations to be played and it has its transitions from one state to another. In the animator component all the animations need to be configured, alongside the transitions from one to another. Each transition has duration and conditions (expressions of the animator, parameters or triggers) that need to be met in order to start. It starts at the “Entry” state and it immediately moves to the next state, which in this case is the animation “natural_idle”. Then it moves to the next animation depending on which move the player wants to do.



2. 5 figure: Animator

As seen in the animator, the first animation of the character is the "neutral_idle", where the animation is played as a loop and it is the go-to animation when there is no movement. From this state the animator plays one or another animation depending on the key that has been pressed. When the "Space" key is pressed, the transition from "neutral_idle" to "Jump" happens and the animator plays the "Jump" animation causing the character to jump and move. Once this animation is finished, the "neutral_idle" is played again until another key is pressed.

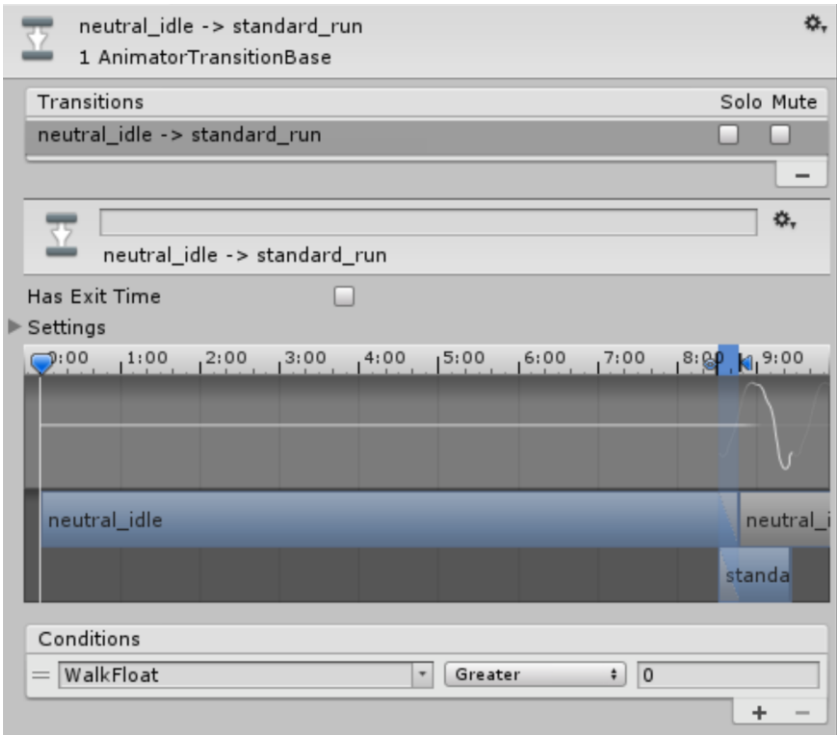
All the parameters for the transitions are set as triggers except for the "WalkFloat" parameter which is the one that triggers the transition from "neutral_idle" to "standard_run". It is designed this way because the "standard_run" animation is a loop that does not stop until the player releases the W key. If a trigger is used, when the player presses W, the "standard_run" animation is played once and then the trigger goes back to normal making the animator play the idle animation again. Since I did not want the animation to be played only once, it was set up as a float. To control the behavior of this transition and all the others, there is a script attached to the character component.

2.5.2 Script

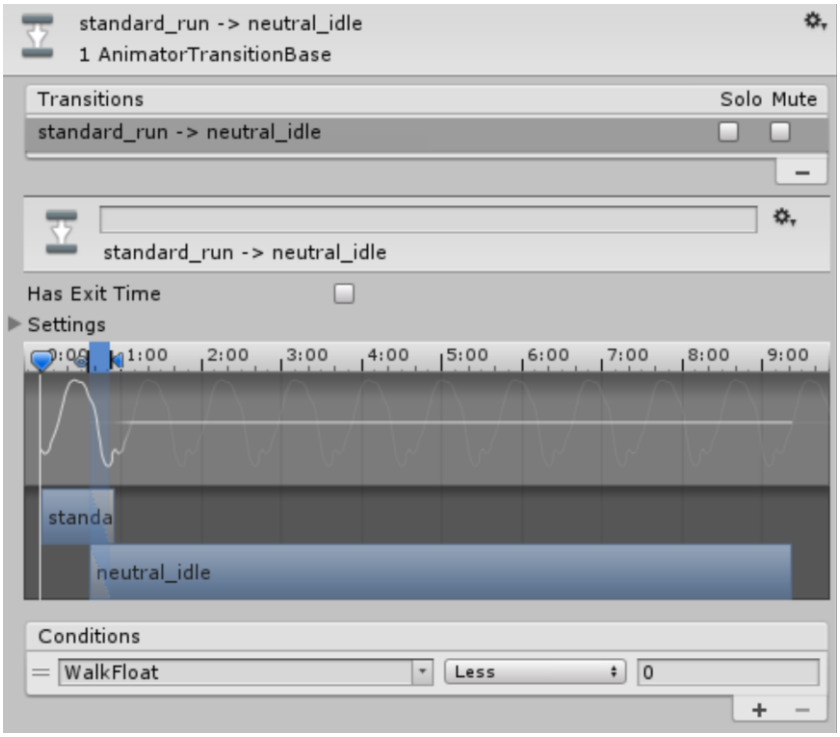
The script attached to the character is what captured the input from the keyboard so that the animator can play the animations and move the character. Any parameter from the animator needs to be the same in the script. So all the triggers and the float are set up in the PlayerBehavior.cs script:

```
void Update()
{
    if (Input.GetKeyDown (KeyCode.Space))
        animator.SetTrigger ("JumpTrigger");

    if (Input.GetKeyDown (KeyCode.W))
        animator.SetFloat ("WalkFloat", 1.0f);

    if (Input.GetKeyUp (KeyCode.W))
        animator.SetFloat ("WalkFloat", -1.0f);

    if (Input.GetKeyDown (KeyCode.D))
        animator.SetTrigger ("RightTurnTrigger");

    if (Input.GetKeyDown (KeyCode.A))
        animator.SetTrigger ("LeftTurnTrigger");

    if (Input.GetKeyDown (KeyCode.S))
        animator.SetTrigger ("BackTurnTrigger");
}
```

As seen in the code, all the keys set a trigger to start the transition except for the W key which sets the parameter from the animator to 1. Then the animator starts the transition from “neutral_idle” to “standard_run” and it keep playing the running animation until the parameter “WalkFloat” is set to less than 0, which happens when the W key is released.



2. 6 figure: Transition from idle to running



2. 7 figure: Transition from running to idle

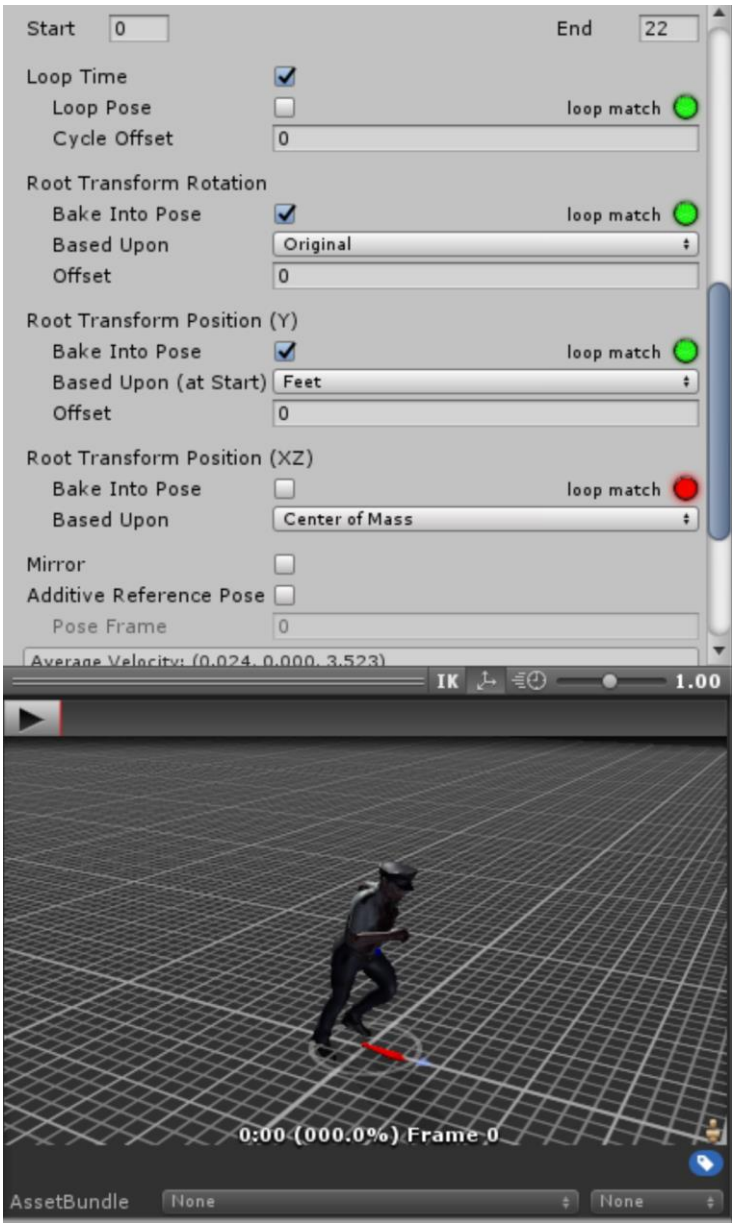
2.6 Character control

Character control is a really important aspect of every game since it is what gives the player the feeling of being in the game. There are different ways of implementing the character control and there have been many changes throughout the game.

In an early stage of the game, the character was a ball, so the best way to move it was using the most standard character movement keys (W, A, S and D) plus the help of a script. Each key added a force to the character that resulted in a move towards that direction.

This was a very good approach when there were no animations on the character, only the ball rolling on the map. But when the character was changed from a ball to a human with legs and arms, this way of moving the character started to fail.

Since the movement of the character and the animations of the body had to match, I changed the character control to be controlled by the animator component from Unity. The animations already have the character displacement so I only had to tweak the speed and the character orientation.



2. 8 figure: Animation settings

In the example of the “standard_run” animation, the character runs and this animation is played as a loop until the character stop running. As seen in the picture, the orientation and displacement of the character are controlled by the animation itself. To enable this, it is necessary to use the functionality of Unity called “apply root motion” which automatically updates the position and orientation of the character according to the animation.

2.7 Lighting

Lighting is a key aspect to make a game incredibly attractive. With the light and reflections the visual details of the game are brought to a whole new level.

In the game, there are three components that together make the lighting.

2.7.1 Sun

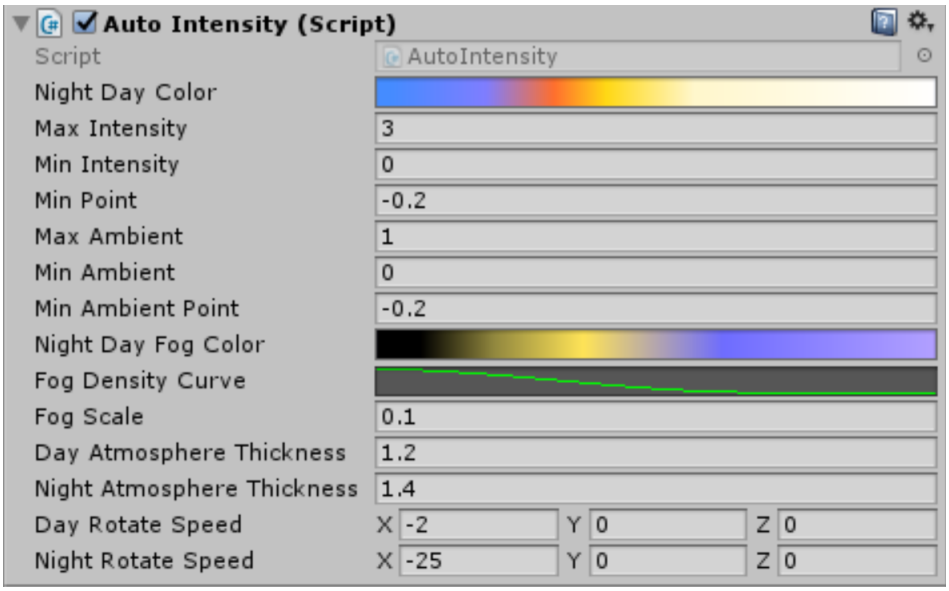
The Sun in the game has the Light component from Unity, therefore it is the main source of illumination of the game. As mentioned before, the sun rises and sets changing the lighting and creating shadows on the level. There are two scripts attached that control the behavior and the parameters of the sun:

SetSunLight.cs:

```
void Start () {
        sky = RenderSettings.skybox;
    }

    void Update ()
    {

        stars.transform.rotation = transform.rotation;

        Vector3 tvec = Camera.main.transform.position;
        worldProbe.transform.position = tvec;

        water.material.mainTextureOffset = new Vector2(Time.time / 100, 0);
        water.material.SetTextureOffset("_DetailAlbedoMap", new Vector2(0, Time.time
/ 80));
    }
```

In this script I have coded how the sun is going to be reflected against the sea. This is very noticeable when the sun is about to set since it is really close to the sea.

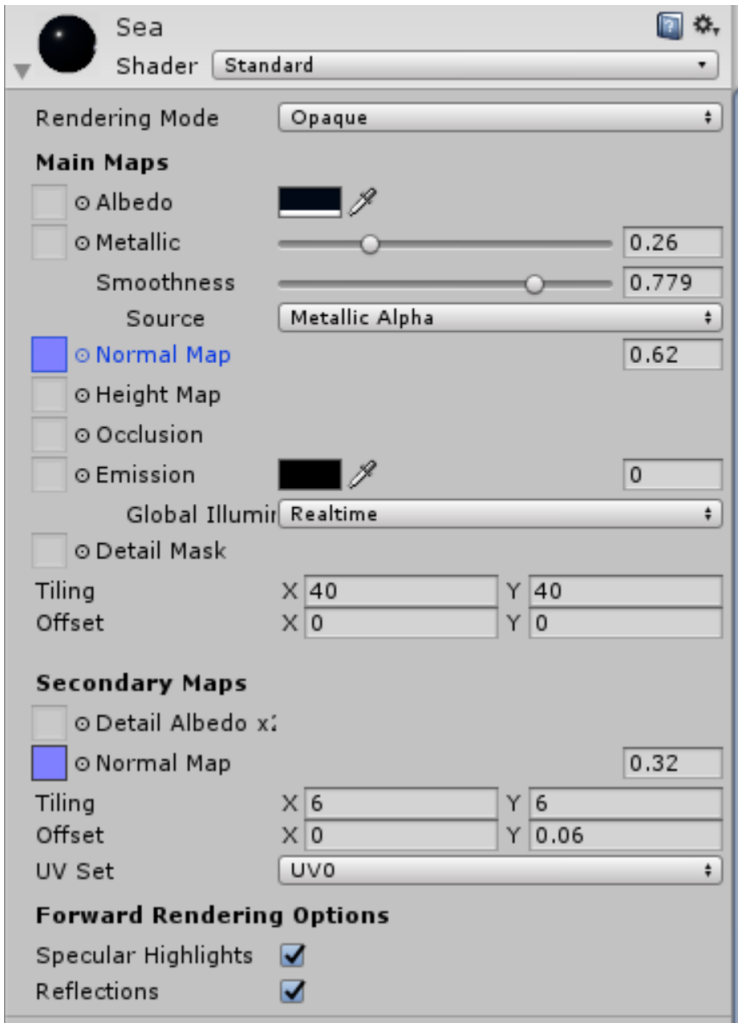
In the other script, AutoIntensity.cs there are all the parameters and how they change based on the current time in the game, from the intensity of the fog to the speed of the day-time and night-time. Many of these parameters are easily tweaked manually using the Unity interface:



2. 9 figure: Parameters of the script

2.7.2 Sea

The sea in the game is a huge plane that reflects all the light in the level. The key to get the effect of moving water is a material that has two normal maps. The texture of the normal map is from the Standard Assets of Unity and when they are stacked one on top of the other they create the effect of waves. This is accomplished when the two normal maps are applied one on top of the other with the following settings:



2. 10 figure: Parameters of the sea texture

2.7.3 Sea reflections

The sea reflection is designed with the component Reflection Probe from Unity. A Reflection Probe is rather like a camera that captures a spherical view of its surroundings in all directions. The captured image is then stored and can be used by objects with reflective materials like the sea. The visual environment for a point in the scene can be represented by a cubemap. This is conceptually like a box with flat images of the view from six directions (up, down, left, right, forward and backward) painted on its interior surfaces. For an object to show the reflections, its shader must have access to the images representing the cubemap. Each point of the object's surface can "see" a small area of cubemap in the direction the surface faces (ie, the direction of the surface normal vector). The shader uses the colour of the cubemap at this point in calculating what colour the object's surface should be. Once the component is created, the only thing left to do was to tweak

the parameters (clipping planes, type and refresh mode as well as the position) and check the results in Unity. After some tests the reflection on the water looked realistic.

All these three components together bring some basic elements of lighting to the game. When this was correctly implemented, the game looked really different and way more realistic.

3 Testing

Every single project requires a lot of testing and developing a video game is no different. There were many hours of tests behind all the components of the level, especially with the character animation.

I have been testing the game since the very beginning to make sure that what I was developing was working as intended.

The way I tested all the components was: I created the breakable glass and placed many of it in a huge plane. Then I moved the character around the level always interacting with the component that I was testing, the breakable glass in this case. I had to make sure that the glass was broken when the projectiles that the character was shooting hit it and that the broken pieces were affected by gravity and the added forces of the collision. This test was not too difficult because everything was written in the script and I knew exactly what values to tweak or what lines to change.

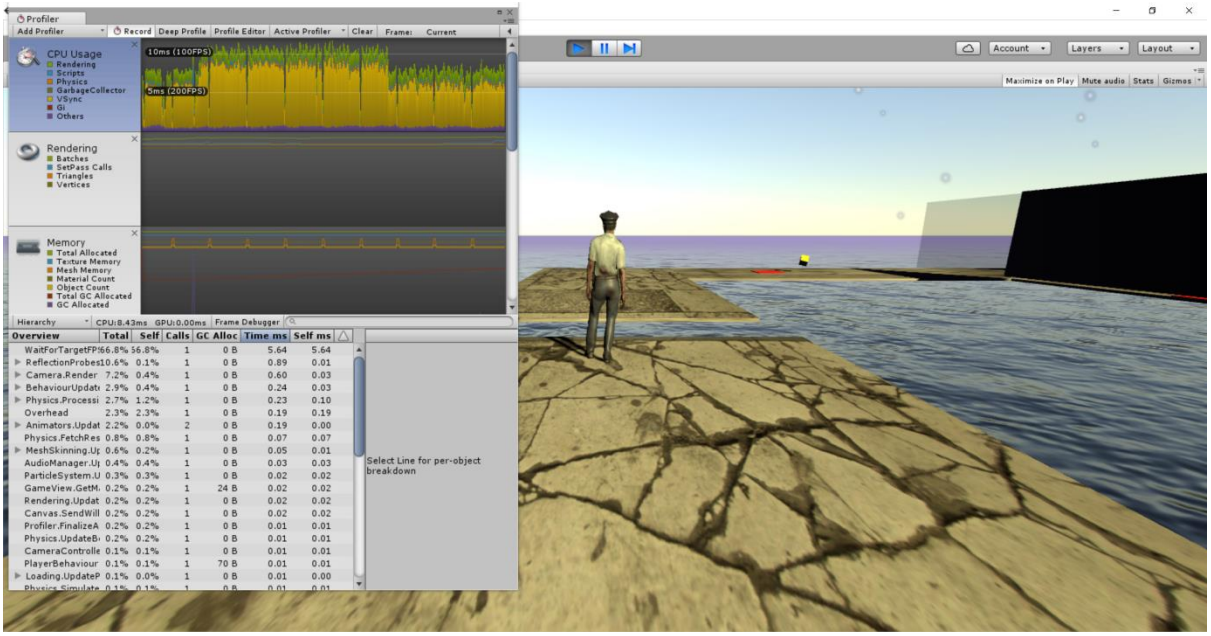
However, the worst part of the tests was the animations. From the very beginning there were many problems with the animations because of the wrong character and this made the tests irrelevant. I was testing the animations but I was not testing the main problem. After switching to the final character, I was able to test the animations observing the animator.

When an animation is played, it is visible in the animator component and it is possible to see how many frames of the animation have been played and when the next transition is going to start. During these tests, I realized that I could not use float parameters for animations that did not loop, I also noticed that the parameter “Has exit time” should not be active for animations that could be interrupted, like the Idle or running animation.

The last part of testing was to play the level itself. While it was not too complicated because all the components were tested separately, completing the level and making sure that it was playable was the most important test of all. During these tests I found some platforms in the level that were impossible to reach, collectable objects not behaving correctly and blind spots where there was absolutely no illumination. To fix first problem I simply had to move the platforms a little bit closer to each other. Taking a look at the jump animation made this part really easy since I knew the distance of each jump. For the second problem, I realized that there were some collectable objects that were not tagged as a Pick Up object, that was the reason why the character could not pick up the item. For the last problem I had to edit the lighting or change the level to avoid having

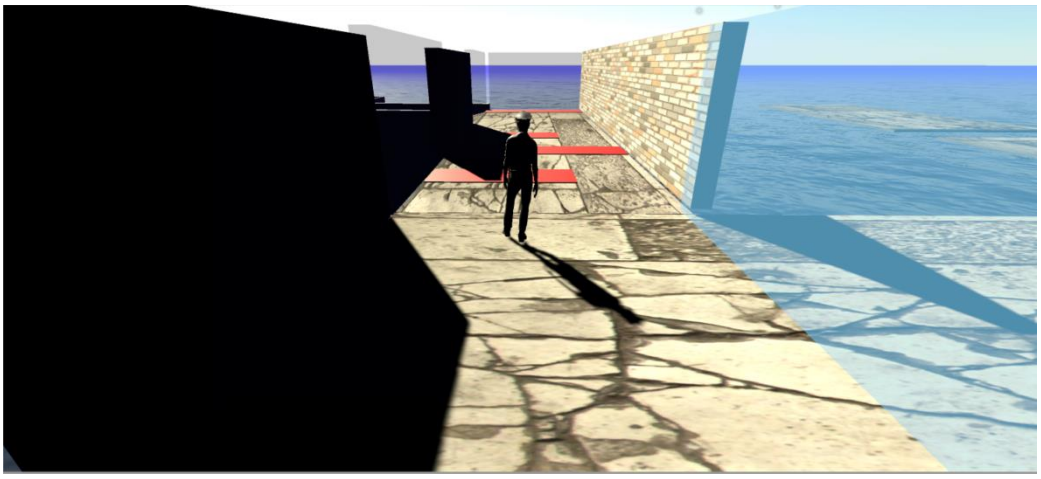
completely dark areas in the level. In the end, the easiest approach was to change some walls and place breakable glass instead. Like this, all the main problems were solved.

Another aspect that had to be tested was the performance of the game. The following figure reports for the time spent in the various areas of the game. For example, it can report the percentage of time spent rendering, animating or in your game logic. In my case, I analize the performance of the CPU. As we can see in the following picture, the critical part is the rendering, it is what takes the longest, but in the end the overall performance was really good.

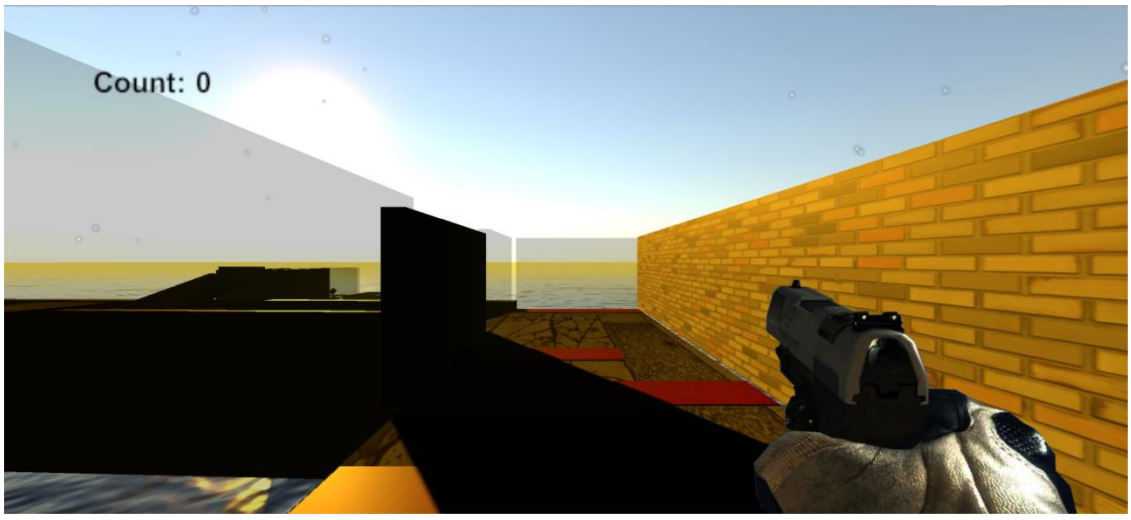


3. 10 figure: Performance windows of Unity

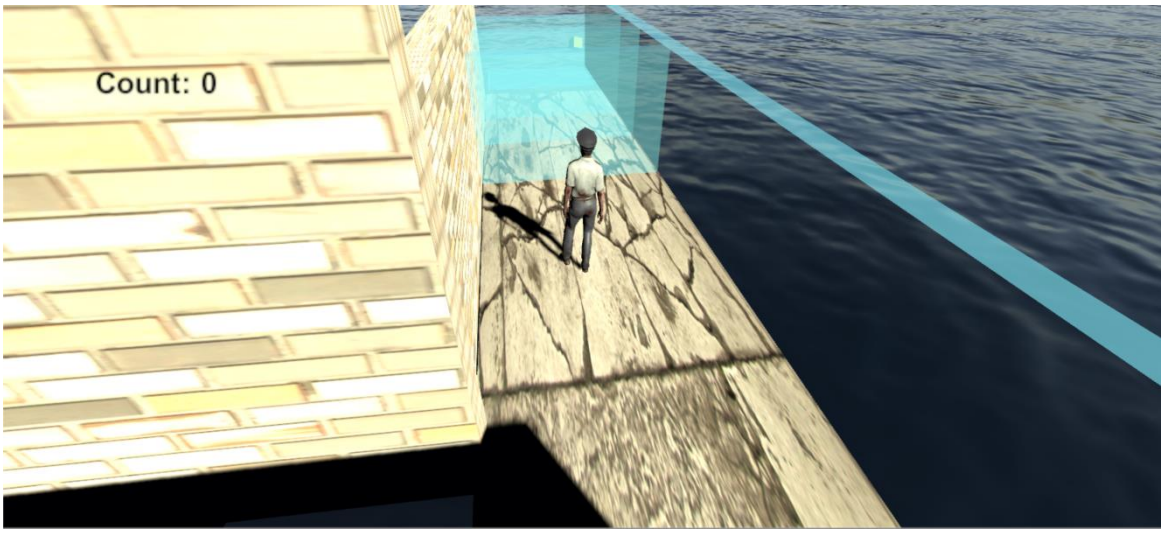
The FPS is stable as well as the rendering. It is noticeable that the FPS increase when the character starts moving however the rendering keeps being high. This indicated me that the performance was excellent.



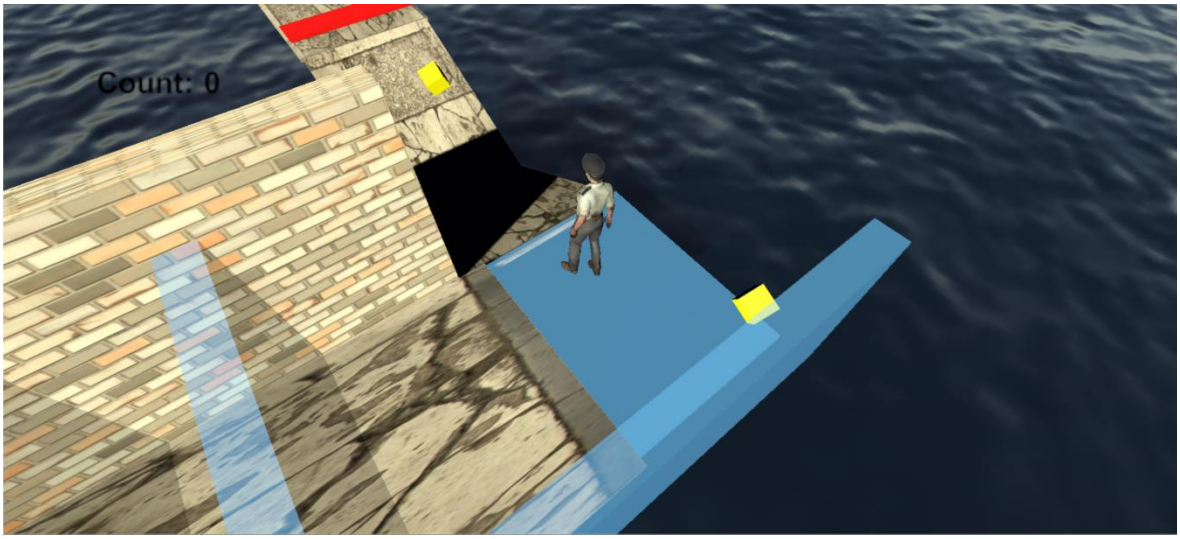
3. 21 figure: Traps of the level



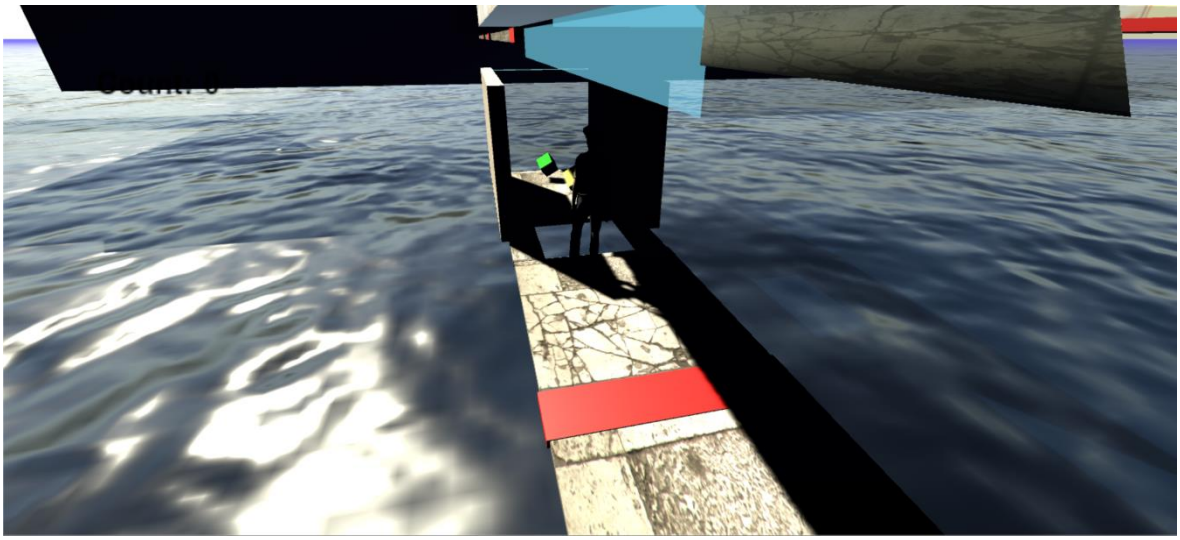
3. 13 figure: First Person Camera



3. 14 figure: Breakable glasses



3. 15 figure: Collectable items



3. 16 figure: Green item to finish the game

3. 17 figure: Winning the game

3. 18 figure: Losing the game

4 Challenges

During the development of a game a lot of ideas cross your mind. One realizes that while working on the character there are many animations to improve, a lot of details to add or increase the amount of movements that the character can do. This happened while developing the whole game, not only with the part of the character development.

This was the first video game that I have ever developed and I was really excited about it. During the first weeks the main goal was to have an awesome game with many levels and a clear objective in each one. The main point of the BSc thesis was to understand how game development works in a basic level so I focused on creating a small working level with a simple sphere and from there I would keep improving it adding more features.

From the very beginning there were challenges, starting with the physics. Detecting collisions was the first big feature of the game that used the Unity interface and the Scripts all together. This was the point where I had to make the Unity interface and scripts work together.

Another challenge was the implementation of characters and its animations from the internet. Since I had only used the ball from the Unity standard assets, there were many problems importing the characters from an external source.

Finally, the biggest challenge I faced during the development of the game was the character animations. This was something absolutely new to me and I had no idea how to start. After many hours of tutorials and discussions the animator was working. The problems were the transitions between animations. I had to understand what was going on behind the animator to be able to make the transitions smooth and to cut the animations when I wanted to. This was really problematic because I could not simply disable the animator; instead, I had to reduce the speed of all the animations to 0. Using this trick the animations would never start, so the character would not move.

5 Improvements

When this project was starting there were many ideas in my head that I wanted to make real. However, the main point of the game development task was to take a glimpse of game developing, not to make a professional video game.

Having this in mind, I changed the priorities of the game and I started from the simplest things such as the rolling ball to finish with the final game.

There is quite a long list of improvements that the game could have because the possibilities are endless. Considering the time I had to develop the game and the fact that I had absolutely no experience with game developing before, I am satisfied with the final results. In any case, there still are improvements that could be applied to the game if I had more time:

5.1 Camera behavior

The camera behavior was the first aspect I focused on after I finished the basic level. I wanted the camera to follow the player and to rotate around while the mouse was moving instead of having to press a button and move the mouse to move the camera.

This was developed and added to the game in the last weeks but there were problems when the character was too close to a wall and the camera had to change its position to avoid being behind the wall having the character out of sight.

After many tests I had to remove this feature of the final version of the game. I managed to make the camera follow the player and move at the same pace as the cursor but when it hit walls the problems started. If I had more time this would have been the first aspect to work on and improve.

5.2 Lighting

As stated before, lighting is what brings the graphics of the game to the next level. In the game, the lighting is good but it lacks reflection or different levels of darkness in the shadows. Right now, all the shadows are absolutely black instead of a mix between blacks and whites.

The reflection on the floors is also non-existent. During the weeks I worked on the lighting I also implemented reflection on the ground and the glasses but it did not work as intended so it did not make it into the final version of the game. With more time to work with lighting and the reflection probes of Unity I would be able to improve the overall graphic details of the game.

5.3 Character animation and movement

One of the poorest aspects of the game is the character movement. The movements of the character are dictated by the animations that I used and since they are not customized they have limited possibilities.

The character and its animations are imported from www.mixamo.com so I had to choose the ones that fit the best with my model and the final level.

One of the biggest improvements this game could have, would be having its own character and the animations. However, this would take a big amount of time that I simply do not have. There were many problems with the character animation when simply importing the character and having it behave like I wanted to, that creating my own character and animations was simply out of the scope.

The character only has six animations which are more than enough for the current state of the game but there should be more if I plan on expanding the game adding more levels and more features.

5.4 Expanding the game

Right now the game consists of one level only. As I said this BSc thesis was to take a glimpse at how game development worked and get familiar with the Unity engine. The game I developed met these requirements perfectly because it had many features that need to be present in all the games. There was no need to design more levels even though it would have been a big improvement if I had more time.

The current level has all the features of the game so expanding it into more levels would not have meant to develop new features, but to design a different combination of the current ones. While I think it would have been an improvement it does not have a high priority in my list.

6 Conclusions

When the idea of developing a game crossed my mind I was really excited because I have been all my life playing video games and I finally had the opportunity to do one of my own to understand what is going on behind the main screen of the game. This BSc thesis has fulfilled that aspect without a doubt.

In the early stages of the game I had many ideas in my head and I wanted to make all of them real but when the game started to grow I realized I would have to remove some features from the original list simply because I was not experienced enough or I did not have enough time to develop all of them.

There were some weeks where the tasks to do were complicated, especially the character animation part. During these days is when I finally realized the amount of detail and hours that developing a professional video game must take. Having the idea in mind and making it real with all the smallest details behaving exactly like the developers want to, is no easy task.

In the final weeks when I was checking all the details of the video game is when I realized that I had developed a simple video game that could be improved and could grow. I had the basic features added to the game and all I needed was to make combinations of them to generate new levels.

Of course there are improvements and I am sure I will be working on this game more time to be able to develop a final version as close as I can to the first original idea that I had.

I have learned how to work in Unity and organize scripts that interact with its engine; I have seen how the character creation and animation works and how hard it is to have everything working perfectly and as intended.

This thesis has been really exciting for me and it has helped me improve my programming skills as well as understanding how important it is to pay attention to the small details, which is where the projects go from good to spectacular.

7 References

[1] "Unity (Game Engine)". En.wikipedia.org. N.p., 2016. Web. 4 Dec. 2016.

[2] "Unity - Learn - Modules". Unity. N.p., 2016. Web. 4 Dec. 2016.

[3] "Unity Shooting Tutorial (Raycast And Prefab)". YouTube. N.p., 2016. Web. 4 Dec. 2016.

[4] "Unity3d - Shoot Projectiles". YouTube. N.p., 2016. Web. 4 Dec. 2016.

[5] "How To Shoot Bullets In Unity In 3Mins!". YouTube. N.p., 2016. Web. 4 Dec. 2016.

[6] "Exploding Fireball Unity 5 Tutorial". YouTube. N.p., 2016. Web. 4 Dec. 2016.

[7] "Unity Tutorial On Explosion". YouTube. N.p., 2016. Web. 4 Dec. 2016.

[8] "Unity 3D How To Make A Magic Ball Particle Effect". YouTube. N.p., 2016. Web. 4 Dec. 2016.

[9] "Breaking Glass - Unity C# Tutorial". YouTube. N.p., 2016. Web. 4 Dec. 2016.

[10] "Unity Breakable Glass". YouTube. N.p., 2016. Web. 4 Dec. 2016.

[11] "[Unity3d] [Tutorial #2] Destructible Objects". YouTube. N.p., 2016. Web. 4 Dec. 2016.

[12] "Unity - Breaking Window Tutorial". YouTube. N.p., 2016. Web. 4 Dec. 2016.

[13] "[Unity3d] Destroying Object On Collision (With Sounds)". YouTube. N.p., 2016. Web. 4 Dec. 2016.

[14] "Mastering Collision - Unity Tutorial". YouTube. N.p., 2016. Web. 4 Dec. 2016.

[15] "Unity 5 - 100% Physically Based Shading - Demo Scene - Full Day/Night - Sunset / Sunrise". YouTube. N.p., 2016. Web. 4 Dec. 2016.

[16] "Mixamo: Quality 3D Character Animation In Minutes". www.mixamo.com. N.p., 2016. Web. 4 Dec. 2016.

[17] "Tutorial: Mixamo And Unity Animation Overview". YouTube. N.p., 2016. Web. 4 Dec. 2016.

[18] Technologies, Unity. "Unity - Manual: Unity Manual". Docs.unity3d.com. N.p., 2016. Web. 4 Dec. 2016.

[19] "Textures.Com". Textures.com. N.p., 2016. Web. 4 Dec. 2016.

[20] "Unity: Now You're Thinking With Components". Game Development Envato Tuts+. N.p., 2016. Web. 4 Dec. 2016.

[21] "Set The Pivot - Interactive 3D Graphics". YouTube. N.p., 2016. Web. 4 Dec. 2016.

[22] "Picking - Interactive 3D Graphics - Youtube". YouTube. N.p., 2016. Web. 4 Dec. 2016.