KINESTHETIC IN A CLASSIC VIDEO GAME

Author:

Néstor Luis Zapata Díez

Tutor:

José Ribelles Miguel

Bachelor's Degree in Video Game Design and Development

Summary

Nowadays, video games have become a fundamental part of entertainment for people of all ages. Every time video game companies invoice more money and can produce bigger and more complete works. However, due to the large number of companies dedicated to the creation of video games, it is inevitable that very similar games are launched on the market through the years. At this point, it is the small details that make the difference between one video game and another. The details that can produce a better immersion of the user and produce a better game feeling are a fundamental part in the popularity of a video game. These elements are called "kinesthetic" and are essential to highlight a video game compared to similar ones and produce a better reaction of the players.

This document presents the technical propose for a TFG in a Video game Dessign and Development degree. This proposal consists in the development of a multiplayer video game for computer platform with classic mechanics like Pong, including different kinesthetic elements and modifications in order to produce an attractive game experience. In order to achieve this goal, this project will include effects making use of shaders, particles, sound effects, post-processing, animations and camera movements in the Unity 3D engine.

Keywords

Video game, Kinesthetic, Game feel, Post-processing, Particles, Shader, Unity 3D.

Index

List of Figures

Chapter 1: Technical Proposal

1.1 Introduction and job motivation

This work was born with the idea of creating a video game that includes kinesthetic elements that provide constant feedback to the player and also be pleasant to play.
The project will start with the development of the game itself, a multiplayer game with similar gameplay to classic video games like Pong **[1]** (see figure 1) or Mario Tennis **[2]**. The main inspiration of this title is the mechanic simplicity and the structure of two players matching each other in a battle of reflexes like the games mentioned. Each player will have its own side of the map and a scenario design based on the use of shaders and colors.

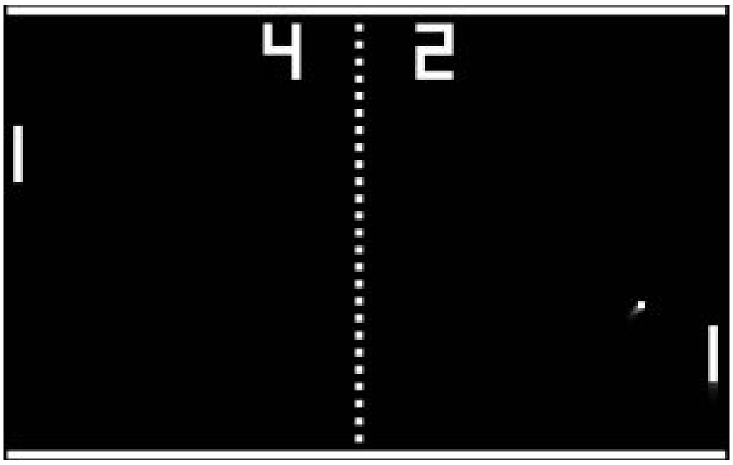

Figure 1.
A frame of the game «Pong» (1972).

After that, the kinesthetic elements will be included. These elements will improve the immersive experience of the player giving feedback to the player and making the game to «feel real». The kinesthetic will be explained in more detail in Chapter 3.
For the kinesthetic implementations, the game will be inspired by games like Lethal League **[3],** which implements a fast gameplay based on matches that uses the ball as a weapon. This video game launched in 2014 implements many kinesthetic elements that will inspire the development of this game like shader efects, camera movements, player movement and flow of the game.

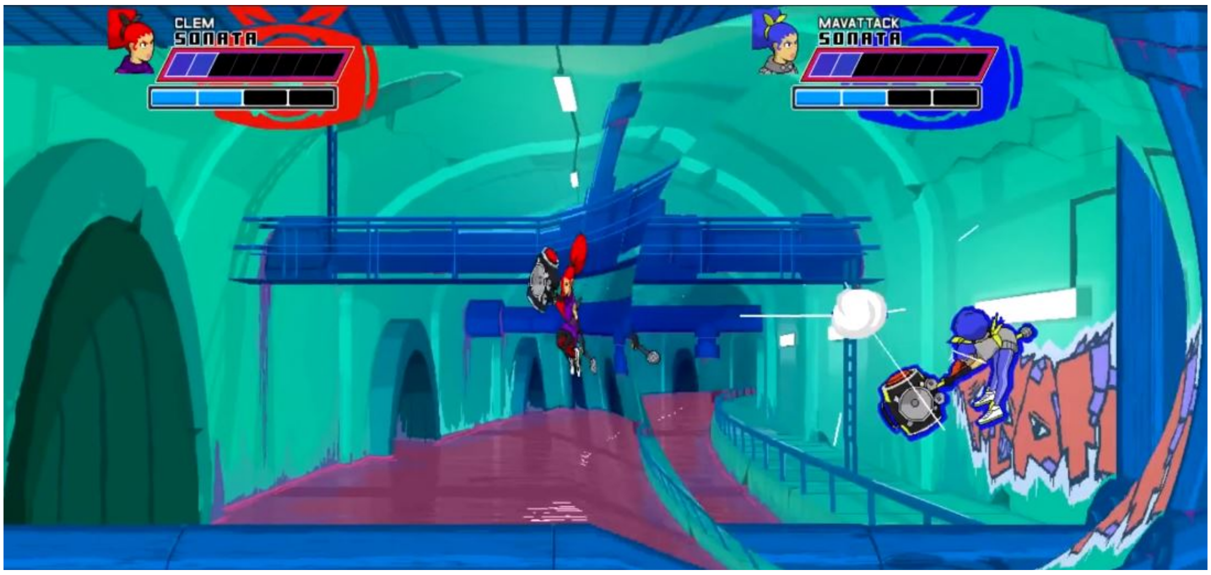

Figure 2.
An example of a shock wave in the game «Lethal League».

In the figure 2 can be seen an expansive wave that appears when one player hits the ball. Also, when certain velocity is reached in the game, more kinesthetic elements appear, like changing the background, as we can see in figure 3.

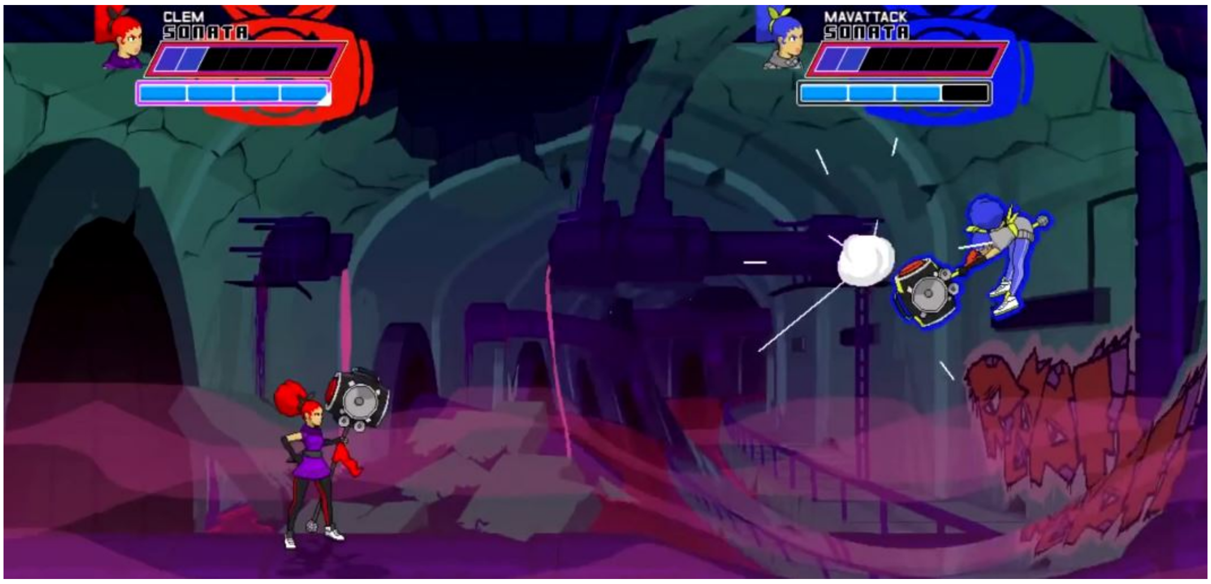

Figure 3.
An example of a different background in «Lethal League».

1.2 Related Subjects

-VJ1221 Computer Graphics
-VJ1222 Video Game Conceptual Design
-VJ1227 Game Engines

1.3 TFG Goals

- Develop a video game inspired by classic gameplay that provide the player with a dynamic and fun gameplay thanks to the kinesthetic elements.

- Implement different kinesthetic elements in Unity making use of shaders, particles, camera movements, sound effects and animations.

- Offer a fun scenario to allow 2 players to compete on ability and reflexes.

1.4 Planning in Tasks and Temporary

Game core	70h
Shader Effects	50h
Camera Effects	30h
Animations Implementation	15h
Audio Implementation	15h
Kinematic Implementation	15h
Particles Implementation	30h
Debugging and Testing	25h
TFG Document	50h

Table 1.
Time ditribution of tasks.

1.5 Expected Results

At the end of this video game development it is expected to have done a fast, fun and enjoyable videgoame with many kinesthetic features that will favor the immersion. Also, it is expected to achieve a smooth and interesting gameplay.

The effects of the game will be representative and easy to understand, giving feedback to the player continually.

1.6 Tools

-Unity 3D
-Shader Graph
-3DsMax
-Audacity

1.7 References

My principal references to do this game are games like “Lethal League”, “Pong” and “Mario Tennis”.
«Pong» is a video game published in 1972 for the Atari platform. This game simulates the

table tennis sport in 2D with basic graphics and simple mechanics.
«Mario Tennis» (figure 4) is a video game published in the year 2000 for the Nintendo 64 platform. This game simulates the Tennis sport in a 3D space.
«Lethal League» is a video game published in 2014 and developed by Team Reptile. This game is a fusion between the mechanics of pong and a typical fighting game, as the player must defeat the opponents using the ball as a weapon.

The main characteristics of these games that inspired the development of this game are these:

- Pong: The simple idea of two fields, one for each player and the main mechanic of return the ball to the other player.
- Mario Tennis: The evolution of the mechanics of Pong in a 3D space.
- Lethal League: The successful implementation of kinesthetic elements to the Pong formula.

Figure 4.
Mario Tennis in-game screenshot.

Chapter 2:
Game Design Document

2.1 Introduction

This project is a local multiplayer 3D game made with Unity 3D where two players can fight each other.

The main purpose of the game is to allow two players to match each other in a fast game with elements of kinesthetic that will allow a continuous feedback between the player and the game. The main mechanic of the game is returning the ball to the other player's field as soon as possible, as the ball explodes when a certain time has passed.

For this reason, the game implements different kinesthetic formulas to make the player feel a feedback of the game at every moment and make the game easy and nice to play, but also challenging.

2.2 Story

This game does not have a story, but an introduction of the playable characters through the initial kinematic in which these two players will see the ball moving and will go after it. If the result of the match is a draw, another scene will be shown, where the players are running again trying to catch the ball, sending the message of the player that there is no winner and the fight must continue.

2.3 Controls.

In order to make the game easy to play and accessible, the game can be played using only one keyboard or only one controller (see figure 5) for the two players.

On the keyboard the controls are these:

- Blue player: WASD for movement and Left Control for the hit action.
- Red player: IJKL for movement and Right Control for the hit action.

On controller the controls are these:

- Blue player: Left stick for movement and Left Bumper for the hit action.
- Red player: Right stick for movement and Right Bumper for the hit action.

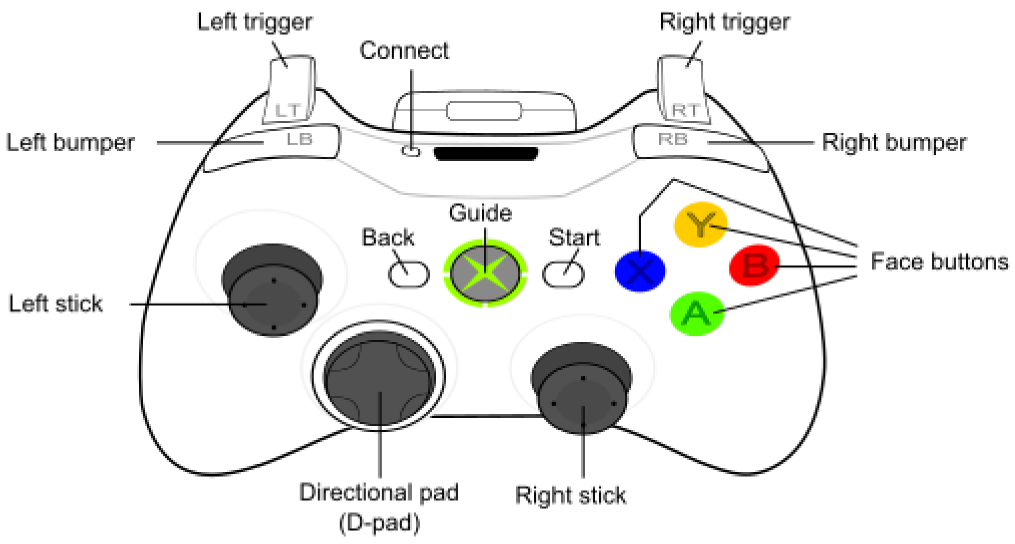

Figure 5.

A PC controller numerated **[A1]**.

2.4 Game Flow

When the game starts will show the players the Main Scene which contains the title of the game with two buttons: the PLAY button and the Exit button.

If the PLAY button is pressed, the game will show an initial cinematic that will present the characters of the game. After that cinematic the main battle scenario is shown, where the two players can move and start a match.

2.5 Game Camera

The camera of the game will be on the lateral of the stage showing all the field, but this camera will not be static all the game because it will be shaking in certain circumstances and it will move at the end of the match.

Also, the camera will make different animations like the movement in the initial cinematic and the general view of the game field at the end of the match.

2.6 HUD

The HUD in this game will be minimal. The intention is not to charge the players with so much information, so the hud will consist in a timer at the top center of the screen and two counters that will show the score of each player.

2.7 Playable Characters

Following the line of «Lethal League» (the game that inspires a big part of this project) the main character of this project will be a 3D modelled character of the game: Candyman (see figure 6). This character will have two different skins that will follow the global aesthetics of the project (see figure 7).

Figure 6.

Candyman, a character of the game «Lethal League».

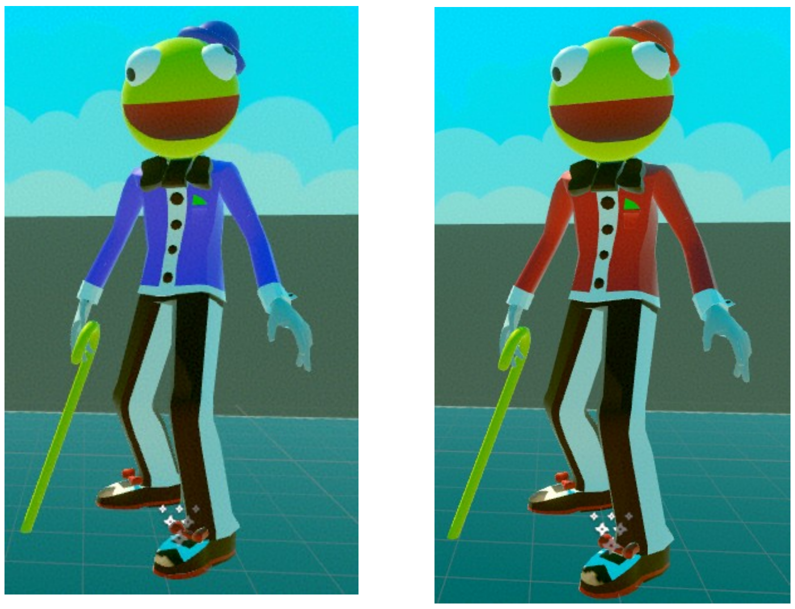

Figure 7.

The playable characters of the project: Player Blue (left) and Player Red (right).

2.8 Combat Mechanics

The game field will be a big terrain divided into two mini fields: one for each player. Each player only will move into its own field.

The match between the two players will start once the global time starts its countdown. Then, one of the players will start the game hitting the ball, which will travel across the field and will enter into the other player's field.

This is the first mechanic of the game: if a player hits the ball, it will go to the other player's field, directly or not (bouncing first into one or more walls).

When the ball enters into a field, it can't exit until that field's player hits it. If the player does not hit the ball, it will bounce between the walls.

Once the ball enters in the other player's field three possibilities can occur:

1. The ball hits the player before the player hits the ball: This is a "dead" for this player.
2. The timer of the bomb reaches 0, so the ball explodes and kills the player in that field.
3. The player on that field hits the ball, making it go into the other player's field, starting again the loop.

Each time that the ball changes of field, the timer will restart, so the intention of this timer is pushing the players to try to hit the ball instead of just avoiding it. It is like a "player turn time". Also, the ball will gain velocity little by little with each hit from the players, making the match harder.

At the end of the global time, the player with a higher score will be the winner of the match.

2.9 Game Modes

There will be only one game mode: the classic match.

This match will be structured like is explained previously.

2.10 Challenges

This game makes the players afford different kind of challenges:

1. Fast reflexes to avoid the ball when is necessary.
2. Fast position prediction to know where the ball will hit and which direction will it follows.
3. Good timing control to hit the ball exactly in the moment it can be hit (not too near or too far of the player).

2.11 Active Abilities

The active abilities of the characters will be:

-Running: The characters will move through the game field running by default.

-Hit: The players have the ability to hit the ball.

2.12 Passive Abilities

There won't be any passive abilities for the characters.

2.13 Health

Each character will have only one health point and this will decrease to 0 when is hit by a ball or an explosion.

Chapter 3: Kinesthetic

3.1 Introduction

As a concept, the kinesthetic (or kinaesthetic) has many different meanings depending of the ambit that we are talking about.

Out of the video games it is defined like a sense, the self perception of the position of the muscles. The capacity of feeling the position of our body without the help of other senses, something very similar to propioception.

In the video game's world, the meaning of this word is more diffuse and, depending the author, it can be defined in many ways. Although, it is related most of the time one way or another with game feel.

Kinesthetic consists in all the different methods than allow the player to gain a «virtual propioception».

Sensations like feeling the weight and momentum of a virtual object with virtual physics, the sensation of toughness when you are facing a boss of one game, the sense of speed in race games or the «fear to fall» when playing a platform game are sensations that allows the player to be immersive in the virtual world and need different elements to be implemented.

There are two ways to make the player be part of these sensations: physical and virtual.

On one hand, we have the physical kinesthetic, that is related to sensations that give feedback to the user in a physical way, like different kind of controllers for specific games, or features like vibration in the controllers. An example of this would be games like «Bass Fishing» and its controller (see figure 8) made for a fishing experience emulation. Also games like «Dance Dance Evolution or «Eye Toy» (see figure 9), that implements different ways of playing.

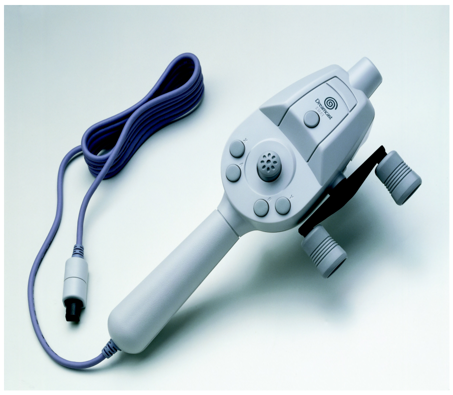

Figure 8.

The video game «Bass Fishing» (left) with its special controller (right).

Figure 9.

Examples of other games that explore the physical kinesthetic: «Dance Dance Revolution» (left) and «EyeToy Play» (right).

Nowadays, the line between physic and the virtual world is vanishing thanks to the creation of new ways to play with technologies like virtual reality and augmented reality (see figure 10).

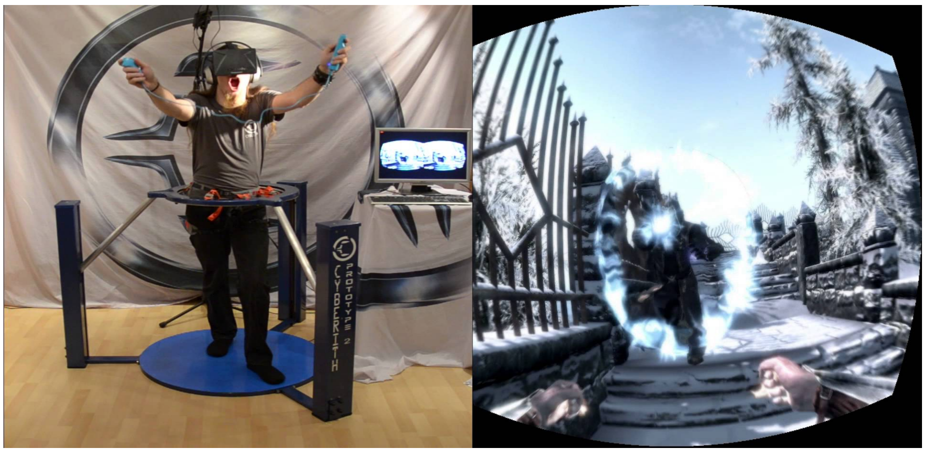

Figure 10.

A new controller for physical kinesthetic (left) and one screenshot of the game played with that controller (right).

On the other hand, we have the virtual kinesthetic, that consists in every method that allows the user to feel part of the game with visual and sound feedback.

Features like the way of moving the character, the animations, the sound effects, the shaking screen effects when doing a hit, the particles, strength waves and changes in the environment are elements that allow a good game feel, giving feedback to the player. This feedback builds a direct relation between the player's actions and the virtual world.

For example, in the figure 11 a view effect can be seen. This effect helps the player to «feel» the speed of the game and feeling inside the game. Also, in the figure 12 another effect can be seen, using particles to make the attack more visual to the player.

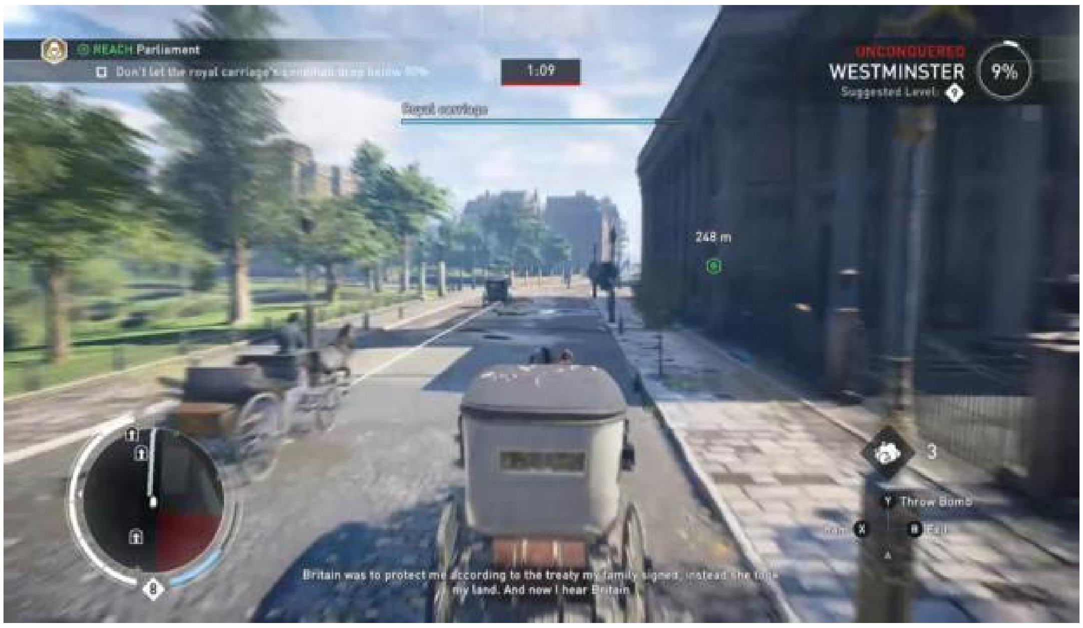
REACH Parliament
Don't let the royal carriage's condition drop below 80%
1:09
Royal carriage
UNCONQUERED
WESTMINSTER
9%
Suggested Level:
248 m
3
Throw Bomb
Britain was to protect me according to the treaty my family signed, instead she took my land. And now I hear Britain

Figure 11.

An example of virtual kinesthetic: View distortion caused by speed in «Assassin's Creed Syndicate».

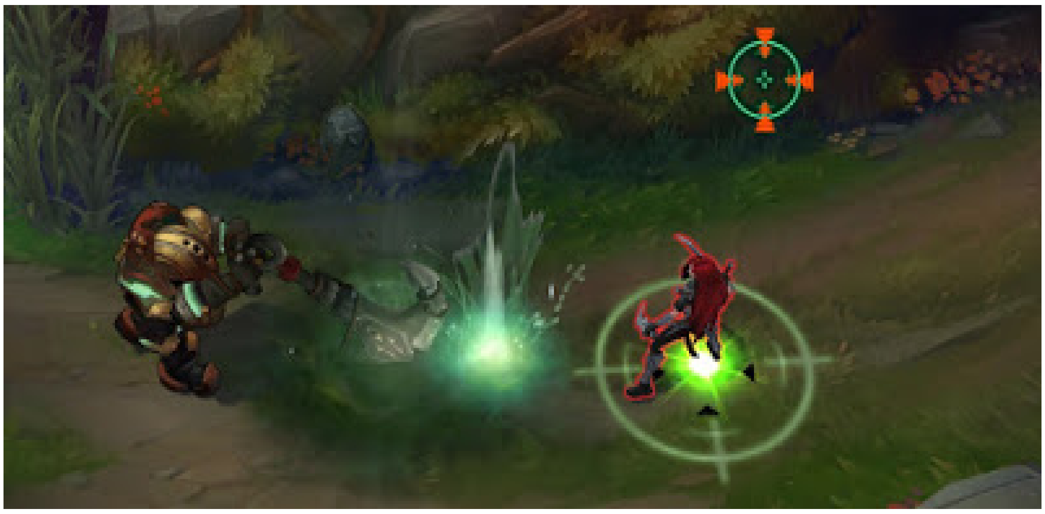

Figure 12.

An example of virtual kinesthetic: Particles caused by the force of an impact in «League Of Legends».

As a conclusion, in a video game the kinesthetic elements are not a fundamental part of the game itself and are not «necessary» for the correct working of the game, but including these elements give the player a better immersive response, makes a feedback loop that marks the difference. It is like driving a car, the driver thinks about the vehicle as an extension of the body. The same with video games, forgetting that the player is using a controller to move a character to «be» this character and moving through it.

3.2 Kinesthetic in this Project

With the goal of giving to the player a good – feeling response of the game, I have researched different effects that fits in my game.

This game includes collisions between a ball with walls and other players. Also, it must be fast, so the effects must complement the action and be agile, not stopping the flow of the game if it is not necessary.

Also, I will use these effects to give coherence to the game. For example, when the ball is placed in its initial position, some particle effects should start and an appropriate sound effect must be played, not just popping the ball in the place. This will also make the experience better and will help to the immersion of the player.

3.3 States of the Game

To understand the different events an how are they related with the feedback to the player, it is necessary to explain which states will be and what events can be found in each one.

The match of the game is divided into these states (see figure 13):

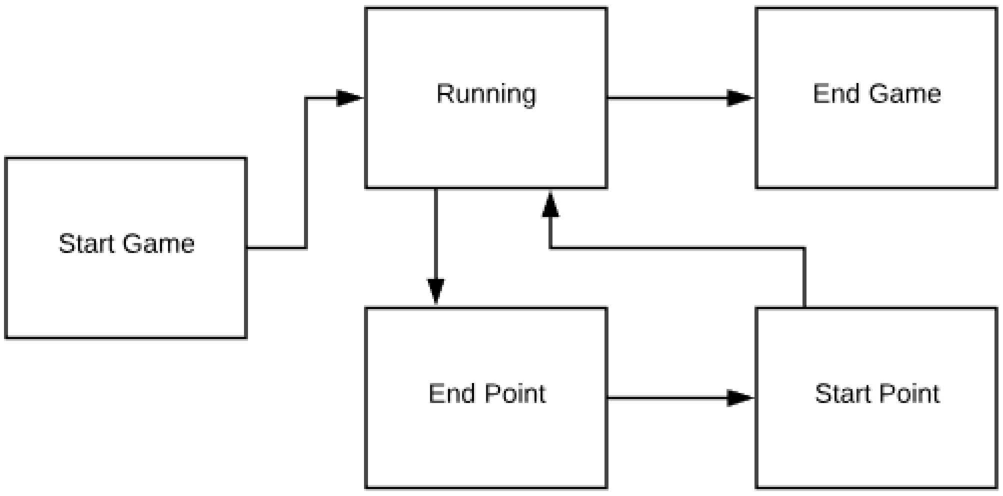

Figure 13.

The transition between the states of the game.

The match starts in the «Not Started» state. This is the initial state when all the GameObjects are placed in their initial positions and the Global Countdown starts running.

This «Not Started» state will be waiting until the player hits the ball to change to «Running». In fact, the global countdown will start at the beginning of this state, although the ball has not been hit yet.

The main state of the game is the «Running», the state in which the game will be played the most part of the match. In this state is where the players fight each other, moving, hitting and avoiding the ball.

During «Running» state, when the ball explodes (by colliding with one player or reaching 0 in its countdown), the match will change to «End Point», that marks the end of one point, when one player's score increase by one.

In «End Point» the ball is moved to one of its starting points. Also, the player that has been hit is moved to its initial position.

Just after «End Point», the game changes to «Start Point», which is a void state that will wait until the player of the field where is placed the ball hits it, starting a new round and changing the state to «Running» again.

Finally, the «End Game» state is the state reached when the Global Countdown has reached 0, the match is over and the game must show the winner, or the scene designed in case of a draw.

3.4 Events in the Game

For each state of the game we can find these events in which we can add kinesthetic elements:

1. Not Started
    - ➔ Place the ball in the initial position.
    - ➔ Place the characters in their initial position.
    - ➔ Start the countdown.
    - ➔ Character moves.
    - ➔ Character hits the air.
    - ➔ Character hits the ball (transition to «Running»).

2. Running
    - ➔ The ball moves.
    - ➔ The ball hits a wall.
    - ➔ The ball hits a player (transition to «End Point»).
    - ➔ The ball's countdown is near to 0.
    - ➔ The ball's countdown reaches 0.
    - ➔ The characters move.
    - ➔ Character hits the ball.
    - ➔ Character does a direct hit.
    - ➔ Character hits the air.

- ➔ Time to change SkyBox (every 10 seconds).
- ➔ Time to change post-processing (accordingly with the night SkyBox).
- ➔ Time to start raining (second 50).
- ➔ Time to end raining (second 80).
- ➔ Global countdown reaches 0 (transition to «End Game»).

3. End Point
   - ➔ The ball disapears.
   - ➔ The ball is placed in one of the starting points.
   - ➔ The defeated player disappears.
   - ➔ The defeated player is placed in one of the starting points.
   - ➔ Character hits the air.
   - ➔ Character moves.
   - ➔ Transition to «Start Point».

4. Start Point
   - ➔ Character hits the ball (transition to «Running»).
   - ➔ Character hits the air.
   - ➔ Character moves.

5. End Game
   - ➔ There is a winner.
   - ➔ The result is a draw.

3.5 Kinesthetic Effects

The kinesthetic effects can be grouped by different types: Camera, Shader, Particles, Sound, SkyBox and Animation.

3.5.1 Camera effects

These kinesthetic effects are in relation to movements of the camera.

3.5.1.1 Camera Shake

Is a relative movement of the camera without changing the direction of the view.

The camera of the game is contained by a «Holder Gameobject». This gameobject will point in the direction we desire for our camera. Then the main camera will be placed inside with coordinates (0, 0, 0) and no rotation. Now the position of the camera inside the holder gameobject is moved by script, returning to (0, 0, 0) at the end of the shake.

With this method we can both shake the camera and moving if (by moving the holder) at the same time and no losing the orientation.

The camera shake is very important for the game feel of any game, because it is a direct feedback of the status of the game.

In this game, it will inform the player when a collision of the ball and other object happens.

The intensity and the duration of the shake varies depending on the event.

3.5.1.2 Soft Shaking

A little shake with short duration and intensity. It will be used when the ball hits a wall.

The collision with a wall is an event that will occur often. For that reason it is necessary to use a soft feedback and no over-charge the player with too much shake when it is not necessary.

3.5.1.3 Medium Shaking

A short duration shake with higher intensity. It will be used when the player hits the ball.

In order to make the player «feel» the impact like a powerful one, it is necessary to make a higher intensity shake. This effect is like the character «makes the camera tremble» because the impact power.

Also, it is necessary to remark the short duration of this shake because probably after the hit the ball will hit soon a wall and a long-time shake could cause an union of the two shakes, resulting in a bad feel because of the abuse of the camera movement and a lack of stability.

3.5.1.4 Hard Shaking

This is the hardest and longest camera shake. It will be used when a round ends, by exploding or hitting a player with the ball.

In contrast with the previous point, when a round ends the game will make a little period of «relaxing time», displaying other effects and later moving each element in their appropriate position in order to start the new round.

Also, it is necessary to give a strong feedback about the end of the round, it is an important moment in the game and the player must really feel a difference with the previous types of shaking.

Taking into consideration these two points, this camera shake has strong intensity shake with longer duration, because the game makes a little «break» and this shake will not occur too near of other camera shake effect.

3.5.1.5 Camera Traveling

A traveling of the camera from its initial position to the front of a player.

This effect is used using the function «Lerp» **[4]** given by Vector3 in Unity. This function interpolates through two positions and can be used for making transitions.

This effect will be displayed at the end of the match, when the global countdown reaches 0.

Also, this effect will only be shown when there is a winner. In case of a draw the game will change of scene.

3.5.2 Particle Systems

Kinesthetic effects in relation with the emission of particles at certain circumstances.

3.5.2.1 Explosion Particles

This particle system emulates a fire explosion with smoke and little pieces in flame flying away.

This effect is created by starting two «son» particle systems, one with the smoke effect and the other with the flying pieces and playing these two systems at the same time.

Also, this particle system is adjusted to decrease the size of the particles over their lifetime, making as a result an effect of «vanishing» an disintegration.

With this effect, we can give an event, like the collision of the ball and the player or the end of the ball's countdown a better feedback and direct information about what is happening. This effect will be displayed at the same time that the ball disappear, achieving the explosion effect.

3.5.2.2 Ball Positioning Particles

This particle system emulates a «canalization of energy» with rays and color smoke.

This effect is created also with two «son» particle systems, one with the smoke effect and another with an emission in the shape of semi-transparent spears.

This effect is used when the ball is placed in one of the start points.

With this effect it seems like the ball is «canalized» in the place, instead of just popping.

There are two almost identical Ball Positioning Particles, one with hot colors and another with cold colors, each one for its current start point.

3.5.2.3 Hit Wall Particles

This effect uses two different particle systems. These particle systems are used when the ball hits a wall.

The first one throws little spears from the point of collision that bounce on the floor. The second one emulates the dust of the wall, an effect that gives the sensation of moving the air when the ball hits the wall.

With these particle systems it is possible to reinforce the feedback given by the camera shake, making a good combination of effects that results in a comprehensive look of the game.

3.5.2.4 Ball Trail Particles

This particle system is used when the ball's countdown is near to 0.

Using this the player can obtain information about the remaining time of the ball without looking at the countdown, so it results in giving to the user more information without leaving the point of vision of the field.

This particle system emits particles of smoke and fire (using two «son» emitters) that will remain for a short period of time in the scene, using the world coordinates, not the local ones of the ball.

3.5.2.5 Field in fire Particles

This particle system is used when the ball's countdown reaches 0.

As the event «Ball's Countdown reaches 0» means a death for one player because the ball explodes, it is useful to make the player feel that all the area of the explosion is affected, so this particle system emulates fire coming from the ground.

Because the position of the camera and the perspective of the game, this particle system is modified to emit in the shape of a cube, using only the sides, not the volume as it is common.

This makes the effect of corners in the smoke of these particles, giving the effect of fire that has been trapped inside the game field.

3.5.2.6 Run Particles

This particle system is used when the characters run and emulates the effect of moving dust while running.

It is used to give the player the feeling that the character is really in the game, that he interacts with the environment and that his actions have consequences.

For this reason, when a character is running, these particles will be emitted, giving the sensation of "moving dust".

These particles are shaped like white cubes. Also to increase the realism, these particles are configured to decrease their size and speed during their lifetime, giving the feeling of being created by the speed of the player and losing strength by the resistance with air.

3.5.2.7 Rain Particles

This particle system is used when one special timer, that is used for managing the weather, reaches its start point. This particle system will stop when this timer reaches its stop point.

In order to give the player the feel of time changing and the pass of the time, these particles will simulate the rain for a period of time during the match.

This particle system is formed by two emitters: one that will emulate the rain and another that will emulate the clouds.

This effect will be complemented with the change of SkyBox and the use of post-processing in order to achieve a better result.

3.5.2.8 Hit Ball Particles

This particle system is used when a player hits the ball. The effect of this particle system emulates the effect of the hit in the comics, doing little white spears at the moment of impact.

This effect is used together with the medium shaking and the hit ball sound.

3.5.3 Shader Effects

The main use of shaders in this project has been to provide an interesting aesthetic. However, its properties have also been used to provide kinesthetic effects to the project.

In order to produce different effects without having problems when painting the final pixel color, each of the effects has been placed in a different layer, so that the most important ones are in an upper layer and cover the lower ones in case there were two effects at the same time.

3.5.3.1 Wave Ground Hit Shader

During the course of the game, when a player hits the ball, an effect will be produced as an expansive wave on the ground that will indicate that the hit has been made successfully. This means that the ball will reach the other side of the stage without bouncing in any wall first and will be called a «direct hit».

This wave is produced by a shader graph that generates transparencies based on a noise function in a shape of an ellipse.

In addition, to keep the aesthetics of the scene, the wave that is performed will maintain the red and blue colors that are used in the rest of the game.

3.5.3.2 Dissolve Shader

This shader is programmed to give an effect of dissolution and composition by modifying an attribute of the material.

This effect will be displayed when a character is defeated, both being hit by the ball or when the ball’s countdown reaches 0, making the dissolve effect.

Also, the opposite effect will be displayed when the character is replaced in its initial position, startin an effect of composition.

This effect gives a better visual impact to the player than just moving suddenly the player between two positions.

3.5.4 Sound Effects

The sounds are also an important part of the kinesthetic elements of a game. Sound can help to build a bigger game immersion. On top of that, sounds give to the user auditory feedback.

The use of an auditory feedback and a visual feedback complements the game experience.

It is necessary to search sound effects easy to comprehend. So easy that a player that is not looking at the screen must have a little idea of what is happening on the screen by hearing the sounds.

3.5.4.1 Hit ball sound

This sound effect will be played when the player hits the ball. It must transmit the feel of a powerful hit.

3.5.4.2 Hit wall sound

This sound effect will be played when the ball hits a wall. It must transmit the feel of the collision with a solid and static element.

3.5.4.3 Hit air sound

This sound effect will be played when the character tries to bat, also if it hits the ball or not.

The sound will resemble the «cut air» effect and must give power and speed sensation to the player.

3.5.4.4 Explosion sound

This sound effect will be played when the ball hits a player and the explosion is produced.

It will also be played when the ball's countdown reaches 0.

The sound will emulate an explosion sound, strong and loud.

3.5.4.5 Ball positioning sound

This sound effect will be played when the ball is placed in its initial position. This sound will complement the particle system that will be played in this event.

This sound must transmit the feel of canalization and concentration of energy.

3.5.4.6 Start game sound

This sound effect will be reproduced only once at the beginning of the match, to tell the players that the match starts.

At the same time that this sound is being played, the global countdown will start running.

This sound must be recognizable for the users, similar to a horn.

3.5.4.7 End game sound

This sound will only be reproduced once in the match, at the end. It will help the player understand that the match is over without looking the countdown.

This sound effect will emulate a thorn too, like the Start Game sound.

3.5.5 Animations

In a videogame it is important to use fluid animations that transmit to the player information about what the character is doing.

It is, therefore, necessary to associate animations with the possible states that the character may have. In this project the states are the following:

3.5.5.1 Idle

Is the base state of the character.

In this state there is no need to add any other effect because it represents the lack of input of the player. In this animation the character only breathes and moves almost nothing.

3.5.5.2 Running

The Running animation will be displayed when a movement input is detected.

With this animation the Run Particles will also be played in order to increase the feedback to the player.

3.5.5.3 Hitting

An animation that makes the character move its stick like a baseball bat.

With this animation the «Hit air sound» will always be displayed, but the «Hit ball sound» will also be reproduced if the character hits the ball.

3.5.5.4 Falling

In this case, there will be two different animations that will depend on the direction of the impact respect the character. Once the game has computed this information, one animation or another will be displayed, making the character fall to the ground «face down» if the impact was from the back or «face up» if the impact was from the front.

In the state «Ball's Countdown Reaches 0», only one of these two animations (face up) will be displayed.

With these animations, many effects will be displayed depending on the situation of the match, like the Explosion Sound, the Explosion Particles or the Field in Fire Particles.

3.5.6 SkyBox

One of the ways to achieve a "time course" effect is by changing the background of the game.

To achieve this effect in the game, 10 SkyBox have been imported and saved in a vector in the SkyBoxManager class. During the course of the game, every certain time interval, a call to this manager will be done in order to change the current SkyBox.

The SkyBox that have been included give the sensation of the course of an entire day: Noon, dusk, night, dawn and day.

3.5.7 Post-processing

In order to create a final scene with the right color and lighting ratio, the use of post-processing is a fundamental tool that allows global adjustments to the final result quickly.

In this project several post-processing profiles will be used depending on whether it is day or night, so that the colors of the scene are coherent and compatible with the lighting.

3.6 Assets

The assets used in this project are as follows:

- Camera effects: Package EZ Camera Shake **[A2]**.
- Particle systems: Cartoon FX **[A3]** and Simple FX **[A4]**.
- Shader Graphs: UnityTechnologies example library **[A5]** and Brackeys shader graph tutorial **[A6]**.
- Animations: Mixamo's webpage **[A7]**.
- SkyBox: Farland Skies collection **[A8][A9]**.

3.7 Summary Table

The following table shows the different events that can occur within the different states of the game and the effects that will be displayed as a consequence. There is also a column in charge of marking the transition to another state because of an event.

State	Event	Effect	Type	Transition
Not Started	Place the ball	Ball positioning particles (Red)	Particle system	
		Ball positioning particles (Blue)	Particle system	
		Ball positioning sound	Sound effect	
	Place the characters	Dissolve shader	Shader effect	
	Start the countdown	Start game sound	Sound effect	
	Time to change skybox	Skybox change	Skybox	
	Time to change post-processing	Post-processing change	Post-processing	
	Time to start raining	Rain particles start	Particle system	
	Time to end Raining	Rain particles end.	Particle system	
	The characters move	Run particles	Particle system	
		Running animation	Animations	
	Character hits the air	Hit air sound	Sound effect	
		Hitting animation	Animations	
	Character hits the ball	Medium shaking	Camera effect	Running
		Hitting animation	Animation	
		Hit ball sound	Sound effect	
		Wave ground hit shader	Shader effect	
		Hit ball particles	Particle system	
Running	Ball moves	None	None	
	Ball hits a wall	Soft shaking	Camera effect	
		Hit wall particles	Particle system	
		Hit wall sound	Sound effect	
	Ball hits a player	Hard shaking	Camera effect	End Point
		Explosion particles	Particle system	
		Explosion sound	Sound effect	
		Falling animation	Animation	
		Dissolve shader	Shader effect	
	Time to change	Skybox change	Skybox	

	skybox			
	Time to change post-processing	Post-processing change	Post-processing	
	Time to start raining	Rain particles start	Particle system	
	Time to end raining	Rain particles end	Particle system	
	Ball's countdown is near to 0	Ball trail particles	Particle effect	
	Ball's countdown reaches 0	Explosion sound	Sound effect	End Point
		Explosion particles	Particle system	
		Field in fire particles	Particle system	
		Falling animation	Animation	
	Global countdown reaches 0	End game sound	Sound effect	End Game
	The characters move	Run particles	Particle system	
		Running animation	Animation	
	Character hits the ball	Medium shaking	Camera effect	
		Hit ball sound	Sound effect	
		Hitting animation	Animation	
		Wave ground hit shader	Shader effect	
		Hit ball particles	Particle system	
	Character hits the air	Hit air sound	Sound effect	
		Hitting animation	Animation	
End Point	Ball disappears	None	None	
	Place the ball	Ball positioning particles (Red)	Particle system	
		Ball positioning particles (Blue)	Particle system	
		Ball positioning sound	Sound effect	
	Place the characters	Dissolve shader	Shader effect	
	The characters move	Run particles	Particle system	
		Running animation	Animation	
	Character hits the air	Hit air sound	Sound effect	
		Hitting animation	Animation	
	Non activity time	None	None	Start Point

Start Point	Time to change skybox	Skybox change	Skybox	
	Time to change post-processing	Post-processing change	Post-processing	
	Time to start raining	Rain particles start	Particle system	
	Time to end raining	Rain particles end.	Particle system	
	The characters move	Run particles	Particle system	
		Running animation	Animation	
	Character hits the air	Hit air sound	Sound effect	
		Hitting animation	Animation	
	Character hits the ball	Medium shaking	Camera effect	Running
		Hitting animation	Animation	
		Hit ball sound	Sound effect	
		Wave ground hit shader	Shader effect	
		Hit ball particles	Particle system	
End Game	Show the winner (if not draw)	Camera traveling	Camera effect	

Table 2.

A summary of the relations between events, effects and states.

Chapter 4: Implementation

4.1 Ball movement

The main idea of the game consists in a ball that will be bouncing constantly between the walls of the scene, gaining speed little by little. The ball must always be at the same height, only moving in its horizontal plane.

The first attempt of making this system was building scenario formed by boxes with box colliders an then moving the ball inside.

A script for the movement of the ball was programmed, which changes its direction when a wall was detected.

At low speeds the system worked well, as expected. The ball bounce correctly between the walls. But when the speed started to raise the ball escaped from the box soon. At high speeds the ball could «jump» the collider.

The size of the box collider was changed, but this only delays the problem, because the ball will increase its speed and this problem will happen sooner or later. Moreover, the size of the colliders was unnecessary big, and this system gives complications with the corners, where there are zones out of the colliders.

Also, this method worked with a ball that never changes its rotation, only was moved around the horizontal plane and this would make more difficult to put some effects in the future, so I started to work in the second system.

This time the ball will always follow its forward direction **[5]**. This direction will always be its advance direction, so to change the direction of the ball it is necessary to change the rotation of the ball, as the instruction for the ball will always be «go forward».

Then, another scene was built, now with planes. I tagged these planes with the tag «wall» (this will make sense later).

A void GameObject linked with the ball was also included. This GameObject will mark at every moment the next position that the ball will have, taking in consideration the direction and the speed of the ball.

What is happening now in the game is that the ball goes forward always in its forward direction. Every frame a Raycast **[6]** is thrown in the forward direction. This Raycast will hit one of the four walls.

Then these data are evaluated:

- The distance between the ball and its next position is computed using Vector3.Distance **[7]**.

-The distance between the ball and the point where the Raycast hit the wall is computed.

Then, these two magnitudes are compared. This can result in one of these two situations:

1. The distance between the ball and the wall is bigger than the distance between the ball and its next position. The ball will continue in the game field and it will no hit the wall yet.

2. The distance between the ball and the wall is smaller than the distance between the ball and its next position. This means that the ball will hit the ball, so we must make some adjustments.

When the ball is going to collide with the wall, the ball is put at the point of contact between the Raycast and the wall.

Also, the ball must be oriented, changing its rotation to let the ball continue moving forward with a new direction in the next frame. For this sake, the function of Vector3.Reflect **[8]** is used. This method uses the normal of the plane and compute the reflected vector of an incident vector. I use this reflected vector as the new direction vector and forward vector of the ball, so I rotate the ball to face this direction.

In the next image (see figure 14) we can see the ball with the 3 vectors: forward vector, normal vector to the plane and reflected vector. Also, the small circle inside the ball is the next position. As the speed in this picture is small, the next position is near.

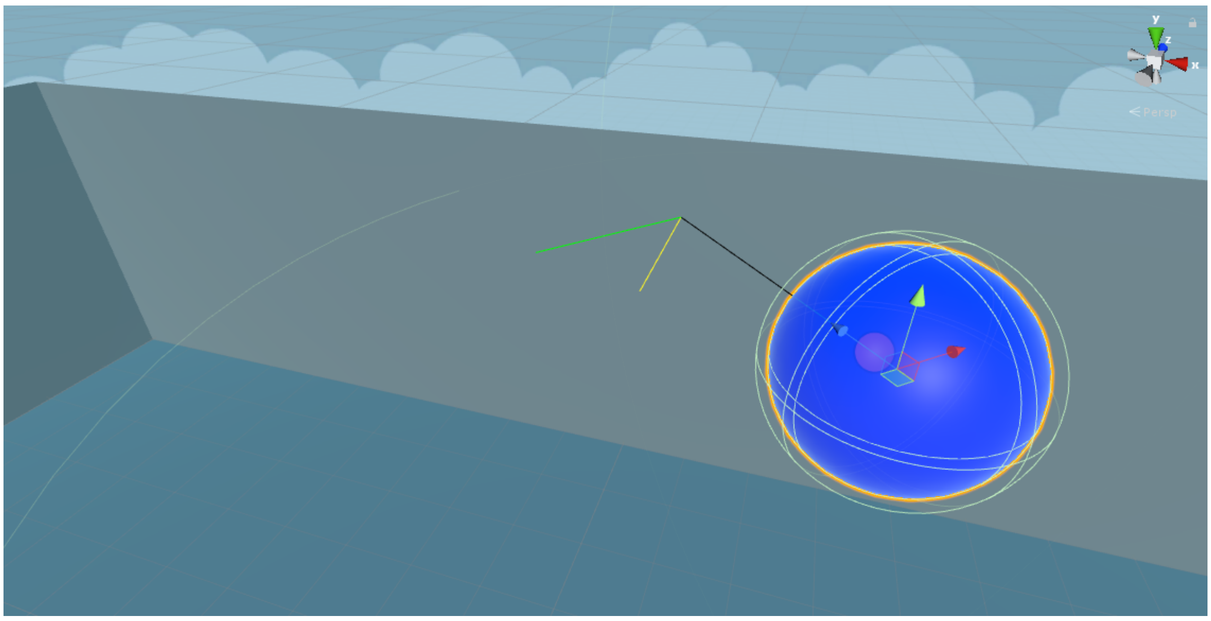

Figure 14.

A frame of an early phase of the game that shows the vectors used in the bounce against a wall and the «next position» point.

In the figure 15 we can see how the next position point is further (as the speed of the ball is bigger) and is out of the field. We can see how in the figure 16 the ball is oriented, placed correctly and the next position of the ball is in the field.

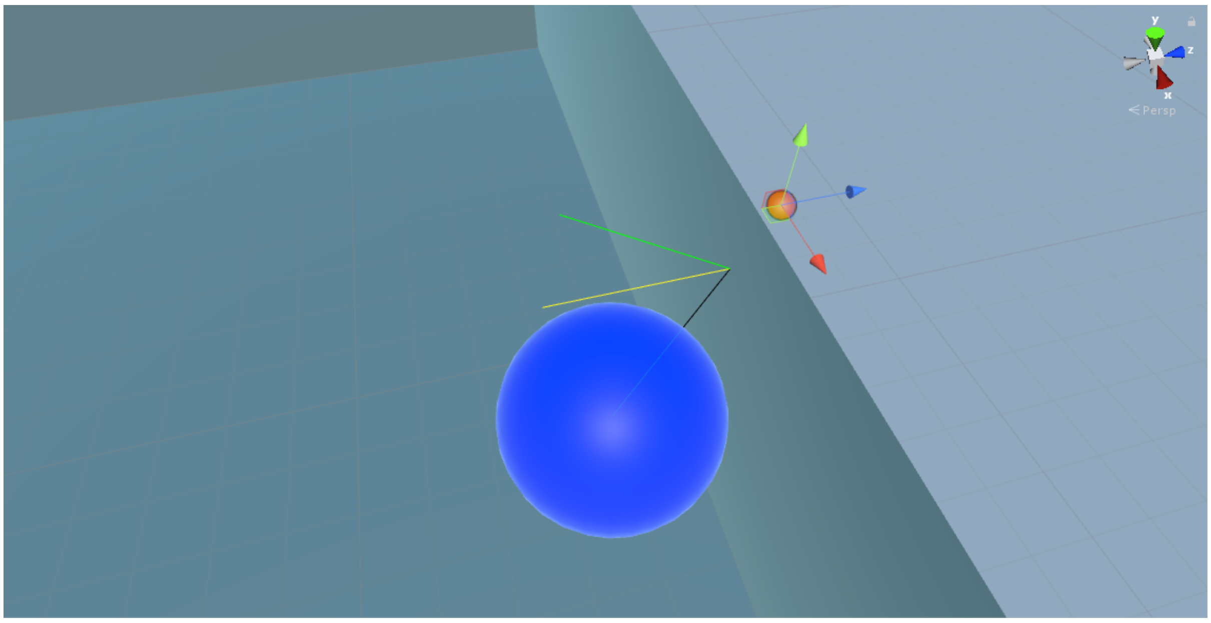

Figure 15.

A frame of an early phase of the game that shows the previous moment of the bounce.

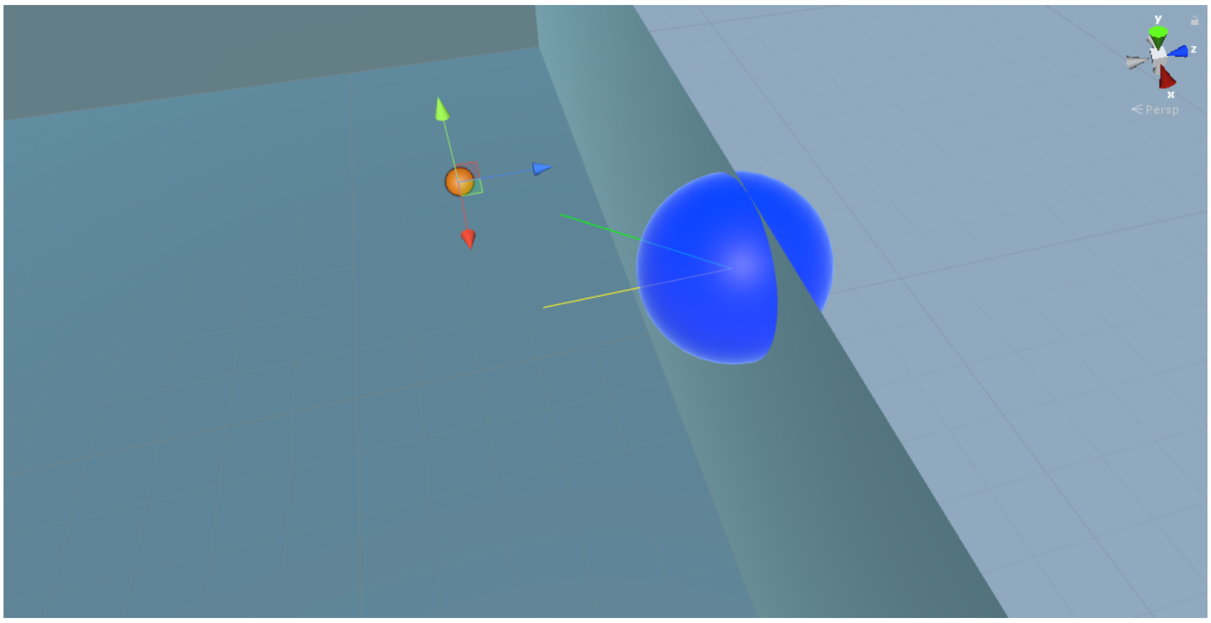

Figure 16.

A frame of an early phase of the game that shows the correct alignment of the ball after the situation of figure 15.

With this second method, the issue with the corners is also solved, making the ball bouncing correctly even at extreme conditions. Now the ball can reach speed levels much higher than the previous system with ease.

Also, the use of the forward vector in the direction of the ball will be used by other functions of the game, like choosing an animation. This will be explained in the next point.

4.2 Character Animations

The characters of the game have different animations that will be shown in the match. These animations are Idle, Running, Hit and the Win animation.

Furthermore, with the aiming of gain a better visual response, I included two different animations for the event «Ball hits a player»: one for a frontal hit and the other for a hit from behind.

To choose between one and another I calculate in the moment of the collision the difference between the forward angle of the player and the forward angle of the ball. With this angle it is possible to choose the correct animation and activate the corresponding trigger of the animation machine.

4.3 Use Of Managers

In order to have the different effects of the game locatable, isolated and easy to modify, empty GameObjects are induced in the game with the necessary scripts and references to act like a manager.

When an effect is needed, the different GameObjects in the game will have access to these managers and will call the adecuate public functions of them.

This way is easy and fast to access the parameters of the effects and modify them if it is necessary.

4.3.1 Particles Manager

It groups all the particle effects of the game, triggering them when it is necessary.

In some cases the particle manager only plays the system, like in the event of positioning the ball at its initial point, because these points are static.

In contrast, sometimes it must move the particles to another position before play them, like in the event «Ball hits player», because this point varies all the time.

4.3.2 Sound Manager

The sound manager has all the sounds of the game and will receive calls of other scripts and play the sound that is needed.

4.3.3 SkyBox Manager

This manager will save in a vector the different SkyBox that will appear during the game.

When this manager receives a call from another script ordering a change, it will increase by 1 the position of the vector that are currently in and load the SkyBox from that position.

This effect in the game will be accompanied by the use of post-processing, which will ensure that the scene is seen correctly at night.

4.4 The speed in the game

The ball of the game not will only go bouncing and being hit during the match without changes, it will increase little by little its speed.

With each hit of one player to the ball, its speed will grow, making every second in the same round hardest than the previous one.

This game mechanic is implemented in order to increase the difficulty curve, avoiding rounds too long or too short.

4.5 Shader Graph

The use of shaders is a powerful tool in the video game development world. Shaders have a lot of potential and can be used in many ways.

Since the beginning of the project, the use of shaders was considered as a fundamental part of the game, making use of them for the environment and also the kinesthetic effects.

Firstly, I started a learning process and a period of adaptation in order to use my shader knowledge in the development of shaders in the engine Unity3D.

But this process was interrupted when I found a new functionality in which Unity's team were working: Shader Graph.

This new functionality was in the beta version of Unity 2018 by the moment of its integration in the project, but the final version was released in June, when the project was updated again.

With shader graphs it is easy to design different shaders fastest, since it allows the user the experimentation of different effects by making little adjustments. Also, it makes easier to modify specific values of a particular effect.

This system works using nodes and allows combination between different characteristics of the material (see figure 17).

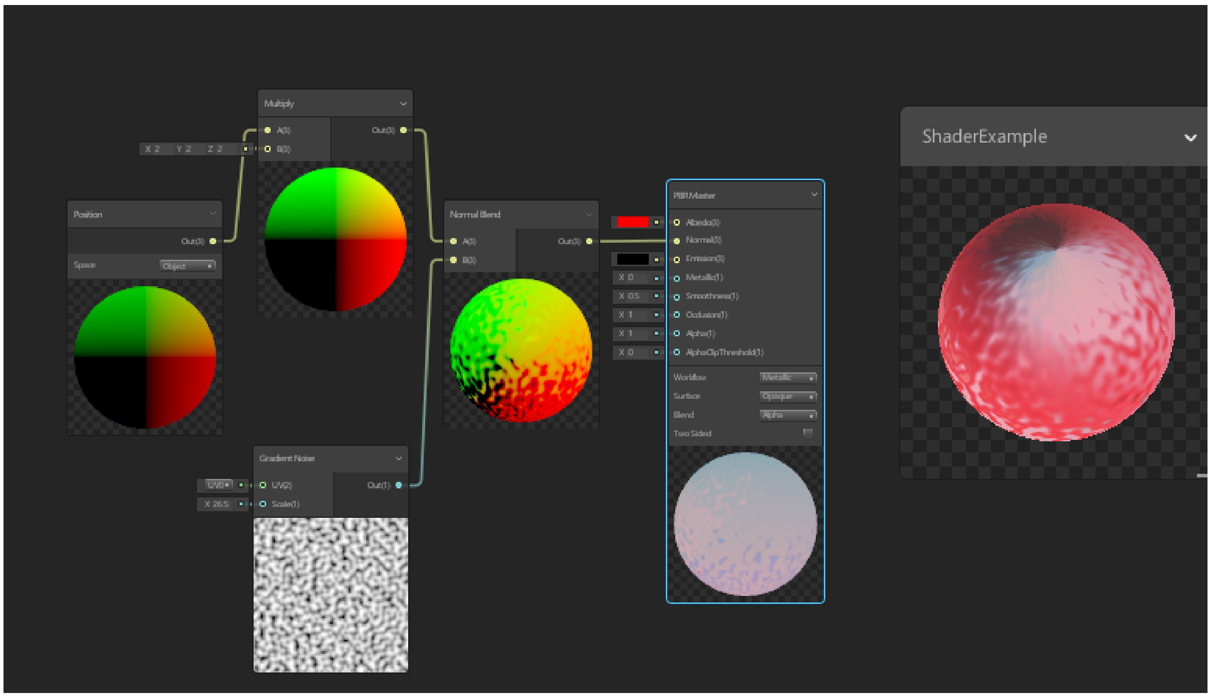

Figure 17.

An example of Shader Graph that gives a rugged effect.

It is possible to create a shader graph and assign it to two different materials. This makes possible to have two materials with similar effects, but with different properties's values (like color or texture tiling and offset), giving as a result a lot of interesting results.

In this project, shader graphs are used for the scene design and as a kinesthetic effect.

4.5.1 Scene design

For the scene design it was necessary to maintain an aesthetic coherence between the player's colors and their side of the map.

4.5.1.1 Wall design

The walls are organised in two different layers in order to give a better result and making the development of the shader graph easier to design.

The first uses a material that emulates a transparent grid of hexagons. These hexagons do not move through the wall, they are static. In order to avoiding dynamism to the scene, the shader of this first wall's material includes a noise function that is responsible for making holes on the surface in a random pattern through time. This gives the walls a «non static» property, that make the scene more dynamic.

Also, for this first layer it is used one shader graph in two different materials, each one with different colors (red and blue).

The second layer is located behind the first. This second layer serves to add color to the field and serves as a base for the first layer. Also, it uses larger hexagons that move slowly, producing small transparencies. It also includes little changes of color in order to produce a hologram effect (see figure 18).

Both the first and the second layer are located and programmed so that the patterns of the hexagons are coherent between the bottom walls and the side walls.

4.5.1.2 Central Wall Design

The central wall uses a material that is managed by the same shader graph as the background walls, but which parameters are modified so that it has a different hologram effect.

It is endowed with tones between red and blue to maintain the colors of the scene. Also includes hexagonal patterns that move faster than the bottom walls and produce transparencies that change in size over time (see figure 18).

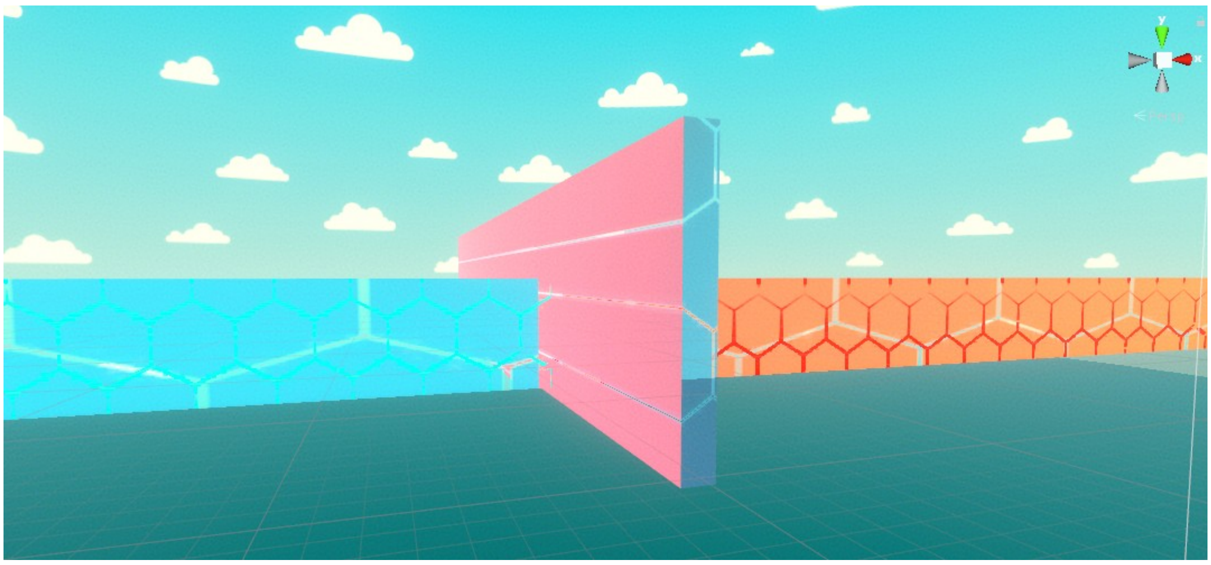

Figure 18.

The final design of the walls in the game.

4.5.2 Initial kinematic

In order to make a kinematic with the presentation of the characters, a shader with an effect of «formation» was needed.

First, there were created some materials with the same colors as the ones that uses the characters of the game. These materials are configured to use a shader that produces a border and leaves the top of that border transparent.

Then, a coroutine is used **[9]** in order to increase little by little the value of the border, making the character appear from the feet to the head.

The materials of these scenes are duplicated for making color adjustments on the border in each character, using blue and red colors.

Also, a camera movement has been developed. The camera turn around of a point of interest which will be moving in order to give a circular view of all the character.

Furthermore, the character will be playing a dance animation that will match with the background music, giving this kinematic a feeling of joy.

Making adjustments in the camera script movement, the kinematic will move through different paths in each scene.

4.5.3 Disolve effect

During the game, when a player is hit by the ball of his field explodes, he performs a «be hit» animation that ends in the ground without being able to move until he is placed back at the starting point.

In order to not produce this transition abruptly, it implements a modification of an existing dissolving shader **[10]** to make the character disappear and appear little by little.

The shader graph allows the election of the dissolve color. The colors are chosen to match with the character ones.

In this way, when a player is on the ground, a dissolution process will start, also carried out by a coroutine, until the character is completely invisible. Then it will be placed at the starting point and will start to «generate» using an opposite coroutine (see figure 19).

This results in a better transition between rounds and the «popping» effect is avoided when the player is placed.

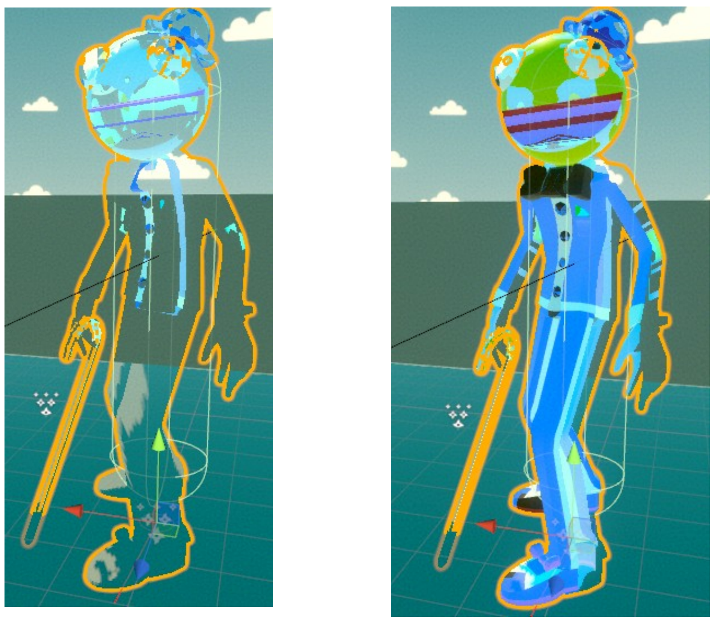

Figure 19.

The dissolve shader making the player appears. The beginning (left) and the end of the process (right).

4.5.4 Ball colors

The first idea was the inclusion of the dissolution effect for the ball at the end of a round too. However, due to the particles of the explosion, this effect is difficult to see. For this reason the shader graph used in the ball will be used for the purpose of highlighting the colors of the ball.

The ball may be using three different materials, which correspond to three situations:

-The ball has not been hit: The game is in the a starting round or starting match state. This situation is represented with the yellow color.

-The ball has been hit by the blue player: It will change its material to a blue one.

-The ball has been hit by the red player: It will change its material to a red one.

These three materials will be managed by one shader graph that will produce a Fresnel effect **[11]** on the ball, together with a light emission effect.

4.6 Post-processing

In order to give a better final aspect of the game, the post-processing functionality of Unity has been included.

Thanks to this functionality, it is possible to make adjustments of color, brightness, saturation and contrast of the final scene in a few steps.

This effect is also included in the initial kinematic, in which this effect adds a slight distortion of the lens, a correction of color to match the color of the player, a noise effect and also a variation of the bloom values.

Furthermore, a hexagonal pattern texture has been added in the kinematic to keep this geometric shape consistent throughout the game.

The post-processing effect is also used through the match when the «night time» starts.

To make consistent the colors and the illumination, another post-processing has been created. This profile gives the scene a dark blue effect and decreases the brightness and the contrast of the scene, making the night more credible.

4.7 Scene Transition

To make the change between scenes in the initial kinematic and the transition to the game scene, a smooth transition between scenes has been implemented, adding an object to the scene that will be in charge of managing this effect.

This is done by animating a black image so that its alpha value increases, making it go from completely invisible to leaving the screen black.

Then the scene change is made for the next one, which will also have an object that allows the change of scenes. This object will produce another fading of the completely black image to be transparent, making the transition smoother.

4.8 Audio Implementation

In order to create an effective kinesthetic in the game, a collection of sound effects that accompany the action of the game have been selected and are complemented with other effects like particles and camera effects.

However, some of the sounds that are used in the game are associated with situations that occur very frequently throughout a game, so listening to them continuously can cause a repetitive feeling to the player.

To avoid this problem there are two possible solutions:

- The first one is to look for a wider collection of sounds and randomly reproduce them when playing a sound effect.
- The second one is the one that has been implemented in this project, the modification of the pitch **[12]** when it comes to playing a sound.

When playing a certain effect, the sound manager implements a function that, before playing a sound, changes its pitch in a range that we have previously delimited in the inspector. This way, with a smaller amount of sound effects, a greater variety of effects are achieved and the repetition of a sound is prevented.

4.9 Limitations

The shader graph is a system that was included in the beta version of Unity 2018 and, although the version is already stable and has been published, the shader graphs have some errors that are difficult to solve until arrangements are made.

One of the main problems I have had is when producing transparencies in the models, because although they are not visible to the camera, the shadows are still calculated as if the objects were visible, producing undesirable effects (see figure 20).

The Unity team reported that these errors are being monitored and are working to solve them, so those visual errors of the project will be solved in subsequent versions of Unity.

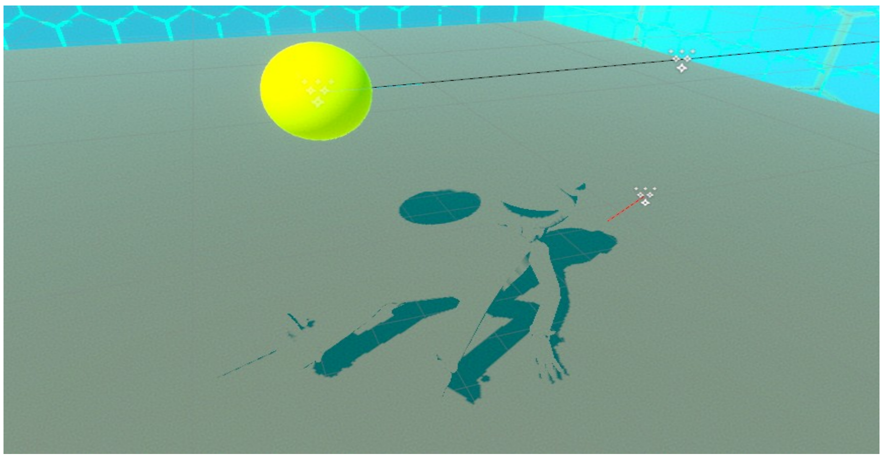

Figure 20.

Example of the errors in the shadow calculation of transparent objects.

Chapter 5: Results

5.1 Code and Executable

The code of the project can be downloaded at the following link:

<https://github.com/al315581/TFG1.git>

The executable of the game can be downloaded at this link:

<https://drive.google.com/file/d/10t3YykV6ixYopCU0jciyfdcP3Se3c9P9/view?usp=sharing>

5.2 Screenshots and Video

- **Figures**

The game has been created following the initial development scheme, developing first a quick game and adding the effects later.

Figure 21 shows the final aspect of the game, where we can appreciate the HUD typography, the colors used, the main SkyBox, the main characters and the effect caused by the shaders on the walls. In addition, this image allows to observe the effect of the particles of the ball in its initial point.

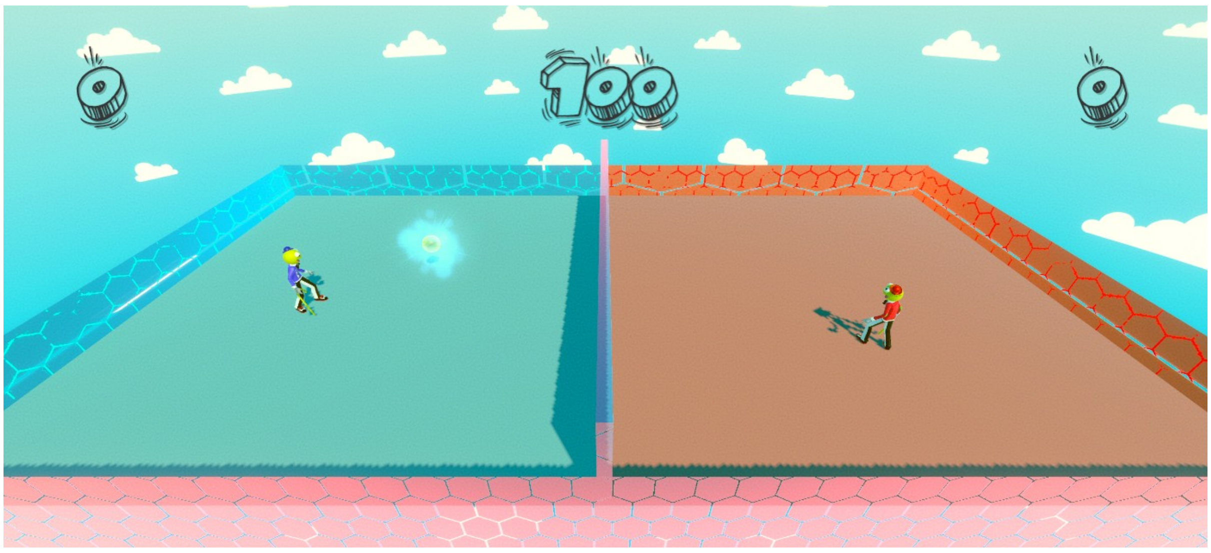

Figure 21.

The final result of the game. Also, the «place the ball» particle effect.

In the figure 22 it can be seen the result of the shader effect that indicates the player that the hit was successful and the ball will go directly into the other player's field. Also, the particle system of «field in fire» can be seen in the right figure, alongside one different SkyBox.

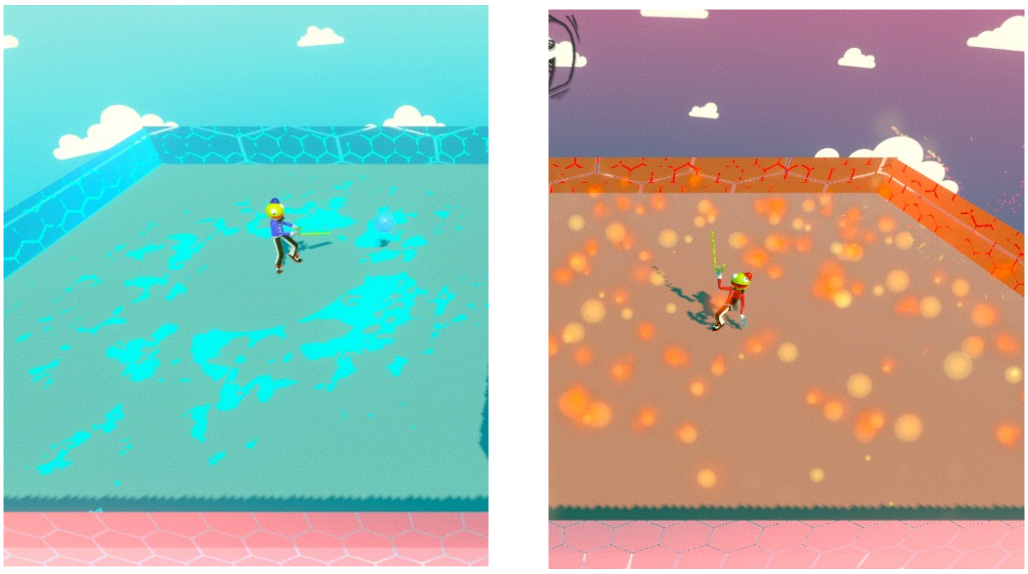

Figure 22.

«Wave ground shader» (left) and «field in fire particles» (right).

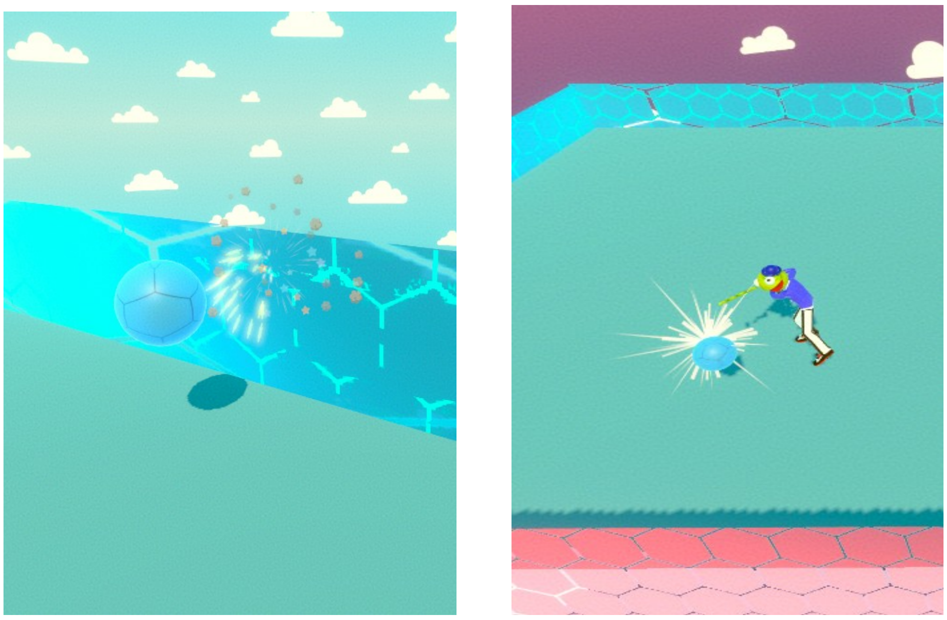

Figure 23.

«Ball hits wall» particles (left) and «player hits ball» particles (right).

In the figure 23 it can be seen the particle systems «ball hits wall» and «player hits ball». Also, the two particle systems in charge of the «ball hits wall» can be differentiated clearly (the spears and the smoke).

The figure 24 shows the previous moment of the collision between the ball and the player in the left figure. In this figure it can also be seen the «ball trail particles». The right figure is a frame of the explosion particle system played when the ball hits a player.

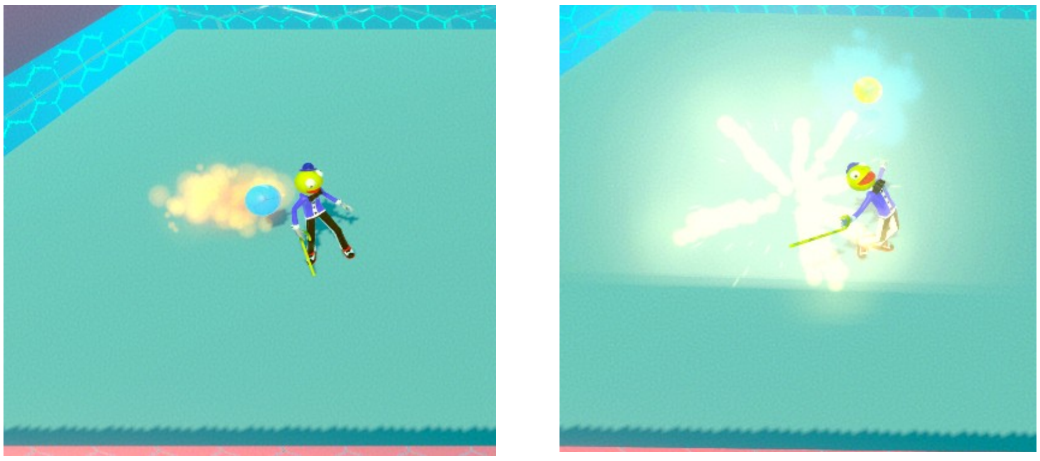

Figure 24.

The «ball trail particles» (left) and the «ball hits player» particles.

In the figure 25 the rain particles can be appreciated. Also, the night post-processing is active, changing the colors of the game.

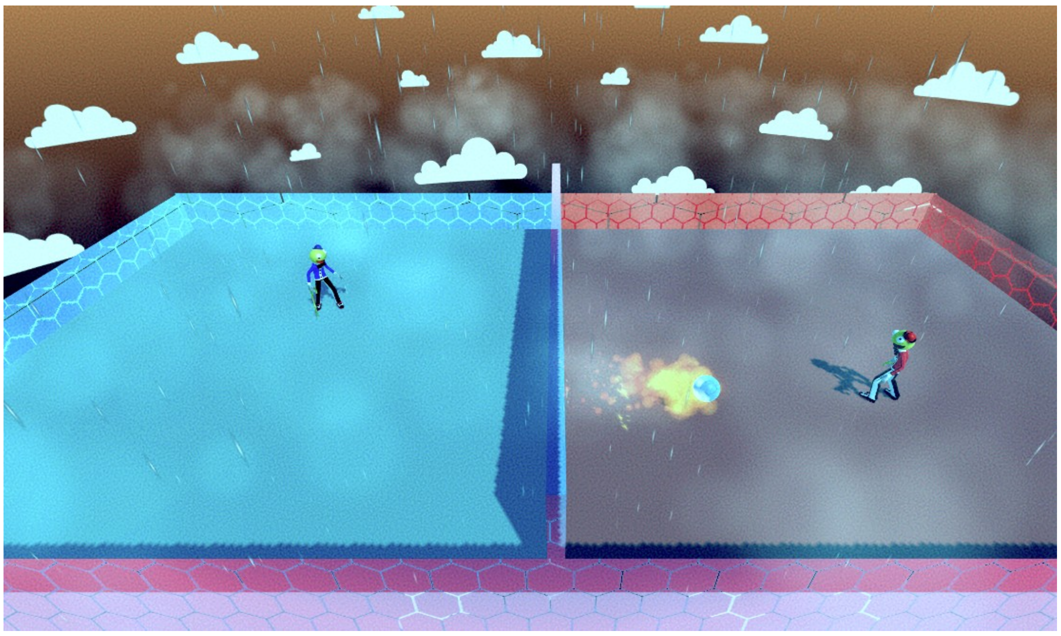

Figure 25.

Rain particles and night post-processing.

The figure 26 shows a frame of the initial kinematic and finally, the figure 27 shows the title scene of the game.

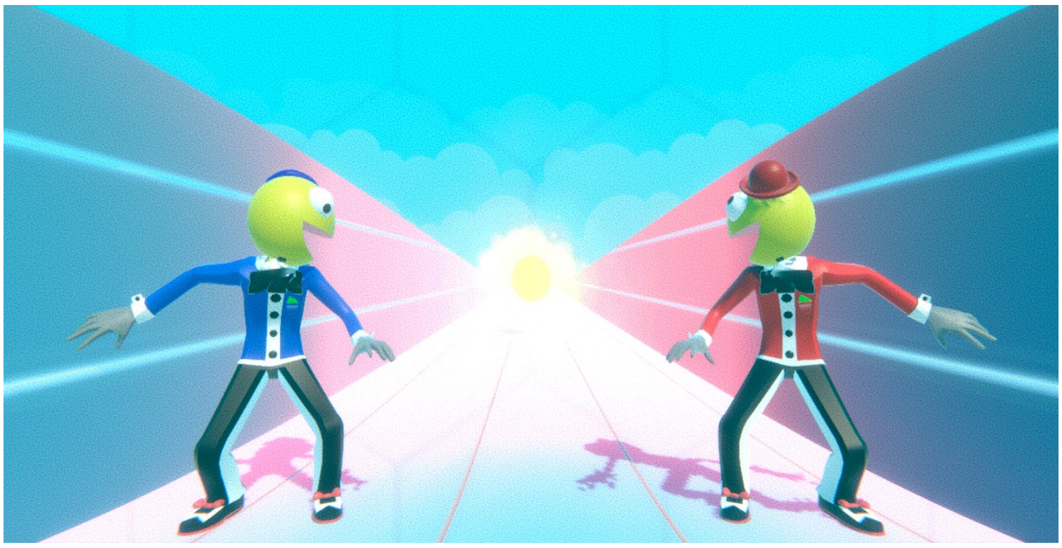

Figure 26.

A frame of the initial kinematic.

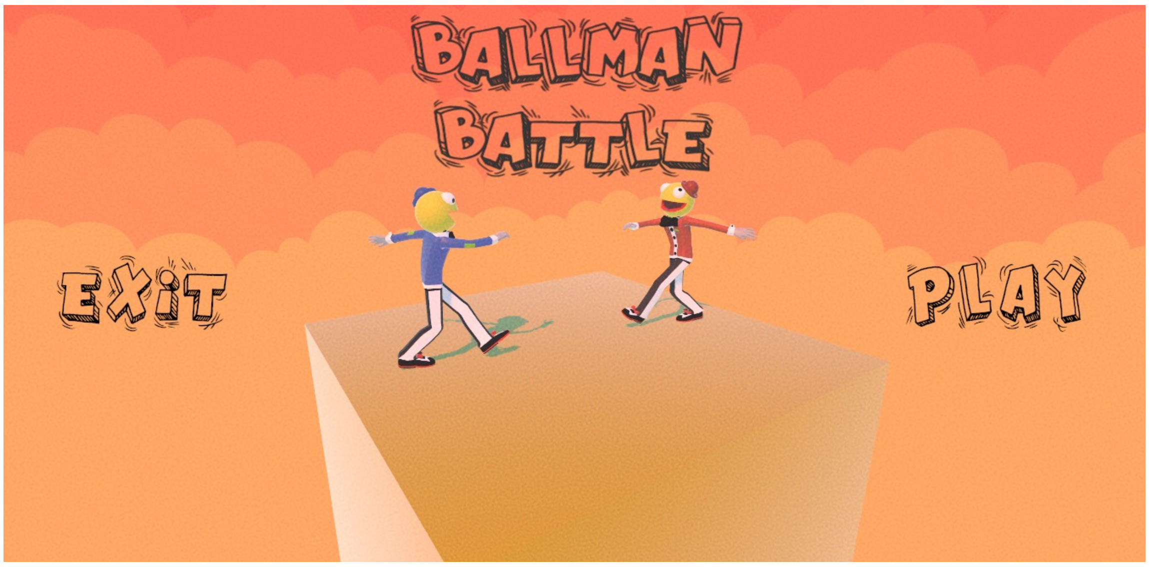

Figure 27.

The title scene of the game.

- **Video**

The video showing the gameplay, mechanics and kinesthetic elements can be seen in this link:

<https://youtu.be/2BVJIH-vq-8>

Chapter 6: Conclusion

In this section of the document the project's objectives are going to be evaluated in order to discuss if they have been successfully completed, as well as future extensions that could be included in this project.

6.1 Project Objectives

Once the project has been finalized, it is time to analyze which of the main objectives that were marked at the beginning of the development have been completed:

- Develop a video game inspired by classic gameplay that provide the player with a dynamic and fun gameplay thanks to the kinesthetic elements.

  The control system and the structure of the game were initially designed with the main idea of providing a fun game full of kinesthetic elements. These objectives have been completed successfully, as well as having expanded the collection of effects during the development of the project itself.

- Implement different kinesthetic elements in Unity making use of shaders, particles, camera movements, sound effects and animations.

  Through the search for information and continuous learning, a large number of effects have been developed, making use of shaders, particles, camera movements, sound effects, animations, post-processing and SkyBox modifications.
  In addition, new effects have emerged that were not part of the main design, but were later included, such as the inclusion of wall shaders.

- Offer a fun scenario to allow 2 players to compete on ability and reflexes.

  This objective has been a fundamental part of the development of the game and was the first objective to be fulfilled, since first the game space was designed in a basic way and little by little it was completed by adding effects until it became what it is now.

6.2 Future Work

The study and implementation of the effects of kinesthetics in a video game has turned out to be a topic that is becoming more interesting as you learn more about it.
During the development of this project, it has been possible to discover «small details» that video games hide and are not properly appreciated by the players.

It is important to highlight the importance of the kinesthetic elements in the final result of a video game, because they convey a pleasant sensation when experiencing them. Also a video game can be monotonous and less fun if these elements are not incorporated correctly.

For this reason, this project could evolve even further by learning new forms of kinesthetics:

1) Virtual:
    - Through the study of new ways of transmitting feedback to the user.
    - Through a more detailed study of the animations of the characters.
    - Through the implementation of different shaders.

2) Physic:
    - Through new ways of control the game, making use of special controls.
    - Through the use of virtual reality glasses.

Thanks to the study of the kinesthetic elements of a video game, it is evident that each field that composes it is very wide and could easily result in a thesis by itself.

Chapter 7: References

[1] Pong Gameplay,

<https://www.youtube.com/watch?v=fiShX2pTz9A> (last visited February 2018).

[2] Mario Tennis Gameplay,

<https://www.youtube.com/watch?v=HRgdH0s2k-Y> (last visited February 2018).

[3] Lethal League's Trailer,

<https://www.youtube.com/watch?v=j78X5LIuM5U> (last visited February 2018).

[4] Lerp Function,

<https://docs.unity3d.com/ScriptReference/Vector3.Lerp.html> (last visited March 2018).

[5] Forward Vector,

<https://docs.unity3d.com/ScriptReference/Transform-forward.html> (last visited March 2018).

[6] Raycast in Unity,

<https://docs.unity3d.com/ScriptReference/Physics.Raycast.html> (last visited March 2018).

[7] Vector3.Distance,

<https://docs.unity3d.com/ScriptReference/Vector3.Distance.html> (last visited March 2018).

[8] Vector3.Reflect,

<https://docs.unity3d.com/ScriptReference/Vector3.Reflect.html> (last visited March 2018).

[9] Coroutines,

<https://docs.unity3d.com/Manual/Coroutines.html> (last visited February 2018).

[10] Brackey's GitHub,

<https://github.com/Brackeys/Shader-Graph-Tutorials> (last visited April 2018).

[11] Fresnel Effect,

<https://docs.unity3d.com/Manual/StandardShaderFresnel.html> (last visited April 2018).

[12] Audio pitch,

<https://docs.unity3d.com/ScriptReference/AudioSource-pitch.html> (last visited April 2018).

Assets

[A1] Controller Figure,

<https://es.m.wikipedia.org/wiki/Archivo:360_controller.svg> (last visited June 2018).

[A2] EZ Camera Shake,

<https://assetstore.unity.com/packages/tools/camera/ez-camera-shake-33148> (last visited April 2018).

[A3] Cartoon FX,

<https://assetstore.unity.com/packages/vfx/particles/cartoon-fx-free-109565> (last visited April 2018).

[A4] Simple FX,

<https://assetstore.unity.com/packages/vfx/particles/simple-fx-cartoon-particles-67834> (last visited April 2018).

[A5] Unity Shader Graph Example,

<https://github.com/UnityTechnologies/ShaderGraph_ExampleLibrary> (last visited June 2018).

[A6] Brackeys Shader Graphics,

<https://github.com/Brackeys/Shader-Graph-Tutorials> (last visited June 2018).

[A7] Mixamo's Webpage,

<https://www.mixamo.com/#/> (last visited June 2018).

[A8] Farland Skies simple,

<https://assetstore.unity.com/packages/2d/textures-materials/sky/farland-skies-simple-cumulus-62565> (last visited April 2018).

[A9] Farland Skies cloudy,

<https://assetstore.unity.com/packages/2d/textures-materials/sky/farland-skies-cloudy-crown-60004> (last visited March 2018).