Ride or Die: VR

Author:
Jon Andoni Fernández Moyano

Tutor of the Project:
José Vicente Martí Aviles

VJ1241 - Final Degree Project

Video Game Design and Development

Universitat Jaume 1

1. Introduction

This is the design and technical document for Ride or Die VR, a video game developed for the Android platform with Unity's graphics engine. The goal of this document is to capture all the elements included in the game: mechanics, planning, flow diagrams, work development, conclusions, etc...

Author of the Project: Jon Andoni Fernández Moyano

Title: Ride or Die VR

Genre:

- ***Sport*:** Because it takes the player to play a simulation of a sport, where he will be able to go at full speed, jump great heights and do many tricks.

- ***Ability*:** This factor is given because the player will have to dominate the controls of the game, to be able to do a good score as well as to be able to jump between platforms, to take well the curves, etc.

***Keywords*:** *Virtual reality, immersive video game, sports, provoking sensations, height, speed*

1.1. Work Motivation

The motivation for the realization of this project arose when I had the idea in my head to adapt conventional skate games to Virtual Reality. Inside my head the idea was quite developed and although there are some existing similar projects, not from the perspective that I want to give to the project. I want to give a pure experience, in first person camera, where the animations submerge the player and make him feel the sensations he can experience with these sports in real life through a video game. The idea of skateboarding was on my mind for quite some time, but I thought that adding Snow and Long tracks would allow me to play with mechanics and add diversity to them.

1.2. Objectives

The game raises the following main points to accomplish:

- *Virtual reality experience*: One of the primary objectives of the project will be that the game is capable of reproducing itself with virtual reality technology.

- *Get an immersive experience*: This project will try above all to achieve a casual experience that makes the player feel within a virtual world where he can practice risk sports without endangering his life.

- *User-friendly approach*: It's not a game that focuses on telling you a story, just playing. To do this, the player must be introduced to the controls and objectives of the game in a clear and simple manner.

- *Gamification of experience*: The objectives within the game will need the skill of the user to obtain the highest possible score. This game will have an online score to allow players to compete and create a more addictive experience. The initial approach is to create a casual experience, but this way seeks to promote the rejugability of the title.

- *Diversity of situations:* Only being able to play on the same stages and with the same discipline can be boring. To this end it is necessary to develop several scenarios that give rise to different situations thus avoiding that the player gets tired of always playing the same thing.

- *Reach the greatest number of players as possible:* There are many people who with VR technology have experienced dizziness and nausea. My goal is to detect what features force this problem and try to solve it in the best possible way.

- *Optimization for smartphones:* The project should tend to be as realistic as possible despite the limitations of the hardware. It is designed to run on a mid-range device so the Assets of the game will have to be adapted to the system requirements.

1.3. Environment and Initial State

For the realization of the project I have had availability of a quiet and extensive workplace where you can dispose of a computer, two screens and an extra space to write in notes that help the project in development..

At the beginning of the subject that gives name to this project I had a small version of the physics that was going to have the movement of the Player in-game. However, this version was not well structured and it was difficult to decouple the methods to apply them to the different disciplines (Snow, Long or Skate). The intention then is to remake the classes from the beginning with the knowledge and references of the old project and in this way make it more easily adaptable and receptive to new mechanics.

1.4. Involved Subjects

For the realization of the project I have recurred to different subjects that I have realized throughout the *Degree of Design and Development of Video games* to help the development of the project.

•***Physics***: because the physics offered by Unity weren't powerful enough to bring my game to life, specific physics have to be developed and this requires knowledge, especially mechanical physics.

•***Algorithms and databases:*** I have applied the knowledge of this subject when developing the internal structure of the application.

• ***Graphic expression:*** since the development of scenarios is at my own cost, I have resorted to the teachings of the subject.

•***Operative System***: In order to optimize the project I have made use of some lessons given in the subject.

•***Game Engines:*** Since we learned to use Unity in this course, it is the groundwork I needed to achieve my goal.

Having the help of this knowledge has improved the final result.

1.5. Tools to use

- **Blender**

This tool is used for 3D game art. In this tool, 3D models are made for Boards, Skateparks, fences, trees, obstacles, etc. The idea is to give a realistic look so that materials will be applied to the objects to give the optimum effect.

UV Maps will also be applied so that the texture drawings fit perfectly to the 3D model that is to be represented.

- **Photoshop**

In this software the different textures that will be maked for the 3D models are created, as well as modifying them in tone, brightness and contrast for their correct application.

The drawings that have been drawn "by hand" have been edited with a digital tablet, a specific hardware that helps in this type of tasks.

- **Audacity:**

This program has been used to record the audios for the game as well as its modification so that the adaptation to the project is as correct as possible and that it works as it should.

- **Unity**

This is undoubtedly the core of the project as it is the engine that will give life to the project. In it will be created the scenes that will include the 3D models, the scripts that execute the mechanics of the game will be compiled, etc.

A large part of the functionalities is used for the development of the game, since the game engine offers a series of tools that are of great help for the elaboration of the project.

It should also be noted that through this software some materials are also modified so that 3D models can be seen in a more realistic way.

Figure 2.1: Planning chart

2. Planning

In this section we will explain the planning of the project over the time available, the results obtained and the conclusions I have reached.

2.1. Initial Planning

Orange → Documentation tasks

Green → Design Tasks

Blue → Development Tasks

- ***Documentation Tasks → 40 hours:***
  - Technical proposal: 5 hours
  - GDD Document: 10 hours
  - Final Report: 25 hours

- ***Design Tasks → 121 hours:***
  - Identification and formulation of game physics: 43 hours
  - Design of 3D models of the vehicles: 12 hours
  - Texture mapping of the models: 11.5 hours
  - SkateBoard Stage Design: 11 hours
  - SnowBoard Stage Design: 10 hours
  - LongBoard Stage Design: 10 hours
  - HUD of the different levels of the game: 4.5 hours
  - Stage lighting: 9 hours
  - Main menu design: 10 hours

- ***Development Tasks → 139 hours:***
  - Implementation of in-game scenarios: 14.5 hours
  - Placement of other objects within scenes: 7 hours
  - Implementation of vehicle physics: 15 hours
  - Programming of the different game mechanics: 20 hours
  - Adding effects: 9 hours
  - Correct positioning of the cameras on the screen: 9 hours
  - Stage development Skateboard: 10 hours
  - Stage development Snowboard: 20 hours
  - Stage development Longboard: 20 hours
  - Development of HUD in-game: 10 hours
  - Main menu development: 4.5 hours

Total → 300 hours

2.2. Deviations from initial planning

I have tried to take very seriously the planning firmly to arrive at the times indicated by the FDP subject. However, due to different events that have occurred the planning has had to be varied as the project progressed.

These delays in the initial programming have been due to two major reasons: problems in the development of certain mechanics and the revision of documents.

The first one was probably the one that delayed my work the most. The development of the Skate was one of the simplest as there was no need to program anything related to the scenarios. However, being the first is the one that has cost me the most time to develop because it marks the base that will use in the other two modalities. In the Snow modality, the design task was one of the most expensive together with Long's one. This was due to the fact that I had to think very carefully how to implement random scenarios, which was not an easy task. It must also be taken into account that the implementation of the Snow and Long scenarios were different, so content could not be recycled between modalities.

As a consequence of these delays in the development of the modalities, the quality of appearance in the scenarios is lower. This is a resounding truth because with more time available I could have left more beautiful experience, however, the priority of mechanical development was higher.

The second thing that has delayed me the most and that has gone beyond the initial planning has been the development of technical documents, more specifically: in the final memory.

The preparation of each of the defined technical documents was carried out on the dates indicated in the table (Fig 2.1). However, the final report was delayed due to corrections that had to be made. This has made the preparation of these documents longer than desired, without taking away other tasks but making extra hours of work. Yes, it may have taken a little time from the final review of the project, but the version was left so polished that it did not affect the final result.

3. System analysis and Design

3.1. Requirements analisys:

- *Functional requirements:*
  - Create real physics and mechanics for skateboarding:

    - In this mode the player will be placed on a skateboard track, in a previously defined position. Along the course there will be a series of fences located in such a way that they can be made following a certain route, although the player will have the freedom to travel as desired.
    - Inside the object "fence" is placed a light that illuminates in a different way marking if the player has already passed through it, the light will be red if it has not yet jumped and green if it has already passed through it.
    - When the player is in the air, he can perform various acrobatic animations to accumulate points during the game.
    - If the player touches the ground before the animation has finished and has returned to its iddle state, he will fall and lose the acrobatics score.
    - The game also features a free mode in which the player can roam the track freely making endless points. Technically, it only differs from the previously detailed mode in which fences and time trial are removed.
    - If the player stumbles due to a failed acrobatics, he will rise at a checkpoint, which are positioned to the extent of the virtual map.

  - Create real physics and mechanics for snowboarding:
    - This game mode will feature a snow descent. This course, in terms of design, is made up of random " pieces " with different obstacles and ramps. These " pieces " are predefined prefabs that are taken by the scenario manager and instantiated when needed. This results in random scenarios.
    - The variable that gives rotation speed to the board is small so if the player tries to rotate it will be difficult to go from one edge to the other. To spin faster, press the button that activates the Boolean "Assisted Rotation", then the function that gives speed to the board will decrease and the one that gives speed to the spin will increase.
    - If the player has activated the "Assisted Rotation" and generates a turn higher than 45 degrees (variable that is saved in the player) and removes the button that activated the Boolean, making it become false, it will be fired adding a certain amount to the speed variable. That acceleration added to the board will be reduced over time by returning it to the original speed.

- As long as the player does not slide, the player receives two types of speeds. A constant speed that makes the player go constantly down the slope and another that will allow going faster in the descent or turn to the sides. This last speed works in a peculiar way. If the board looks directly towards the descent that second speed will be maximum, but if it looks to the sides it will reduce a certain amount but allowing you to move to the sides.
- The player will not be able to go backwards on the track as the variables that give speed are set in such a way that he cannot change the direction of the speed..
- Every impulse the player manages to generate will give a certain reward in the form of points. The impulses can also be generated if the player jumps one of the ramps scattered around the field, these will give twice the score of the impulse.
- If the player tries to get out of the field or hits a tree, he will fall and his position will be reset to the middle position of the field he was crossing.

- Create real physics and mechanics for Longboard:
  - This game mode has a similar structure to the previous one. It is created randomly with different prefabricated pieces of road.
  - The Longboard can slide if you activate the "RotationAssisted" boolean, which will reduce the value of the speed variable and increase the one that determines the speed of rotation..
  - If the Long does not have the "Assisted Rotation" variable activated, it will have a low rotational speed. Also, the speed variable will increase as a function of time, with no maximum limit, so if you don't brake your speed will increase to infinity. Obviously it will not reach a value that determines an unrealistic experience since the road is made in such a way that the player must obligatorily use the "assisted rotation", otherwise he will fall.
  - Along the road there will be different obstacles that will hinder the player's journey. To cross them you must pass between them by pressing the acrobatics button and adding points to the score. The locations of the obstacles are defined beforehand.
  - If the player tries to reverse he will be redirected to the centre. The way to achieve this effect is different from snowboarding but similar. To achieve the same effect, along the "pieces" of road are established some points and colliders "Checkpoints" responsible for defining the direction that the player should take at all times.

        - If the player unsuccessfully crosses an obstacle or leaves the road, the obstacle will fall and be redirected to the mid-point of the road he is currently traveling.

    - Controlling the player's Input through a controller:
        - Use the Unity tool to identify and use input buttons.

    - Create an interactive and Virtual Reality interface:
        - Make good use of the space, as the player can rotate on himself 360º.
        - That the look of the player determines the option that he is going to choose.

- *Non-functional requirements:*

    - Attractive HUD design:
        - Create HUD that does not disturb the player's vision but can be consulted whenever he wants.
        - Make HUD clear and concise to the player.

    - Optimization of 3D models:
        - Reduce polygons as much as possible without noticeably affecting the vision of objects.
        - Apply maps of Normals to add quality without much cost.
        - Reduce the size of texture and normal maps so as not to overload processing.

3.2. System design

3.2.1. Flow Diagram

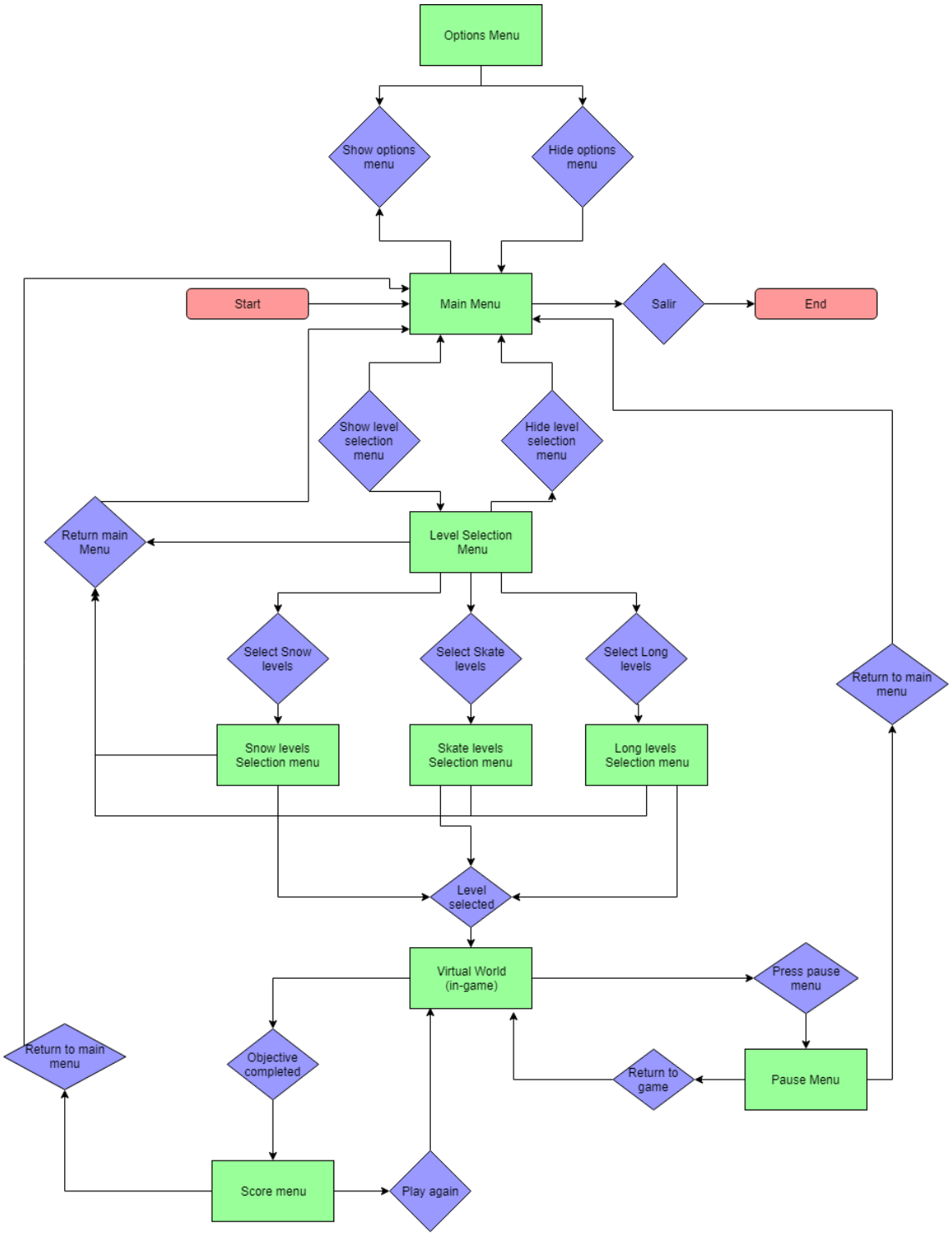


Figura 3.1: Flow Diagram

Use Diagram Practic Case Study (based in Figure 3.1 diagram):

The player opens the application and the first screen he sees is the main menu, which is made up of an "Options Menu" panel and a "Select Levels" panel.

If the player presses "Quit" the application will close and exit the game. If the player presses the "Options Menu" panel he will access this menu. Finally, if you press the "Level Selection" panel you will access the level selection menu.

We assume that the player wants to adjust his experience, such as the sound of the application, and access the "Options Menu", he will be able to return to the main menu at any time.

The player finally decides to play and accesses the "Level Selection Menu". In this new screen, very similar to the previous one from which you came. Then you will find three different options: "Snow Levels", "Skate Levels" and "Long Levels". In any of these three screens the player will choose the level he wants to play within the mode. These three options are very similar in terms of flow as their destination is the same, the "In-game" screen.

In the virtual world, which will be the stage where the player will enjoy the real experience, you will be able to access two panels in different circumstances: "Pause Menu" and "Score Menu". By accessing the "Pause Menu" the player can go to the main menu again or follow the match "In game". Once the player has fulfilled his objective within the game he will access the "Score Menu". In this last menu the player after looking at his final score will be able to decide between returning to the main menu or starting another game.

3.2.2. Use case Diagram

- **Skate Diagram**

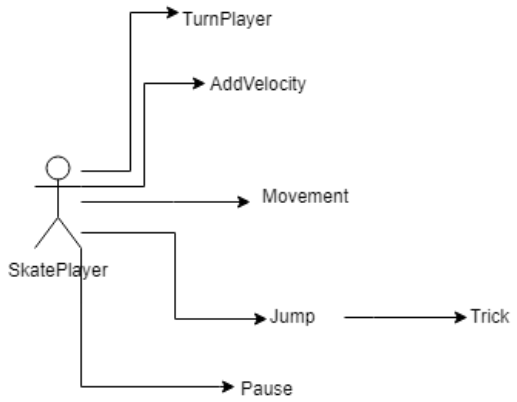


Figure 3.2: Skate Use Case Diagram

- **Snow Diagram**

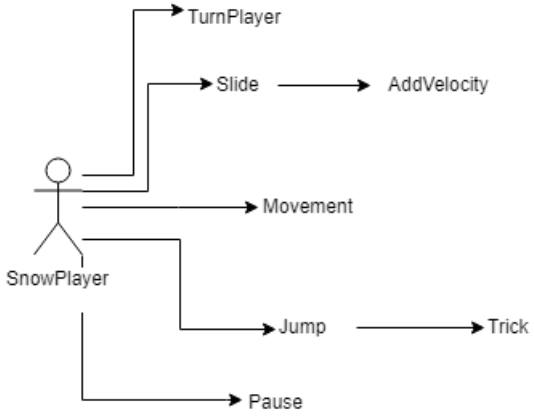


Figure 3.3: Snow Use Case Diagram

- **Long Diagram**

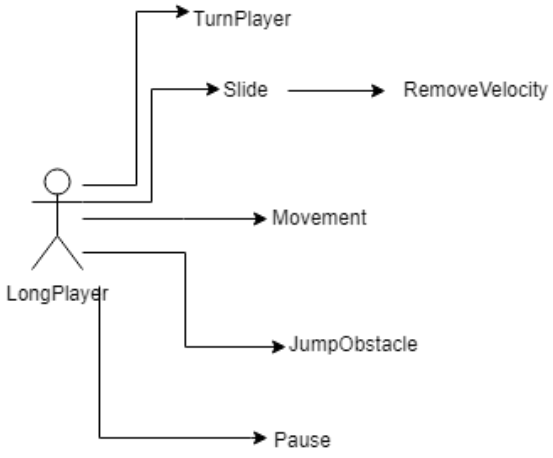


Figure 3.4: Long Use Case Diagram

- **Use cases:**

Case of use	**TurnPlayer**
Description	The main character makes a turn from a certain angle (Public Variable).
Main actor	SkatePlayer, SnowPlayer and LongPlayer *(Fig. 3.2, 3.3 and 3.4)*
Preconditions	The character shouldn't be down at that moment. The player must be playing.
PostConditions	The player will have a different rotation based on the entry value given by the player.
Normal Flow	❖ The Player is in play ❖ The user turns the Joystick ❖ The Player makes the corresponding turn ❖ The Joystick is left at a neutral point and the player stops spinning.
Alternatives	❖ The player is in the fall phase ❖ Joystick does not work

Case of use	**AddVelocity**
Description	Speed increase in the variable that determines the movement of the table.
Main actor	SkatePlayer, SnowPlayer and LongPlayer(the same but in the oppossite) *(Fig. 3.2, 3.3 and 3.4)*
Preconditions	The character shouldn't be down at that moment. The player must be playing. The player must be touching the ground.
PostConditions	The player will have a different speed value than before the use case.
Normal Flow	❖ The Player is in play ❖ The user presses the action button ❖ The variable Speed is modified ❖ (Long) When you un-push the button, it returns to its normal state.
Alternatives	❖ The player is in the fall phase ❖ The Player is not in contact with the ground ❖ The player is doing an acrobatic ❖ Button does not work

Case of use	**Slide**
Description	The main character makes a turn at a certain angle greater than the normal rotation.
Main actor	SnowPlayer and LongPlayer*(Fig. 3.3 and 3.4)*
Preconditions	The character shouldn't be down at that moment. The player must be playing. Must be in contact with the ground.
PostConditions	The player will have a different rotation based on the entry value given by the player.
Normal Flow	❖ The Player is in play ❖ The user presses the action button ❖ The user turns the Joystick ❖ The Player makes the corresponding turn ❖ The Joystick is left at a neutral point and the player stops spinning. ❖ The user un-pushes the action button
Alternatives	❖ The player is in the fall phase ❖ The Player is not in contact with the ground ❖ The player is doing an acrobatic ❖ Button does not work ❖ Press the button and do not rotate the Joystick

Case of use	**Movement**
Description	The main character moves with a certain direction to a different point in the scene.
Main actor	SkatePlayer, SnowPlayer and LongPlayer *(Fig. 3.2, 3.3 and 3.4)*
Preconditions	The player must be playing.
PostConditions	The player will have a different position based on the entry value given by the player.
Normal Flow	❖ The Player is in play ❖ The Player makes the corresponding movement
Alternatives	❖ The player is in the fall phase ❖ You are in the main, options or level selection menu

Case of use	**Jump**
Description	The main character makes a jump with a certain force (Public Variable).
Main actor	SkatePlayer, SnowPlayer and LongPlayer *(Fig. 3.2, 3.3 and 3.4)*
Preconditions	The character shouldn't be down at that moment. The player must be playing. The player must be on the floor.
PostConditions	The player will have a different rotation based on the entry value given by the player.
Normal Flow	▪ **SkatePlayer:** ❖ The Player is in play ❖ The user presses the action button ❖ The Player loads the jump he's going to make ❖ The user releases the action button ❖ The player jumps with a certain forcé ❖ The force of gravity attracts the player to the ground ▪ **SnowPlayer** ❖ The Player is in play ❖ The user enters the collision of a jump point ❖ The player jumps with a certain forcé ❖ The force of gravity attracts the player to the ground ❖ **SnowPlayer** ❖ The Player is in play ❖ The user presses the action button ❖ The user enters the collision of a jump point ❖ The player jumps with a certain forcé ❖ The force of gravity attracts the player to the ground
Alternatives	❖ The player is in the fall phase ❖ The Player is not in contact with the ground ❖ The player does not release the button.(SkatePlayer) ❖ Button does not work

Case of use	**Trick**
Description	The main character does an acrobatic and adds points.
Main actor	SkatePlayer, SnowPlayer *(Fig. 3.2 and 3.3)*
Preconditions	The character shouldn't be down at that moment. The player must be playing. The player must be in the air.
PostConditions	The player will have more points than before.
Normal Flow	❖ The Player is in play ❖ The user presses the action button ❖ The acrobatics are performed ❖ The player falls to the ground ❖ The corresponding points are added
Alternatives	❖ The player is in the fall phase ❖ Button does not work ❖ The player does not finish the acrobatics before touching the ground

Case of use	**Pause**
Description	The user stops the game until he wants to.
Main actor	SkatePlayer, SnowPlayer and LongPlayer *(Fig. 3.2, 3.3 and 3.4)*
Preconditions	The player must be playing.
PostConditions	The game will have stopped and the state of the game will have been preserved.
Normal Flow	❖ The Player is in play ❖ The user presses the action button ❖ The user pauses the game ❖ The user presses the action button again ❖ The user resumes the game
Alternatives	❖ The player isn't in Play ❖ Button does not work ❖ User does not exit pause menu

3.2.3. Class diagram

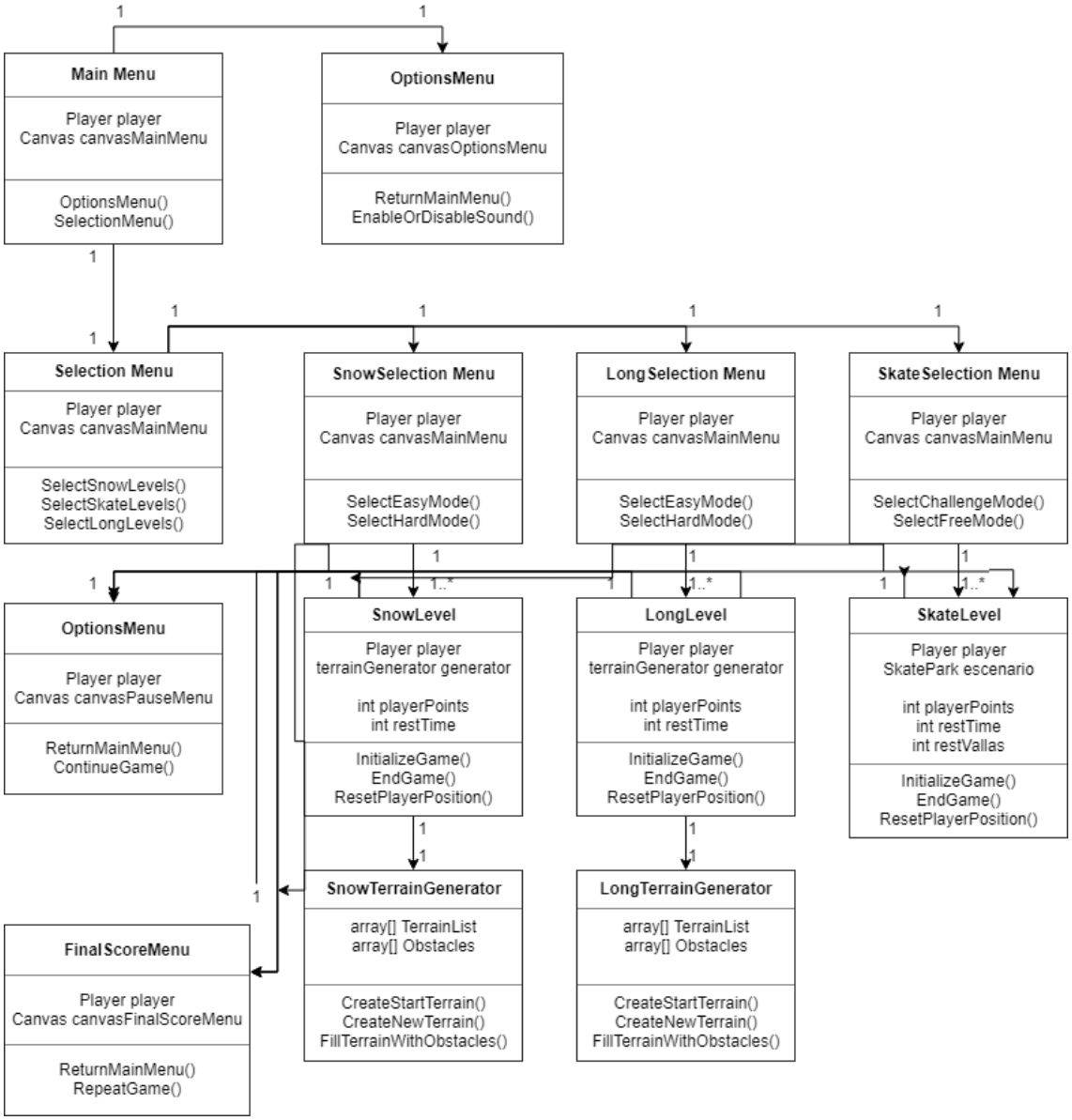


Figure 3.5: Diagram with classes and methods used in the project

3.3. System architecture

- Android Mobile 4.4 KitKat or higher. It is necessary for the API with which the app has been developed to work on the device.
- Mobile with screen of 5" or more. Since it is a Virtual Reality application, it is necessary a screen with a minimum size for the visualization of the game.
- Mobile with Bluetooth 3.0. This technology is needed to be able to connect the controller to the mobile in order to play.
- Mobile with accelerometer function. It is necessary for the game to direct the player's gaze correctly.
- FullHD or higher resolution recommended. Recommended so that the graphics are not pixelated.
- *2GB RAM or higher. So that the application can store data correctly and run the application optimally.*
- *200mb of free memory. Necessary space for the application to be installed in the phone's internal memory.*
- Count on CardBoard platform. Non-electronic device that will serve as a helmet to put the mobile and recreate Virtual Reality.
- Have a Bluetooth Remote. The remote connects wirelessly so that the player has no wiring problems.

3.4. Level design

This project was started with the purpose of being simply a playable experience, however, in its development it has been decided to add gamification aspects to make the game more addictive and competitive. At this point we will detail the challenges, objectives and scoring systems, etc., as well as the mechanics of the player and his adaptations for the proposed VR system.

- **Snowboard:**
    - **Challenges and objectives:**

In this mode, the user must traverse the descent accumulating as many points as possible in different ways. The player must make use of his abilities so that taking advantage of the mechanics offered by the game can make a good score. Speaking in playable terms, can be divided in two the sum that will give the final score: skills and time.

- **Difficulty and its variations:**

The player can choose between three different difficulties: Easy, Standard and Difficult.

These three difficulties will make changes in the game by varying the length of the course, making it more difficult for the player to reach the target time so that he does not score the player in negative.

Taking into account that in the most complicated difficulty to make a good time is around 220 seconds in maximum difficulty:

- ***Difficulty Easy:*** The track consists of 10 fragments
- ***Medium difficulty:*** The track consists of 17 fragments
- ***Hard Difficulty:*** The track consists of 25 fragments

- **Mechanics and scoring systems**

On the snowboard track, the player will be able to descend at a prudent speed, which will vary according to the orientation of the board with respect to the direction of the track. To increase this speed the player is offered two mechanics: **sliding** and **jumping**.

During the game, if the player jumps across the distributed fences and falls successfully, a speed will be added to the Snowboard as a reward so that it can go for a few moments faster. The same impulse will be given if the player presses the slide button, braking the board and making it spin faster. If the player slides and turns at an angle greater than 45 degrees, he will receive the impulse..

In this mode the player will have several ways to add points, one is through time as stated above. Along with the other ways to get:

1. ***Time:*** In the initial state of the game, the player is shown a counter that starts from 0, but internally there is a timer of 200 seconds counting backwards. As soon as the user reaches the goal, the player's game time is subtracted from the total counter, resulting in

a score that will be added or subtracted depending on the speed of the race.

2. ***Jumps:*** Along the terrain there are several types of ramps on which the player can jump. When landing, if the player falls correctly, the amount of 100 points will be added.

3. ***Aerial acrobatics***: When the player jumps and is in the air has the possibility to perform an acrobatic. If while performing the acrobatics the player falls to the ground, his position is reset and he does not get the score. If he manages to complete the acrobatics before reaching the ground a score of 50 points is added.

4. ***Sliding impulse:*** During the game the player can slow down so that he can spin faster. If the player starts sliding in an "X" orientation and stops sliding in a "Y" orientation at an angle greater than 45º, he will boost by increasing the speed and adding 50 points. If the angle between the two angles is smaller nothing will happen, it will continue to slide at the braking speed and will increase progressively until the standard speed is established.

- **Longboard**

    - **Challenges and objectives:**

In this mode, the player will travel the randomly generated circuit, accumulating as many points as he can. The player must make use of his skills to avoid obstacles and reach the goal in the shortest time possible. In a very similar way to the Snowboard modality, the total score can also be divided into concepts of speed and skills.

- **Difficulty and its variations:**

The player can choose between three different difficulties: Easy, Standard and Difficult.

These three difficulties will make changes in the game by varying the length of the course, making the road longer and reaching the target time more difficult, much like snowboarding.

- ***Difficulty Easy:*** The route consists of 8 fragments.
- ***Medium difficulty:*** The route consists of 14 fragments.
- ***Difficulty hard:*** The route consists of 20 fragments.

- **Mechanics and scoring systems:**

The player will drive along a downhill road. The longboard will accelerate without pause and without limit, so it is possible to accelerate so much that it is impossible to steer. To give the player control of the board, he is given the mechanics of sliding.

The slide can be executed as long as the player is not jumping. When sliding, the player will stop the Long in a forced way making it slow down at high speed. Apart from that, he will also have more rotation speed, which will help to control the board in the tightest curves.

In this mode the player will have several ways to add points, through speed, skills. Here are exposed each one:

1. ***Time:*** In the initial state of the game, the player is shown a counter that starts from 0, but internally there is a timer of 200 seconds counting backwards. As soon as the user reaches the goal, the player's game time is subtracted from the total counter, resulting in a score that will be added or subtracted depending on the speed of the race.

2. ***Jumps:*** Along the terrain you will find several fences that the player will be able to jump planning the maneuver beforehand. If the player orients the Longboard well and goes at a cautious speed, he can make it pass underneath while the player jumps over the fence. If the step is executed correctly, 100 points will be added to the player. In the event that he stumbles upon it, his position will be reset.

3. ***Trash:*** Along the course there is scattered rubbish that the player must avoid. This rubbish cannot be jumped in any way, it can only be overcome by dodging. They are strategically placed to force the player to brake and not be able to reach high speeds.

4. ***Slip-errape***: To stop the great speed at which you can put the longboard, has been developed a mechanic so that similar to the Snowboard, there is a button to brake and help the rotation of the board. That is to say, if the player is in front of a very closed curve and does not have time to turn, he will be able to brake and turn faster to continue his way. During the time that the player is sliding some points are accumulated depending on the time that will be added when you release the button.

- **Skateboard**

    - **Challenges and objectives:**

This is the modality that is less similar to the previous two but the basic gameplay is still quite similar. This game mode does not consist of a track but of a skate park, a specific location, which means that the terrains are not randomly generated.

However, the purpose of the game mode remains the same, to achieve the goals in the shortest time possible. In this case, jump as many fences as possible doing acrobatics to accumulate as many points as possible. There will be a light that indicates if the fence has already been passed or not, being red in negative case and green in positive case.

- **Difficulty and its variations:**

Since it is a scenario that is not generated randomly, the difficulty has had to be approached in a different way.

Three different fields have been developed for this modality and each one has a different number of fences to jump, so the player will take longer to jump them all.

- ***Skatepark Municipal*:** There are 5 fences to jump.
- ***Skatepark Pro**:* There are 12 fences to jump.
- ***Skatepark Puerto:*** There are 15 fences to jump.

It should also be borne in mind that walking around the stage and doing acrobatics can also get many points so the best way to get the best score will be a sum of time spent doing acrobatics and the sum that they give the fence jumps.

- **Mechanics and scoring systems:**

The player will be able to move freely throughout the scenario. The speed of the board will be managed by the player, who will be given impulse by pressing the corresponding button, without being able to exceed a speed limit. This impulse can be given as long as the player is on the ground.

You will also have the option to jump a certain height depending on what you have loaded the jump. The user presses the jump button and the player will reload the jump. As soon as the button is released, the force will be released and a jump will be executed with a force relative to the time the button has been pressed.

Being in the air, the player can execute a maneuver that gives extra points to the record. This is one of the many ways to add points:

1. ***Time***: In the initial state of the game, the player is shown a counter that starts from 0, but internally there is a 90 second timer counting

backwards. As soon as the player has jumped all the fences in the scenario, the game time is subtracted from the total counter, resulting in a score that will be added according to the speed of the game.

2. ***Acrobatics****:* When the player is in the air the player will have the opportunity to perform an acrobatic. If during the execution of the acrobatics the player falls to the ground, it will count as a fall and the points will not be added. If the player completes the acrobatics and then falls to the ground, 100 points will be added.

3. ***Fence's jump****:* Jumping the fences will also have a positive reward for the player, otherwise they would lose sense and make the player simply go around the scene ignoring the fences. If the player jumps a fence in a satisfactory way, a sum of 150 points will be added.

- **Adaptation of the game to VR technology**

The task of adapting the game to Virtual Reality has had several things to consider when developing the game.

The first was whether the game should be in 3rd or 1st person. Since the objective is to create an experience that generates real sensations it was decided the second option that will be in charge of offering the expected results.

The second question was whether to make the mechanics of the game based on the possibilities offered by Virtual Reality (use the look as a user input) or look for some alternative. I was thinking carefully about how the game was going to be, I wanted it to feel like a real experience in which you can move on your board while you look around. If I decided to use the gaze as an input, it would mean that it would give the player less freedom to explore his surroundings and that there would be more possibilities to make the player dizzy.

As an example, I thought of the mechanics in which the player looks down and up to give impulse, brake, slide, etc., however, this prevents the player from seeing what is in front for a few moments apart from generating dizziness by the continuous movement of head.

In response to this problem, I decided that the best way to control the character was with a wireless controller that would allow the player to rotate freely and control the character smoothly.

Once I figured out how to control the character, I had to keep thinking about how to take advantage of this system. For ease of programming, the entire UI in-game facet is handled with the controller, making the gaze independent of the player's decisions, as it is attached to the gaze. Otherwise the main menu is presented, where the gaze is going to be the main input of the player's selection. This is done so that the player moves his head inside the main menu thus inciting to explore around it, where you will find instructions and indications to make easier the execution of the game.

3.5. Interface design

At this point we will specify in detail everything related to the interactivity that the player has with the game, its characteristics and functionalities.

As it is a Virtual Reality game, the interface must be displayed in a way that is pleasant for the player so that this feature is well taken advantage of.

- **Main menu**

Description:

In this scene the different game modes will be presented, as well as the settings and instructions for playing.

If you look at the centre, you'll find three different game modes: Skateboard, Longboard and SnowBoard *(Figure 3.6)*. By clicking on them you access the different levels within each mode, all within the same scene if you look at the centre. If you have entered the level selection screen and look to the right, you will have exposed the objectives of the specific mode you have chosen.

If you look in the main menu to the left you will find an image with the game controls. These will be marked with the illustrative buttons of the game: Jump, Impulse, Trick, Movement, Pause. However, on the level selection screen you will find the special features of that game mode.

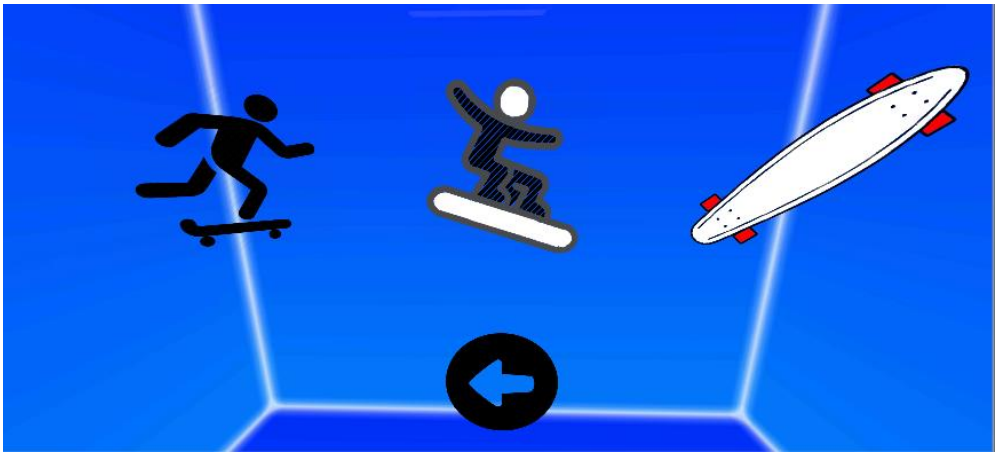


Figure 3.6: Menu in the Level Selection Screen

Game states:

This interface will be present on different occasions during the game. At the opening of the application this scene will be opened. If at the end of the game the player wants to exit, he can press a button that takes him to the main menu. It can also be accessed if the player is in the pause menu can also get to the main menu.

- **Pause menu**

This menu exits when the player stops the game during the game. This will be represented by a picture exposed in front of the player so that with his head facing the front of the table, he can see it without problems *(Figure 3.6)*.

There will also be different options that will be executed with some combination of buttons, either to resume the game or to exit the game to the main menu.

Figure 3.6: Pause Menu in a Snowboard Level

Game states:

This interface will appear when the player within a game presses the game button to stop the game.

- **Final Screen**

In this screen you will see the final results of the game (*Figure 3.7)*. The sum of the scores and the final score will appear. This screen will appear in front of the player as well as the pause screen. Within this menu the player will have the option to repeat the level or return to the main menu.

Figure 3.7: Final Screen in a Snowboard Level

Game states:

This interface will appear whenever the level has ended, either successfully or in a player failure.

- **HUD**

The HUD allows the user to know at any time the score of his marker, as well as the time he has or has left *(Figure 3.8)*. This will be marked in the upper left corner, trying not to obstruct too much the view of the player.

Figure 3.8: in-game HUD

Game states:

This interface will appear whenever the game is in execution state, since the player must know at all times the state of his game.

3.5.1. Design Development

The development of the game design can be divided into two parts: 3D modeling and 2D graphics.

- ***3D Models***

This has been the most expensive development in terms of design. Using the programs that have been specified throughout the memory, boards, obstacles and scenarios have been developed. The procedure for the development of each of these elements has been as follows:

1. **Modeling development**: Taking blender as a tool to use, each object is built from zero and then introduced into the game. Depending on the type of modeling, some techniques have been used. For example, for tables and ramps, a normal topology has been used, based on extruding points, edges and faces. However, using the Longboard roads as an example, a Bézier curve has been used to trace the route and, based on this curve, the road has been elaborated subsequently.

   Once the development is finished *(Figure 3.9)*, a pass is made to all the vertices to eliminate doubles, in this way when a texture is applied, there will be fewer adaptation problems.

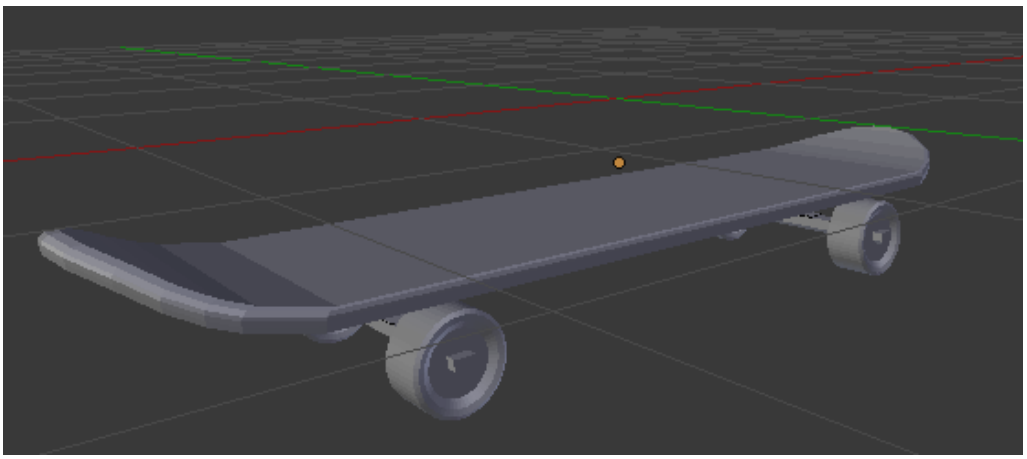


Figure 3.9: Skateboard Model without texture in Blender

2. **Texture maps:** Once the desired 3D model has been obtained, a suitable texture is applied. For this, on each model individually, a mapping of textures is created so that *(Figure 3.10)*, by means of an image, the modeled one is represented in a correct way. Once this is done, the model is ready to be exported to Unity in .fbx format.

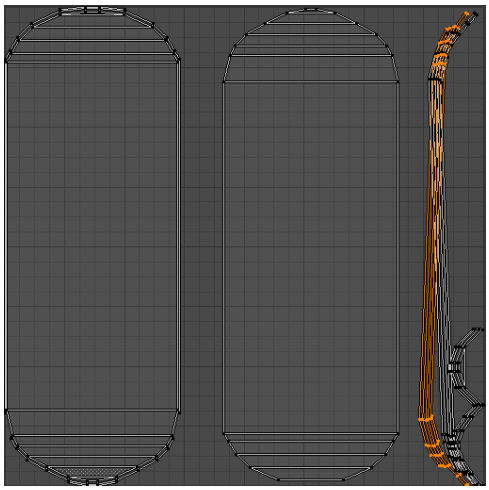


Figure 3.10: SkateBoard Model UV Map where be applied the Texture

3. **Albedo and Normal Textures**: Knowing and taking into account the mapping of the models, textures are developed to be applied later. Two types of textures are developed. In the first place, the Albedo texture is in charge of giving colour to the model. This texture is processed by Photoshop to make variations in brightness, tonality, etc., until the expected result is reached *(Figure 3.11)*. Once the texture has been developed, the normal texture is obtained through an Online program. This texture will give some extra details to the modeling without the need to overload the system. These textures once exported to Unity are compressed to occupy less inside the game.

Figure 3.11: PNG of the texture will be applied over Skate UV Map

- ***2D Graphics***

For the elaboration of the 3D graphics a Wacom Intuos Graphic Tablet was used as hardware and Photoshop was used as a software tool.

The elaboration of these graphs has been simpler than that of 3D models since it is enough to draw the desired graph and export it directly to the project.

First, I make some hand sketches on what is the idea I have to illustrate within the game. Once I have made a few designs, I scan the one I like the most and put it as a template in Photoshop while I draw over it. Once I get the desired drawing I paint, if necessary, the graphic, adjust the levels of saturation, brightness and contrast and are saved in .png format to store the transparencies.

Once in Unity they are configured so that they are correctly compressed without occupying more space than necessary and they are added to their respective place.

3.6. Audio

Both the music and the sound effects will be stored in .mp3 format, since they are light and although they do not have the best sound quality, they will serve for the case. The audios have been configured to have an adequate volume according to the sound that is reproduced and its importance in the scene.

- ***Music***:

The music in general consists of pieces of electronics, inciting the player to move and have fun. At each level the pieces change adapting to the theme being played.

The music, excepting in the main menu, has a secondary presence with respect to the sound effects of the game. This is done so that the sound effects that the player makes such as jumps, acrobatics, etc..

The soundtrack consists of 4 pieces: one for the main menu and three for each of the modes to play.

- ***Sound Effects:***

The elaboration of the sound effects has been generated from a unidirectional C1-U microphone, which in spite of not being a professional microphone obtains good results if the recordings are manipulated.

What has been modified of the recordings to improve the obtained result has been to suppress the noise of the different recordings that in some cases were quite big because they did not have an adequate space for the recording. The different sound effects are as follows:

- ***Fall to the ground:*** When the player is hit for playing incorrectly this sound will sound as a warning that something has been done wrong.

- ***Skate/Long & Snow Ride:*** When the player is in contact with the ground and is driving over it, a sound must be made from the wheels or from the board indicating the player who is on the ground.

- ***Jump***: When the player jumps in the Skate or Snow modes, a sound is made indicating that the player has gone into the air..

- ***Pressed Button on Menu:*** To tell the player that he is pressing the select button when he is in the menu this sound has been added to sound when he presses.

- ***Success Sound:*** When an option in the main menu is correctly checked or an acrobatic is performed, a sound is emitted that conveys that something has been done right.

4. Work development

4.1. Adaptation of board to surface

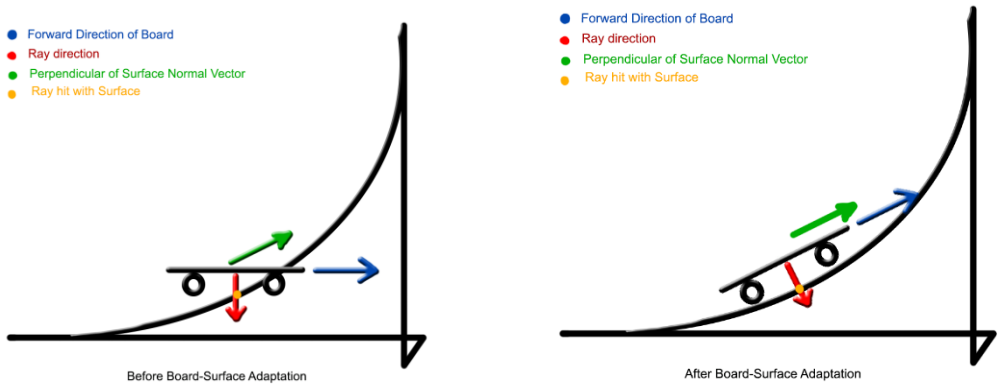
This is one of the fundamental bases of the gameplay, especially in the mechanics of the Skate. This aspect had to be developed using the Player from a CharacterController in Unity. RigidBody component can recreate the collision system and adapt the object to the base on which it stands, but it doesn't achieve the expected result. The CharacterController allows me to have more control over the physics of the object.

The tool usually used for objects in Unity is RigidBody, but why did I choose CharacterController? A RigidBody object has realistic features since forces such as gravity, board velocity, friction, etc. are added to it. However, most forces are applied automatically and cannot be manipulated. One of the big reasons I was told to use the CharacterController instead of RigidBody is that RigidBody did not have the "isGrounded" variable that indicates if the object is touching the ground. This is necessary to know which forces to apply in each moment and when to restrict certain movements. For example, when the player jumps in a "U", the player should make his movement perpendicular to the ground. If the movement were made with a RigidBody, when it was in the air, it would have velocities in the three axes, when what is wanted is that it is restricted in the axis that is jumping. Otherwise, with the help of the CharacterController I can handle this movement in a simpler way because I can handle the speeds in different ways: when it's stuck to the ground, when it's in the air/jumping and when it's jumping in a "U".

One of the disadvantages of using CharacterController is that the collision box does not correspond to the Mesh of the object and cannot be replaced by the one given by the component (a capsule), only modify its size and height. The RigidBody has the collision box that the object should have, but when the ramps are climbed, the object does not adapt to the surface as it should because the program's attempt to exercise realistic physics ends up being an incorrect result. However, by having total control of the physics using the CharacterController, the objective is that the object adapts to the surface it is touching in real time.

To achieve this, what has been implemented is a ray whose position starts from the base of the object downwards (taking as reference the direction "in front" of the object). This ray that points downwards with a certain distance will be responsible for determining if it detects an object in that range and if it is a surface on which to drive. If the surface is labeled "Ground", the ray detects the normal direction of the object with which it has intersected. With the vectorial product obtained from the direction vectors of the Player and the normal direction of the surface we can solve the problem of the exact orientation that the object must take at any moment.

In addition, this ray is also used to know if the Player is touching the ground or not and to take it as a collision with the ground to obtain information about the mobility that the player is going to have in each frame (if the ray touches the surface it is on the ground, if not, it is in the air). I have decided to use this function to determine if the player is touching the ground and not the "isGrounded" variable that the CharacterController has because the results it offers are imprecise and it has only been used in the project for certain checks. For example, if the CharacterController variable indicates that the player is touching the ground but the ray does not detect any surface it means that the object is looking at the sky, so the player should fall and reset his position.



Apart from the ray that is fired from the center point of the Board, there are also rays introduced in key places (e.g.: the four wheels of the table). So that when the player is in the air and close to the ground, if possible, smoothly adapt the board a little so that when it finally touches the ground the turn is not so abrupt. If these rays are not available at these key locations, the Board may be buried under the ground if it falls at a certain angle. These key rays were put to avoid bugs.

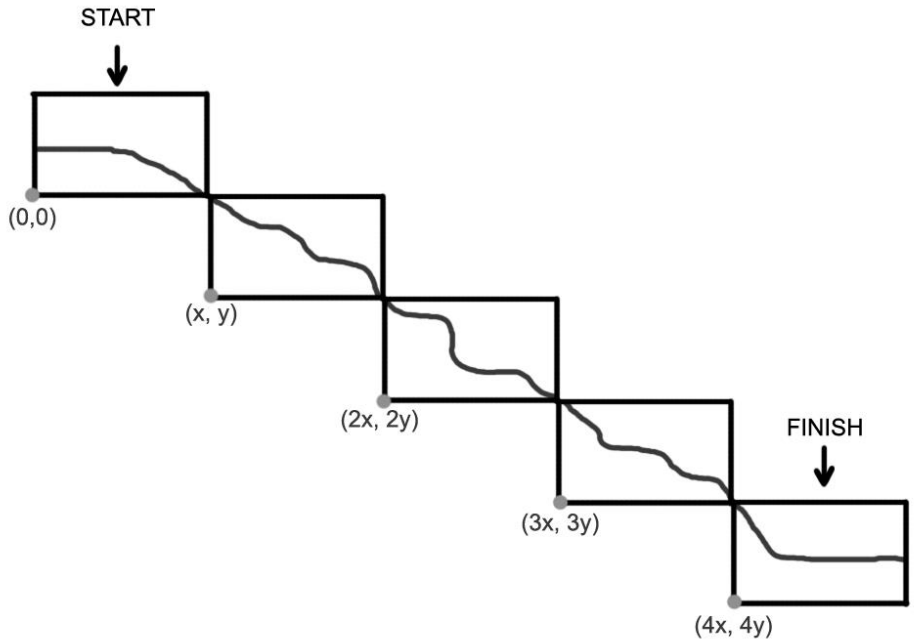
4.2. Random terrain generation

This feature has been developed for the disciplines of Snowboard and Longboard, as they are the ones that will randomly generate scenarios to prevent a repetitive experience.

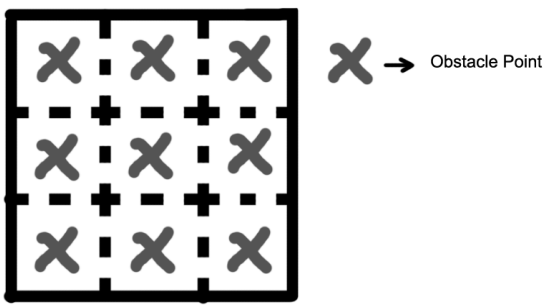
The principle that is going to be developed to fulfill this objective has been the generation of some templates of different forms and characteristics and to alter them, so much in order as the obstacles that are on them to be able to concatenate lands of a logical and effective way.

- *Snow Terrain Generator*

The first thing to solve the case in the discipline of Snow has been to identify that the sense of the track is going to be downhill and will not have curves. This characteristic has been of great help when elaborating the solution. Why? Because in this way I don't have to worry about where a terrain ends and where another one should start because when dimensions of all the templates are the same thing (even if the content is different) I can concatenate them one after the other, the only thing to take into account is to update the values of X and Y on which the next terrain is going to be generated.



Once you have resolved how the terrains are going to appear, the next point to deal with is how you fill in the obstacle templates and that each one is different. To do this, I create a Manager object that contains all the available obstacles instantiated. This Manager is called when a new terrain is generated. The terrains have 9 points evenly distributed. These spaces are filled with blanks, trees or ramps. It is filled in in such a way that its construction cannot give rise to scenarios that are inconsistent or impossible to complete.



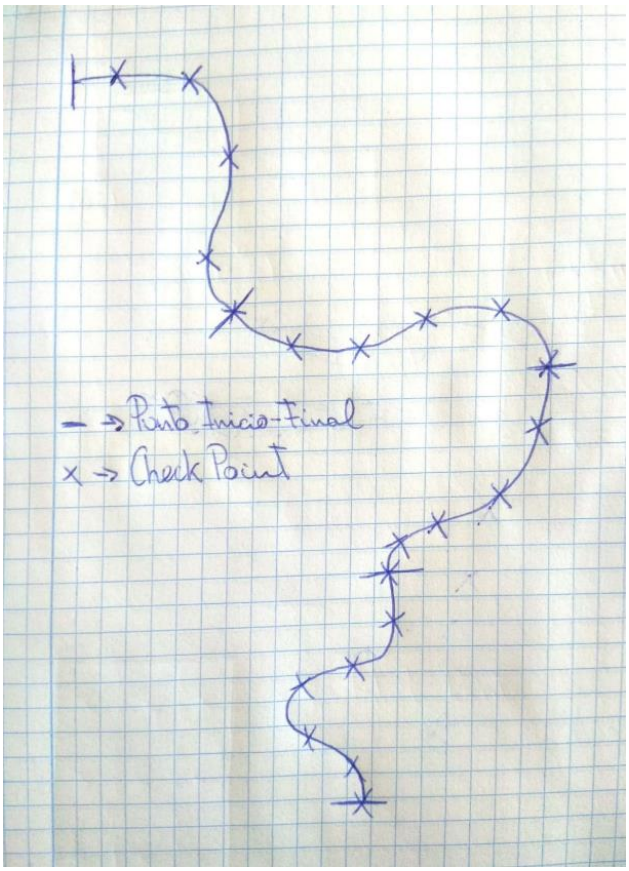
- Long Terrain Generator

This method has been very similar to develop with respect to the previous one but it could not be recycled because it has its peculiarities.

To begin with, it has not been possible to reproduce the concatenation of terrains in a uniform way due to their size, since they are curved terrains and will not have the same size. In this way, what has been decided to implement is an object that serves as a reference both at the beginning of the road template and at the end. Both points are centred on the road and looking in the same direction as the road at the point it indicates. Now yes, taking as a reference how the previous section has been implemented, we can recreate the same concept, but instead of indicating the values (X, Y) that mark the starting point and update them in each iteration, now take those points of Start and End to know where to put the object.

The principle of mechanics indicates that the player must make a continuous descent, but when executing the code, it is observed that being curved terrains there is the possibility that the road will ascend again and not obtain the desired result. To solve this problem, at the time of instantiating the terrain, a component is added that compares the positions of Start and End of the road. If it turns out that the Final position is higher than the initial one, the object is scaled in the Z axis so that as a result the object "mirror" of the same one, doing that now the highway does point towards where it should.

When it comes to filling in the obstacle roads, the implementation has been simpler. We have recycled the concept of the Manager that contains all the obstacles of the discipline. Each road has distributed "x" points that are filled when the game instancia the terrain.



Another consideration that has been taken into account at the time of developing the terrain, is that when the player falls, his position is reset to a point close to the one he was taking. In the case of Snow, it's simple because being a downhill terrain just have to adjust its position X so that this centered and there would be no problem but, how to solve it on a curved terrain? The solution has been to create along the terrain a series of Checkpoints that are updated as the player goes through them. In this way, if the player falls, his position is reset to the last Checkpoint through which he has passed..

4.3. Board movement

As previously described, this project has rejected the implementation of mechanics using RigidBody, which makes the physics system itself. The proposed alternative, CharacterController, works in such a way that in each frame a movement is made with a Vector3 that has been updated according to the state of the player.

The CharacterController API provides two methods of moving the object: SimpleMove and Move. Both receive as argument the Vector3 of which already it has been spoken, however, they act of different way. SimpleMove executes the movement according to the argument passed to it, ignoring the Y component of the vector. However, it adds a negative velocity on the Y-axis to simulate gravity. Otherwise, the Move function makes the exact motion you pass to it as an argument, including the Y component. The purpose of choosing CharacterController is to have more control over the object, so the first option is discarded.

It is really important not to execute the method that gives way to movement more than once per frame because it can lead to errors. The measure taken in this respect has been to initialize at the beginning of each frame to 0 a Vector3 and add the different velocities that will be applied to the object at the end. Once all the corresponding operations have been carried out, the movement method is executed and wait for the beginning of the next frame.

The first speed to be added is gravity which is formed by an initial fixed variable that increases as the Board is in the air. This speed only applies if the Player is in the air.

Then the direction and amount of speed is determined based on the direction of the player. For this speed you first take the skateSpeed variable which is the amount of speed that the object is going to have. This variable can vary depending on if the player is braking, if it has been driven, etc.. Once made the changes due to the variable is multiplied by the Vector3 of the Player's direction and is added to the global one.

Finally, if the player is jumping a positive speed is added on the Y-axis to raise the Player a certain amount. The jump speed is treated in the following way: when the jump is ready (the jump order has been executed), a quantity is added to a variable depending on how big the jump is. In each frame will be added that value that will decrease in function of the time until it reaches 0. When the jump value has reached 0 will mean that the player is at the highest point of the jump. After that moment the gravity force that will attract the object to the ground.

```
private void FixedUpdate()
{
    //Init Vector3 to zero
    Vector3 moveDirection = Vector3.zero;

    //Apply forces to the vector
    AplicarGravedad();
    VelocidadSkate();
    CogerInputSalto();

    //Execute move method
    playerController.Move(moveDirection);
}
```

Once these speeds have been added and calculated, the method is executed that changes the position of the Player and executes the collisions, if there are any, and restarts the process.

5. Results

It is important to point out that the project in my mind was a proposal to make a physics system without the one offered by Unity's Rigidbody. If we put that as an objective, it has been fulfilled since a system of self-sufficient physics has been developed that handles the speed of the player and the different accelerations that are applied to it (friction, gravity, impulses, etc.).

Apart from this, it has also been possible to implement the Virtual Reality system, which was the main motivation of the project and one of the fundamental objectives to be reach. In order to do so, it will be necessary to evaluate which technology is more profitable both in economic and implementation terms. The resolution has been possible thanks to the plugin offered by Google to develop Carboard applications in Unity's software. Not only has it been implemented, the mechanical systems have been developed taking this feature into account, making menus appear before the player's eyes, or using his eyes to execute some actions.

It is also an important goal to get an immersive experience, as the game tries to move the player to a unique experience. In combination with the technology of virtual reality, it is necessary to develop a good gameplay smooth and faithful to reality, along with scenarios and environments that, as far as possible, are able to make the user forget the real life.

Reduce the dizziness that can generate the application in Virtual Reality if it has been a complicated task. Dizziness is usually caused by the desynchronization of the sound and what is being seen. To avoid this, it has been necessary to manipulate the sound, making it different according to the speed of the character. In addition to this, the transitions between speeds are very marked since an unnoticed change in speed can cause nausea to the user.

In order to make the gameplay easy to understand, I have implemented information that appears in the main menu in order to explain the mechanics to the player. This way, the first moment the player is aware of all the mechanics contained in the game and can enjoy them without problems. However, an attempt has also been made to introduce sounds that indicate to the player when he is doing well or when he is doing badly. The whole application is built with simple and concise images so that anyone can understand them.

The project started as a game focused on Skateboarding, but seeing the possibilities of the framework I was developing I decided to take it to Snowboard and Longboard, with different mechanics, which has ended up being a big success. Not only have they been created but in some aspects have been even more fun than the initial modality because the design of levels is different. This was done in order to have a diversity of different situations so that the player doesn't get bored always playing at the same level. The intention of the game to include the three different sports is to give the player the option to decide what style he wants to experience that moment.

This game will have an score after each run to allow players to compete and create a more addictive experience. The initial approach is to create a casual experience, but this way seeks to promote the rejugability of the title. This generate a competitive component among players. The difficulties that have been implemented also encourage the player to try levels from lower to higher difficulty, generating a sense of progress in skills and in the scoreboard.

In order to recreate the realism within the game where the scenarios are faithful to reality and allow the player to enjoy the experience while the graphics do not affect the experience, textures need to be added to each model within the game. The size of these textures had to be reduced so that they don't saturate the internal memory of the device making the experience unstable. The final result can be improved as it is not a project dedicated to the artistic section although it has been important part. However, it is a game that does not impact the eye, in bad therms, and that allows the player to enjoy in its fullness without lag in mid-range devices.

Link to Playable Demo, Project folder and Video Demo-Gameplay:

https://drive.google.com/open?id=1uaXPeIdyMsV3748LnXEWRkXUyb8LUDl7

6. Conclusions and Future work

6.1. Conclusions

It should be pointed that this project has not only been the elaboration of a product but also a way of learning through various aspects.

The first aspect has been the elaboration of the game design from 0. It has been one of the most difficult things at the moment of making the game, since it has been a section that has not gone out directly, but that has been coming out by trial and error. This is notable in the different options I developed for the mobility of the Board on the surface, as I tried several ways to reach the one that best fit to my objectives. Similar debates I had to raise for the generation of Snowboard and Longboard scenarios.

In fact, even though I knew that in order for a project of this size to be polished in each of its sections, several people with specific knowledge would be needed. The fact that I developed the project alone has allowed me to work on each of the components that compose the game. I probably don't have full knowledge about how to make each one, but thanks to the development work, I have managed to have many ideas that allow me to elaborate specific ideas for each tool used.

In conclusion, not only have I learned to use better the software to develop the game I had in hand, also I have learned methodologies that can be applied to other projects that are inside my head in the future.

6.2. Future work

The work on the project has not yet been completed. For the future it is wished to upload this application to GooglePlay's digital platform. Not only this, I want to provide updates that add value to the product.

That means adding new options that configure the experience, especially audio volume and sound effects and optimization for different devices. In addition, with the framework developed, it would not be difficult to develop new game modes to try to make the experience more dynamic and extensible.

7. Bibliography

7.1. Used Assets

For the facilitation of the development of the project I have made use of some Assets that have added value to the application.

7.1.1. Cardboard Google Asset

This Asset is in charge of implementing the Virtual Reality system in the project. The Git repository with all its versions (used versión → 1.200):

https://github.com/googlevr/gvr-unity-sdk

7.1.2. Snowy Low-Poly Trees

Asset used to add inexpensive tree models to the scene.

https://assetstore.unity.com/packages/3d/vegetation/trees/snowy-low-poly-trees-76796

7.1.3. Music without Copyright

YouTube page with the non-copyright music used in the game:

https://www.youtube.com/channel/UCB1acOLDCxU8Te9OsgMRtSg/featured

7.1.4. Normal Map Online Creator

Page that generates normal maps from a texture:

http://www.smart-page.net/smartnormal/

7.1.5. Skybox repositorie

Collection of Skyboxes from which some have been extracted for the project:

http://www.custommapmakers.org/skyboxes.php