TFG

BOOM BOOM GO!

DEVELOPMENT OF AN ONLINE MULTIPLAYER GAME WITH AUGMENTED REALITY CAPABILITIES FOR ANDROID DEVICES

Submitted by Marc Prades Carceller
Mentor: Juan Miguel Vilar Torres

School of Technology and Experimental Sciences
Bachelor's Degree in Video Game Design and Development
Academic year 2017-2018

ABSTRACT

This End-of-Degree project consists of the development of an Online Multiplayer game using Augmented Reality technology.

The objective of the project was to develop a multiplayer online game that used the camera device to place a digital environment on a flat surface using Augmented Reality technology where the players would have played. There is no need for the players to be together in the same room as the scenario is shared across all the players.

The game is a "*Bomberman"* like type of game with fully network capabilities developed using Unity3D and is played on Android devices.

The users will play on a predefined arena in which they can move and drop bombs using a virtual joystick and a virtual button. The bombs explode after three seconds and have a certain blast radius. The objective is to eliminate the other players using the bombs. There are crates around the arena that can be destroyed by the explosions and have a chance to drop different power-ups.

Keywords: Game Development, Android APP, Augmented Reality, Online Multiplayer.

INDEX

LIST OF FIGURES

1. TECHNICAL PROPOSAL

This chapter constitutes the technical proposal of the End-of-Degree project for the Bachelor's Degree in Video Game Design and Development. It contains an overview of the project, the motivations, a list of the principal objectives to achieve with the work done, a planning of this development and the expected results.

1.1. PROJECT MOTIVATION

Nowadays the increasing popularity of online multiplayer games has ironically made them less and less social than ever. Back in the day, the limitation of the devices and multiplayer capabilities forced you play alongside with your friends side by side, which made the games much more compelling and competitive.

Many new technologies are trying to address this social insolation problem. Virtual Reality is building virtual spaces with avatars in which the people can connect, see each other, interact and play together. But all that stills being an abstraction of the real world.

Augmented Reality is currently the technology related with video games that has the greatest potential. It can merge the real world with the digital one.

The main goal of this project was to bring back that feeling of playing with your friends, create a game in which the players can be together around a table playing and interacting with each other using the Augmented Reality capabilities of their devices. But not being together is not supposed to be a limiting factor, so players could be able to be in a different physical space, as a shared virtual stage is created for all of them.

1.2. RELATED SUBJECTS

Several Subjects of the degree in Video Game Design and Development are related to this project, but the main ones are the following.

VJ1227 - Game Engines: The subject introduces the basic architecture of a game engine with special emphasis on the graphics engine and gameplay. In relation to the graphic engine, advanced aspects of the creation and efficiency of polygonal meshes, techniques for the representation of complex scenes and visual realism are studied. In connection with the creation of videogames a game engine is used to learn how to manage the scripting system.

VJ1228 - Networks and Multiplayer Systems: This subject pretends to teach the principles used on the communication layer protocols for the development of networked applications. It describes the principles of parallel, concurrent, distributed, and real-time programming issues for the design and deployment of multiplayer network games.

VJ1215 - Algorithms and Data Structures: This subject teaches programming as a method of problem solving and the influence of the chosen data structures and algorithms on the final efficiency of the programme. This gives students the foundation necessary to write programs that must handle large amounts of data in a short time.

VJ1221 - Graphic Informatics: This course focuses on the fundamentals of the process of obtaining synthetic images in real time, taking into account the pipeline of current graphic processors. This teaches the basic methods and algorithms of image synthesis, two and three-dimensional object interaction, and GPU animation, as well as some advanced techniques for applying textures, lighting, and visual realism.

VJ1216 - 3D Design: This subject pretends to teach techniques of modelling and texturizing of 3D objects and for its later insertion in videogames. In addition to studying the generation of illumination maps of the models to adapt them to the virtual scenes.

1.3. OBJECTIVES OF THE PROJECT

The following objectives are those that are intended to be achieved with the realization of this project.

- To develop a fully playable Bomberman like game.
- To introduce into the game an Augmented Reality system using the camera of the mobile device.
- To introduce Online Multiplayer capabilities so up to four users can play at the same time.
- To model the characters and the make the animations.

1.4. TASKS AND TIME PLANIFICATION

This section describes the division of the project's tasks and its estimation of required time in hours. The totality of the project has been divided into nine different phases.

Develop a playable single player game. – 30 hours

Player Model – 40 hours

Player Animations – 40 hours

Networking Implementation – 60 hours

Android Adaptation – 15 hours

Augmented Reality Implementation – 40 hours

Environment – 15 hours

Finishing Game Development (Power-Ups, HUD, Menus…) – 20 hours

Writing and revising the Technical Proposal, Technical Report and the Project Defence Presentation – 50 hours

1.5. EXPECTED RESULTS

It is expected to develop a fully functional game with all the multiplayer and mixed reality functions combined. On the gameplay and aesthetic field, the game will be quite simple, because the main effort has been focussed on other aspects.

1.6. TOOLS

The main tools used during the process of this project are the following:

Microsoft Word: Word processor belonging to the Microsoft Office suite

Unity 2017: Game engine for 2D and 3D games.

Vuforia: Framework that helps to implement Augmented Reality capabilities into Android and IOS applications.

Unity Networking: Unity native multiplayer framework.

3DS Max: 3D modelling software belonging to the Autodesk software family.

Photoshop: Digital painting software belonging to the Adobe software family.

Sublime Text 3: IDE that support C# and can be linked with Unity

UMotion Plus: Paid asset to complement the Unity work pipeline. It allows an easy and visual way to animate 3D objects and characters inside Unity.

GitHub: A Git repository hosting service with version control system.

Prezi: Program for creating and editing presentations.

2. GAME DESIGN DOCUMENT

This chapter discusses the Game Design Document (GDD) which outlines the developmental and design specifications of the game as its mechanics, controls and the environment where it is developed.

2.1. OVERVIEW

2.1.1. Game Concept

The game is an online multiplayer deathmatch game where four people drop bombs across an arena and try to hit each other with the resultant explosions.

Four players need to carve a way across the arena destroying boxes with explosions. The boxes have a chance to drop power-ups for the players. Each player starts with a bomb, that will be replenished after it explodes. They will have to be very careful with which boxes they destroy, because new paths will be opened, not only for them but also for their enemies.

Initially, players had to scan a mark with their camera devices in order to place the game scenario in Augmented Reality, and as soon as everyone was ready, the game would begin. Due to the problems and limitations encountered with the multiplayer framework, that will be discuss later, this option was incompatible with the multiplayer capabilities, so in the end had to be removed. Illustration 1 shows an early game image with a basic scenery setup.

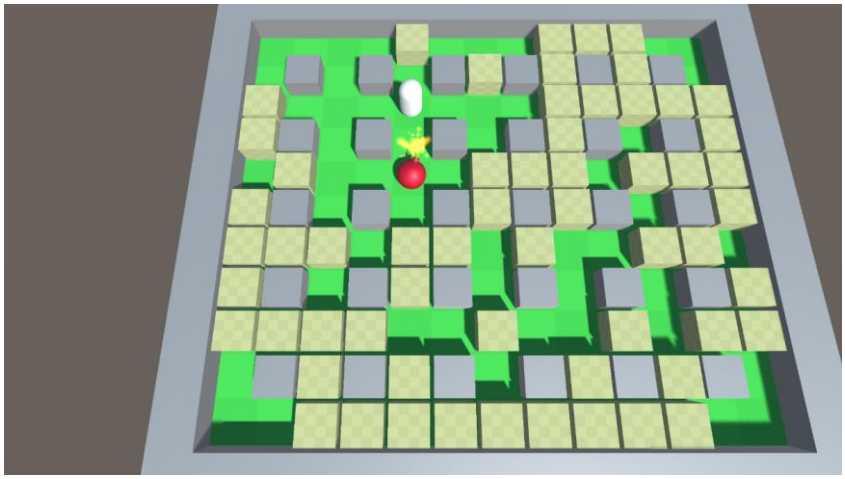


Illustration 1 Early game image.

Each player starts on a corner of the arena, and every match has no time limit, it ends when one, or none player is left standing.

The reason to choose this game style is due to his frenetic, competitive and fast paced gameplay. This game style allows the player to interact with each other, trying to defeat the others and to not be defeated. This allows for healthy competition between them. This scenario is perfect to develop the main motivation of the project, bringing players together and have a good time with friends.

2.1.2. Game Controls

The players will interact with the game by the following means.

The users have a digital joystick on the screen and an action button.

With the joystick, the user can move his character in any direction and control the movement speed up to the max speed.

With the action button, players can drop a bomb if they have one available

Illustration 2 shows how the controls will be seen on screen.

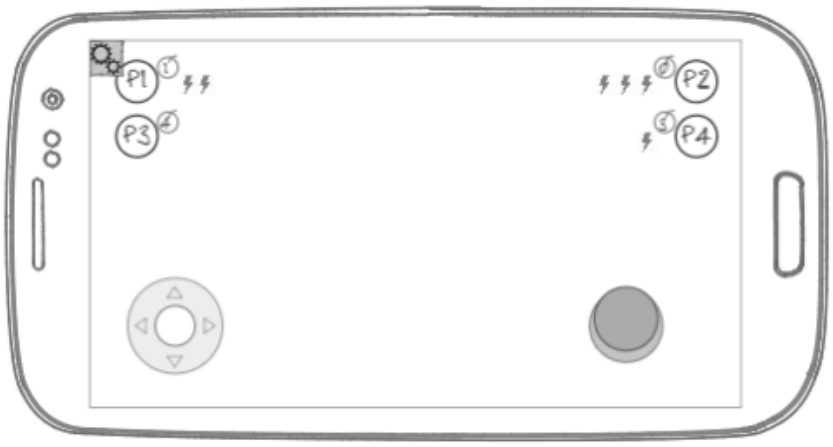


Illustration 2 Game controls layout

2.1.3. Game Goals

The goal of a match is to end up being the last player standing. In order to achieve this, the player must avoid bomb blasts both from enemies and his own. Once a player wins 5 matches, he/she wins the game.

2.1.4. Project Goals

The goal of the project is to explore the new Unity Networking multiplayer system and the capabilities of Augmented Reality to create games.

By using together multiplayer functions, Augmented Reality and a compelling competitive play style, it is possible to get a group of friends playing around a table, competing with each other and sharing a good time together.

2.1.5. Genre

The genere of the game can be catalogued as: Casual Multiplayer, Strategic-Labyrinthian and Deathmatch.

2.1.6. Target Platforms

The game is built for Android platforms. In a future, with the installation of Xcode, an IOS version could be easily ported.

During the process, a PC version was also developed. This version contains fully networking capabilities that works perfectly and with no delay at all.

2.1.7. Target Public

The target will be users that play other competitive games with quick-paced matches like Clash Royale, commonly aged between 14 and 25 years old. But the game is simple enough to be played by any kind of user. Adults that played Bomberman when they were young can be a potential target too.

2.2. GAMEPLAY MECHANICS

Main game mechanics are:

Player movement: The players can walk around the areas with no walls, pillars or blocks. Players can not walk across bombs.

Bomb drops: Players can drop bombs on their current position. The players can drop bombs on empty positions until they run out of them. They have to wait until the previous bomb explodes to automatically recharge a new bomb.

Bomb explosions: Bombs explode three seconds after they have been dropped. The blast of the bombs expands across four directions: Up, Down, Left and Right. The blast radius is of three units initially. If the blast hits a player, they die. If the blast hits another bomb, that bomb explodes instantly. This can produce a chain reaction.

Breakable objects: Pillars and walls are not breakable and will stop the blast. Boxes can be destroyed to open a path, but the blast will not go further in that direction. Destroyed boxes can drop power-ups like an increase of the player speed, more bombs or extended radius of the blast.

2.2.1. Game Flowchart

In this section the game flowchart for the different screens is explained.

A title screen is the first thing that the player will see when starting the APP. In this screen an art title is shown together with a text that says, “Tap to start”.

The Main menu will appear when the screen is tap. In here, the player can choose to create a new game room or to find one already existing. The players can write the name of the new room or search all the rooms available.

Once on inside the game room, each player can select his colour, that will change the character main colour, and write a name for his character. The name will be displayed on the top of the head of the character during the game.

On the game room, the players can go back to the previous screen by pressing the back button. When everyone has pressed the ready button, the scene will change automatically. The game will only start if there are four players connected and ready.

On the game scene, there is a phase that was initially used to allow the players to scan the mark and place the AR world, once everyone was ready, the game would start. Right now, this ready check prevents the match to start before everyone is inside the game. Some devices are slower to load the game scene, so this step is needed.

Illustration 3 shows the screen flowchart and a mock-up of the game screens.

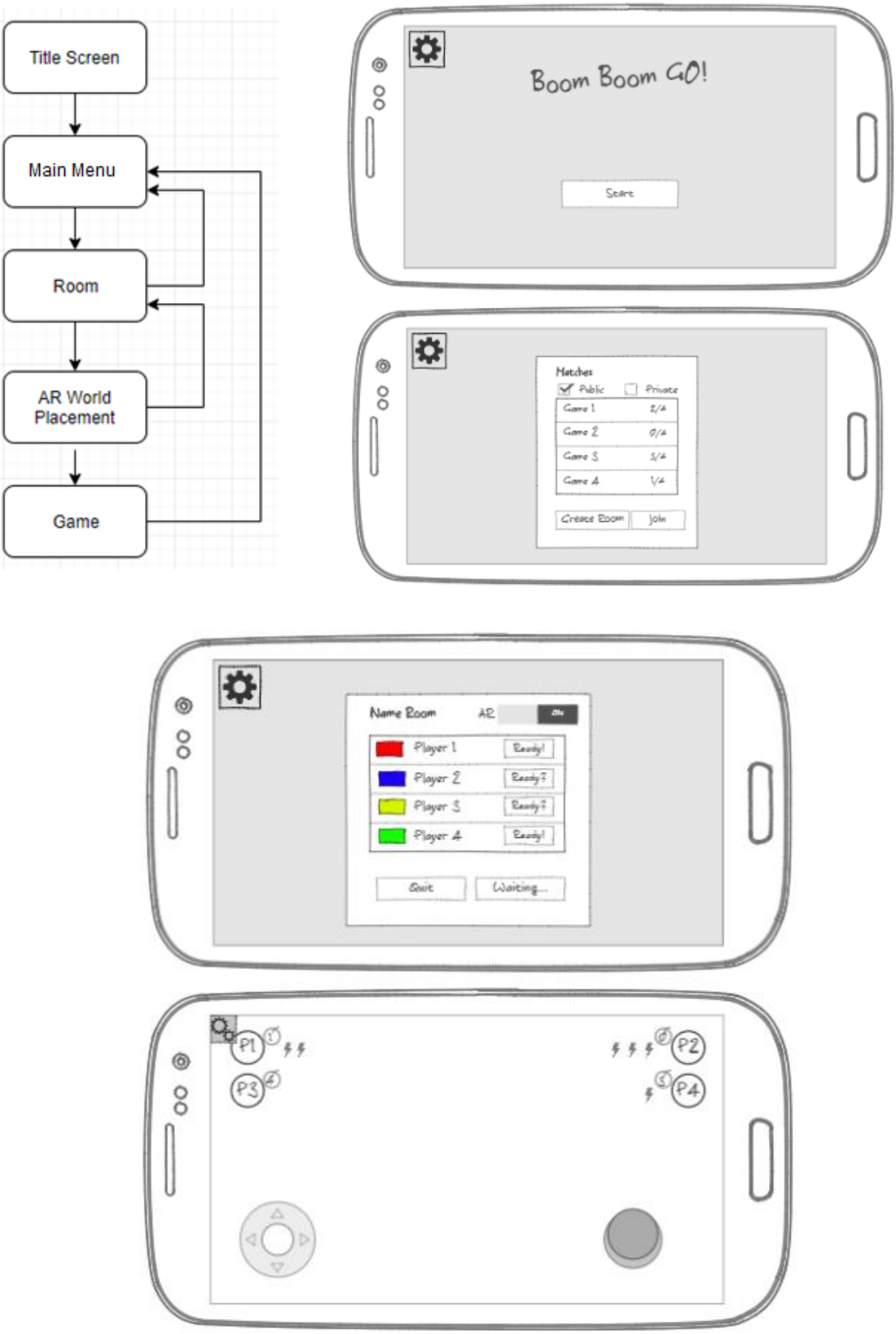


Illustration 3 Game screens flowchart

2.3. PLAYERS

2.3.1. Player Character

The player character works just as an avatar. A jester was the best fit for the game. It's a bright and happy character, that the player can easily imagine throwing bombs.

Each player can choose the main colour, and everyone will have a different one. Illustration 4 shows the character model.

Ilustration 4 Final model of the Jester

2.3.2. Player Metrics

Each player has the same maximum speed. Each player has one bomb, that is replenished three seconds after it dropped. The three seconds period is also applied when the players has more than one bomb. In that case, each bomb is replenished three seconds after a bomb has been dropped.

2.3.3. Player Skills and Power-ups

The players can get a series of different Power-ups by breaking the crates on the game. Those Power-ups can be: Increase the number of bombs, increase maximum player speed, increase bomb blast radius.

2.4. GAME WORLD OVERVIEW

2.4.1. Levels

During the development of the project, only one level has been considered. Due to the continuous round multiplayer type of gameplay, more levels are not needed.

The level has rock wall textures applied to the indestructible objects, box textures for the breakable ones and a green grass texture for the ground to create a nice contrast.

The textures have been bought on the Unity Asset Store, and normal maps have been created for all of them. Illustration 5 shows an image of the final look of the stage.

Ilustration 5 Screenshot of the PC version

2.4.2. Sound and Music

It is difficult to find music and sound that fits a certain game. The best option in these cases is to hire a freelancer musician to compose music specifically adapted to the game.

But for this project the decision was to search and buy music and sound effects from online libraries. There are two tunes, one for the menu and other that is played during the development of the match.

The menu song is a catchy tune, resembling those of the arcade games of the 80s, with a simple base, little accompaniment and repetitive. The gameplay song instead, is a rockabilly instrumental tune, fast paced and very rhythmical. It fits perfectly with the action-packed gameplay.

Several sound effects (SFX) has been bought too. These sounds are added to the bomb fuse wick, the bomb explosions, and the players footsteps.

2.5. INFLUENCES

This section explains the influences that has been taken into account for the game and why they have been, in both the gameplay and the art fields.

Gameplay

Bomberman - Videogame (1993)

Illustration 6 Bomberman game image

The decision of making a game with this gameplay was taken after thinking of a game that could benefit from being together one next to the other. The game had to be quick paced and competitive, something like when one plays Mario Party with friends.

Art

Final Fantasy IX – Videogame (2000)

Illustration 7 Final Fantasy IX Art concept and in game scene.

For the game art the style had to be something bright and cartoonish. The reason for this is that this type of aesthetics is the one that best fits visually for mobile games. I thought on Final Fantasy IX and its SD (Super Deform) style.

2.6. PROJECT SCOPE

2.6.1. Technological Requirements

Game Development Framework. Unity is the Game Framework that fits the best for a 3D Android game development. Unity is a user-friendly software for developing games and APPs with a lot of support and information from the community.

Network Capabilities. After study the different existing supports for networking and narrowed the decision down to two, Photon and UNET. Photon is the most popular because it has been around several years and it has a good performance. UNET instead is the new Unity Networking framework, and it is supposed to be more flexible. For this project, UNET has been decided to be the framework to use, to see if it offers the performance that is need for the game.

Augmented Reality. Android has developed his own AR framework, ARCore. The main problem with ARCore is that it is compatible with less than fifteen devices [1]. Vuforia is the best alternative to go. Vuforia has its own layer to make easy the AR implementation and decides depending on the device sensors and capabilities witch AR engine to use (ARkit for the compatible Apple devices, ARCore for the compatible Android devices or his own engine for others).

2.6.2. Publishing

Initially, the game will not be published. In the case that it is decided to publish it, it would necessarily include an IA and bots to play against.

The main problem of this kind of games is that at release, there are not enough players to find matches, and that makes players leave the game. Because of this reason, it is necessary to create bots that simulate human behaviour to fool the players and make them think that they are playing against actual players.

2.6.3. Monetization Model

The game will not have any monetization model at all. It will not have objects in-game for purchase nor ads.

2.7. ASSETS NEEDED

2D

Walls textures, floor textures, crate textures, character textures, HUD joystick, HUD action button, menus, power-ups, game HUD.

3D

Character model, blocks, crates, bomb model, bomb fuse particle effect.

Sound

Bomb wick, blast, footsteps, menu music, match music.

Animations

Player idle, player walking, player dropping bomb, player dying.

2.8. SCHEDULE

Playable Base Game - 12/02

GDD – 25/02

Player Model – 04/03

Player Animations – 15/03

Networking – 30/03

Android Adaptation – 08/04

Augmented Reality Adaptation – 22/04

Modelling Environment and Memory development – 06/05

Finishing Game Development and Memory corrections – 16/05

2.9. RISK ASSESSMENT

During the development of the project there may be several risks to consider.

Analysing the steps to be taken to develop the project, the most conflictive points are: the player modelling, the player rigging and animations, the networking, the Augmented Reality adaptation and the environment modelling.

As a preventive action for the problems that can occur, there is a contingency plan for each scenario.

Player Modelling

Risk: The model does not fulfil the expectations, its quality is not acceptable or the number of triangles is too high for a mobile game.

Contingency: Buy a model for the player from the Unity Asset Store.

Player Rigging, Skinning or Animating

Risk: The process can take too long, and the end result cannot be acceptable.

Contingency: The maximum time spent is one week longer than expected on the schedule, hence it will be three weeks at the most.

For making the animating process easier a professional animating framework for Unity, UMotion Pro, has been bought. UMotion Pro is a Powerful Animation Editor for animating any type of 3D model right inside Unity. It is very user friendly and visual, and it offers some online tutorials to learn to use it efficiently.

If the maximum time is exceeded, a default animation to a default humanoid skeleton downloaded from the Unity Asset Store will be applied.

Networking

Risk: The networking implementation is not possible for any reason.

Contingency: If a functional PC version of the game is achieved submit that version. If networking does not work at all, ask the mentor for a change of project and submit the Business Card Builder Framework developed during the internship period.

Augmented Reality

Risk: The Augmented Reality implementation is not possible for any reason.

Contingency: Submit a completed version of the game without Augmented Reality capabilities.

Environment Modelling

Risk: The process takes too long, or the results are not satisfactory.

Contingency: Buy environmental assets from the Unity Asset Store.

3. PROJECT DEVELOPMENT

This chapter explains the in-depth development phase, highlighting the implementation work done in detail.

The structure of this chapter is linked to the methodology carried out. For the project progress and development, an agile methodology based on Sprints has been applied, in which the small and complete objectives must be met in order to give solution to the different components of the project. These sprints have a duration of one, two or three weeks depending on the complexity and difficulties that have appeared.

The particular agile methodology that has been used is called Scrum. Scrum is a work methodology which people can address complex problems, while productively and creatively delivering products of the highest possible value. It allows to iterate around one core product that is always evolving.

By using this work methodology in the project, there is always a working version of the project at different stages.

Those sprints can be split into six large blocks of work: Playable Demo, Fully Singleplayer Game, Online PC Game, Online Android Game, AR Online Game, Final Game.

3.1. PLAYABLE DEMO

The first of the major development block aims to create a basic playable game with most of the core mechanics.

The firs steps were to set up the project and work framework, to achieve an optimal workflow. A Github repository was created, and Sublime Text 3, a sophisticated text editor for code, was integrated into Unity.

To start with the game scene, the first thing to do was to create a basic player with a responsive movement controller. There are two ways: changing the object position manually or moving a rigidbody applied to the player.

The rigidbody is a component needed to handle physics and it has a variety of methods to handle its interaction with the world, movement and rotation included. The problem of the physics engine is that it runs one hundred and twenty times by second, while the regular update engine runs at sixty.

To optimize performance, the player is moved by changing his position manually on the regular update. This is made by checking if the user is pressing one of the movement keys: W, A, S or D. Then, the elapsed time between frames is calculated and is multiplied by a base movement speed. The final position is calculated, and the player is set to that position using a movement interpolation.

A bomb model from the asset store has been used, but the light fuse particle effect and the blast fire effect has been manually created. This has been a chance to experiment with the Unity particle effects system.

The bomb explosion code was developed afterward. That was a problem, because the bombs could not be dropped anywhere, they had to be mathematically aligned with a non-existing grid. If the bombs are not aligned with the floor, a lot of detection problems would occur. Illustration 8 shows how this problem was addressed.

```
72
73          CmdBombInstantiate(new Vector3(Mathf.RoundToInt(myTransform.position.x),
74          bombPrefab.transform.position.y, Mathf.RoundToInt(myTransform.position.z)),this.radius);
75
76          StartCoroutine(AddBomb());
```

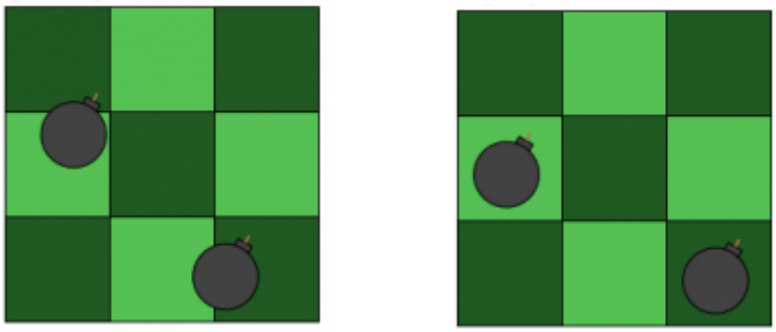


Illustration 8 Code of the bomb snapping function and visual example

Then the bombs had to explode and expand the fire in the correct directions. A three seconds timer is set in every bomb and it starts when the bomb is instantiated. After the three seconds, the bomb mesh disappears, and four raycast are thrown in four directions, Up, Down, Left and Right. A radius is passed to the function to set how many squares the blast moves. The raycast check if there is something on the way, and act according to the different elements on the scene. The code will create explosions following the raycast path. The explosion does not go further if there is a wall or pillar. If it finds a box, the box is destroyed and the explosion does not go any further. In the case that a player is hit by the explosion, his movement and ability to drop bombs are disabled.

The next move was to limit the number of bombs that the player could drop. A counter keeps track of how many bombs the player has. Every player starts with one bomb, and when one is dropped, the counter decreases. The player is not able to drop another bomb on the same space that there is one already nor the bomb counter is zero. After the bomb explodes, the bomb counter will be increased by one, so he/she can drop another one once more.

At this point, by testing the game, several problems were found. Sometimes, players get stuck when they dropped a bomb. The bomb collider did not allowed the player to move. The solution was to deactivate the collider on instance and activate the bomb collider once not a single player was colliding. The next problem was that during the game, when an explosion hit another bomb, nothing would occur. A chain reaction system was coded, to make the game more interesting. When an explosion raycast finds a bomb on the way, it automatically triggers the explode function.

The last problem was that the mathematically correct hit detection to check if the player has died seems unfair to the user, so the solution was to reduce the hitbox of both, the player and the explosions, until it seemed like it was already right to the player's eye. The players do not want mathematical precision, they want to fool themselves into thinking that they are good. This game strategies are very common during the game development [2].

The last step was to implement the breakable blocks and put them into the scene. The decision was to put them always in a fixed position to make the scene as balanced as possible. Illustration 1 shows an image of the game at this state.

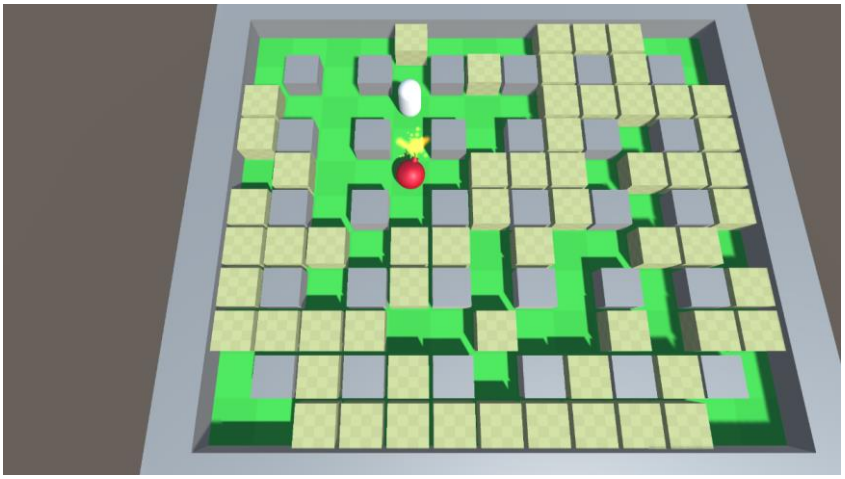


Illustration 1 Early game image.

3.2. FULLY SINGLEPLAYER GAME

Once a playable demo was created with the core mechanics of the game, the next step was to create the models and animations for the player, together with some basic user interface. This decision was made to avoid making changes on the future once the multiplayer capabilities were added. By doing this first, the game could be tested on a more stable environment.

At this point there were plenty of time to decide what kind of player model and aesthetics would be implemented into the game. A cartoonish aesthetic with a bright illumination is the graphic style that suits the best in a mobile game. It is the most appealing and the most efficient to a mobile device.

The decision of what type of character include in the game was made at this point. It had to be something that fit the aesthetics, something bright, but that the player could easily imagine throwing bombs at each other. At the end, it was decided that a Jester would be the perfect fit for the game. The Jester could have a balance between a bright happy character and a creepy one that could happily try to murder the others using bombs.

A graduate in fine arts helped with the concept design of the jester and the modelling process started. The model was low poly, and once was finished, its mesh was optimized to simplify the number of triangles to the minimum without losing quality. Illustration 9 shows the art concept evolution for the Jester and Illustration 10 shows how the 3D model is built

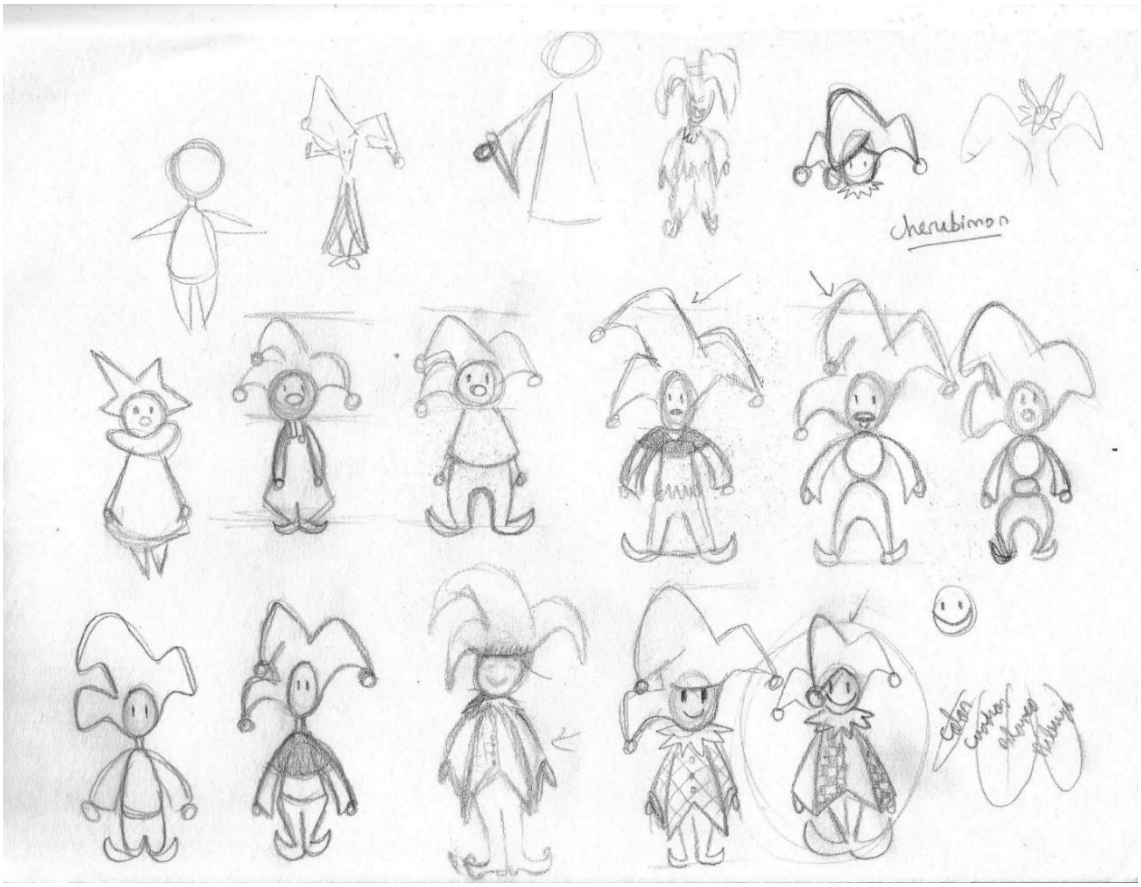


Illustration 9 Jester concept art

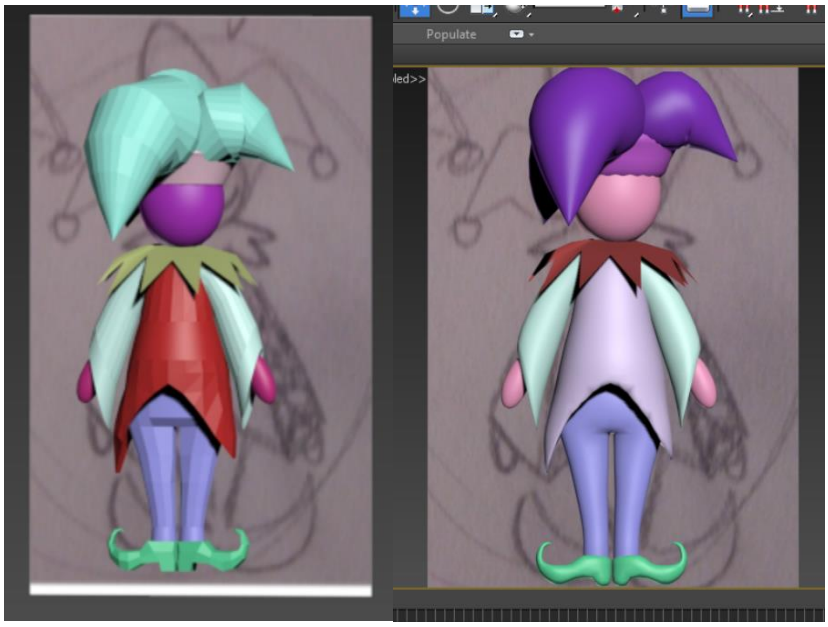


Illustration 10 3D design made on 3D Max

The next step was thinking of how to manage the different colours that the player could choose. The solution, at the end, was to create several textures, a different one for every player colour and change them on runtime. The images of the hole process can be found in the following link: https://www.dropbox.com/sh/rul5e9h8il6uey4/AAAa2jixbJM2eyvK4eNHLixza?dl=0

The animations were not done yet. An animation framework that could be implemented into Unity was bought. This framework, called UMotion Plus, allowed to simplify the animation process and did not interfere with other Unity systems like the physical animation system.

When trying to animate the model, there was found that the skeleton, the rigging and the skinning of the model had to be made by hand on the 3D modelling software. This process by itself took two weeks as it is a really difficult process. It had to be do and undo several times because once it was done, on the animation process, it was find out that the skinning or the bones were wrong. Advice was asked to some 3D artist on how to position the bone anatomy and how to move the bones. An example of the problems found where that if the arms were moved as they are supposed to be anatomically, the cloth would defy all laws of physics. In the end, the bones were put in a way that is not the anatomically correct one to avoid some movement constrains. Decisions had to be made and animations had to be changed.

The animations had to fit the whole character flavour. They had to show a happy bouncy character. This is represented across all the animations. Figure 11 shows the bone structure of the model.

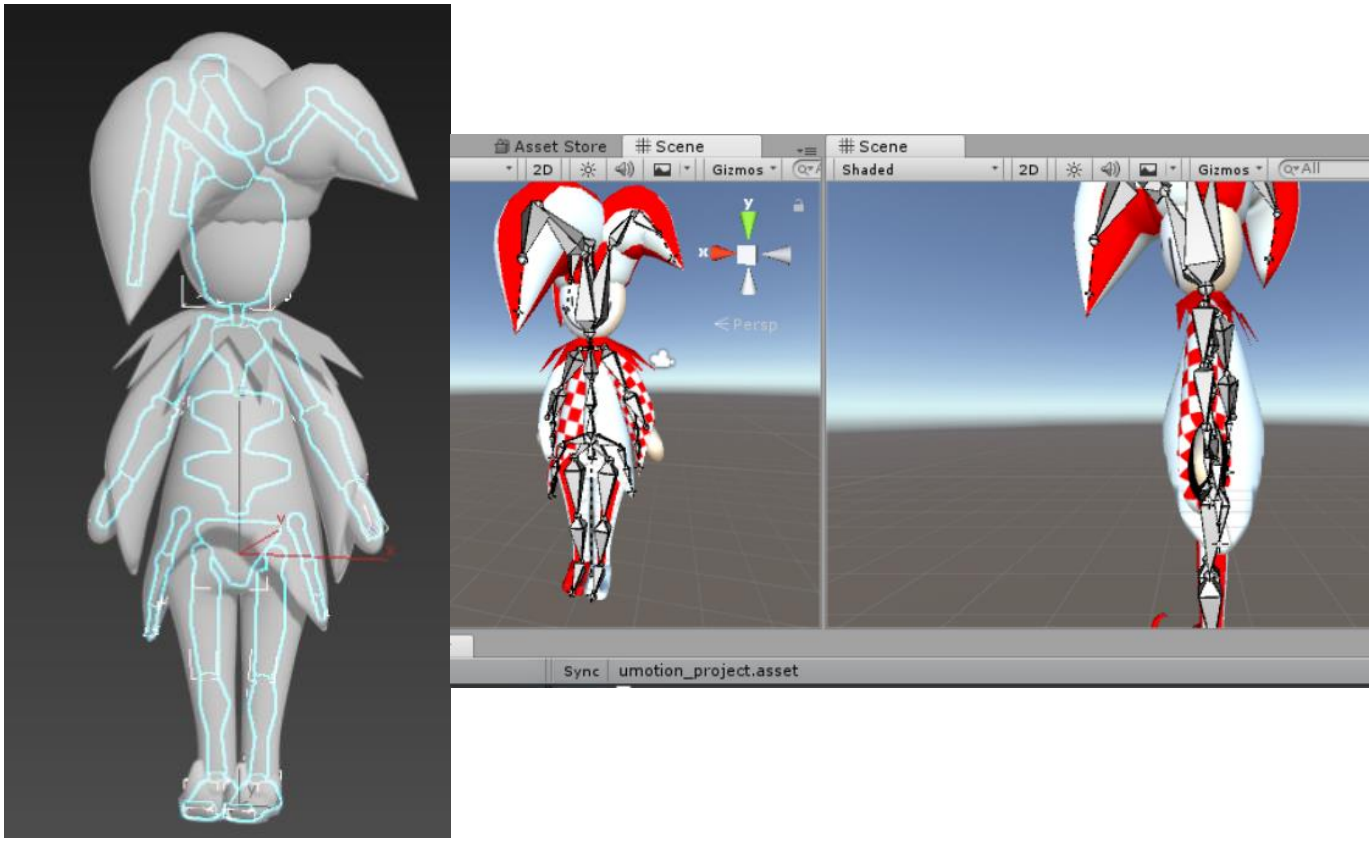


Illustration 11 Model with Bones and imported into UMotion

Once the animations where done, to add some flavour, information about how to implement physic-based animations was investigated, and some parts of the model, such as the hat, the sleeves and the shirt, have been animated using physical based animations. This enhance the animations and made them far more interesting. The final result of the animations can be seen in the following link: https://vimeo.com/273840105

Lastly, to finish the single player experience, there were left some work to do. The power-up system was implemented. Every crate destroyed has a chance to drop a random power up. The chance is calculated depending on the crate distance to the centre of the stage using this function and represented on the Illustration 12.

Chance = 6 - distance*0.5

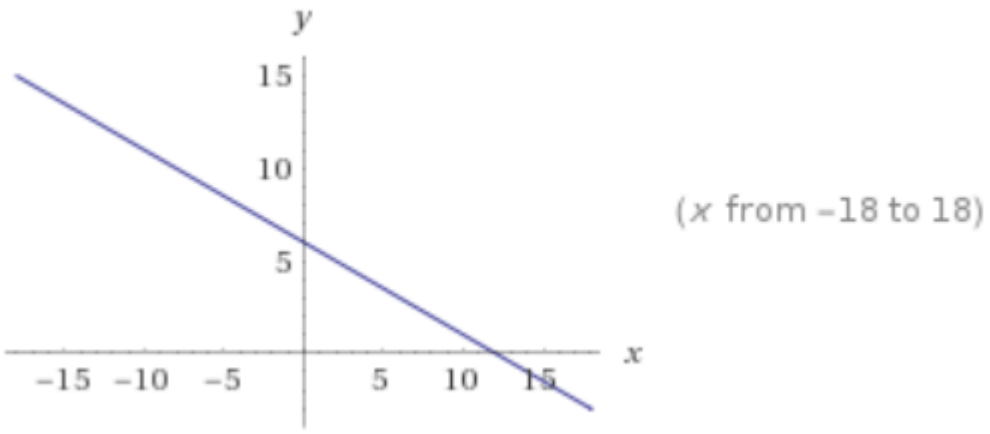


Illustration 12 Graphic representation of the power-up drop chance in relation with the distance

As an exception, the crate on the centre will always drop a power-up.

The textures for the environment were bought on the Unity Asset Store. In the end, it was decided that the textures were rocks for the indestructible elements, such as walls and pillars. Crates for the ones that could be destroyed, and a green bright grass for the ground, to give a nice colourful contrast. To enhance these textures, a normal map was created to each one of them, to give more depth to the scene. A normal map changes the light calculations that are applied to the material, this allows the elements to have a perception of depth and prominence.

3.3. ONLINE PC GAME

At this point, the game was already consistent, so it was the time to implement the multiplayer capabilities.

For the Networking framework, the final decision was to use UNET (Unity Networking). This decision came after much deliberation. I had used Photon in the past. Photon is the most used Networking framework. It has been around several years, and it is the most solid one. The problem with Photon is its high learning curve. There is a lot of documentation, but a lack of practical examples.

Last year, Unity 5.6 was released, and they called it “The Networking Update”. Previously, the Unity Networking framework was practically unusable. During this last year, the word has spread about the goodness and ease of making an online game with Unity. It was the perfect chance to try UNET and compare it with Photon and learning on the process to use both networking frameworks.

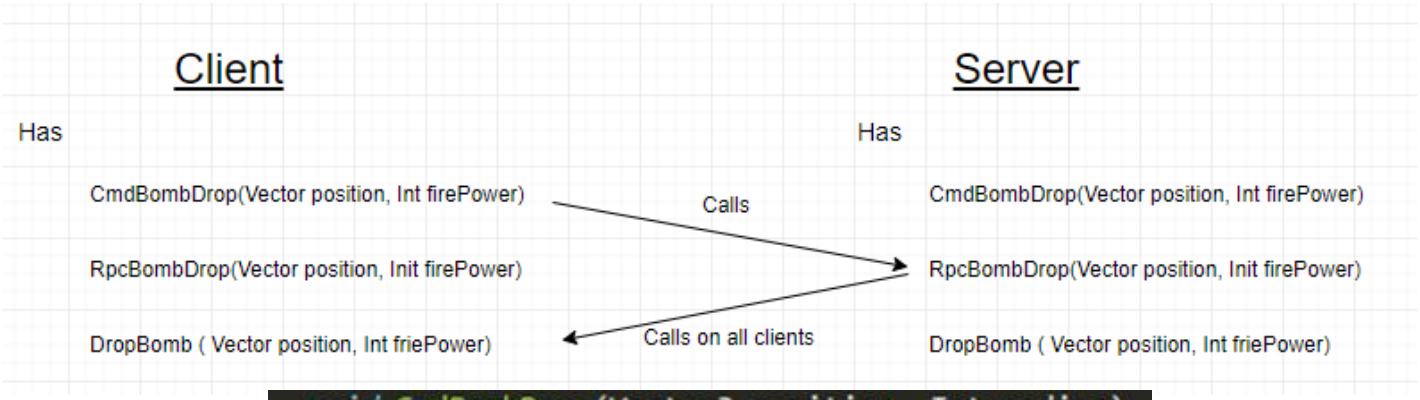
The implementation of the networking capabilities was easy due to the previous experience with Photon. For example, some things to consider are that when one player spawns, it spawns into the game the other three players. As a networked object, that will cause that all spawned characters are created across all the clients, so at the end, it is a common error to end up with the number of players to the power of two, because every login player has spawned three additional characters for the enemies.

The developer has to address this common problem by spawning only his own character as a regular character and sending a signal to the other clients to spawn an empty character with his properties linked to the original one. The empty players are constrained to not being a real player unless it is spawned by the local player. The player position, rotation and animation state are continuously synchronized by the Unity networking.

When developing a networked game, the functions have to be as independent as possible, receiving all the parameters needed externally. This is because of the RPC calls.

RPC (Remote Procedure Call) are functions that once called can be executed in all the instances of the object across the network. The only one with the permissions to execute that kind of RPC is the server. So, for a client to execute one action, it has to send an RPC to the server, and then the server has to execute the RPC across all the clients. An RPC function that is called only on the server by a client is called a CMD.

As an example, Illustration 13 shows the way the functions are called:




```
void CmdBombDrop(Vector3 position, Int radius)
{
    RpcBombDrop(position, radius);
    BombDrop(position, radius);
}

void RpcBombDrop(Vector3 position, Int radius)
{
    BombDrop(position, radius);
}

void BombDrop(Vector3 position, Int radius){}
```

Illustration 13 Pseudocode written to show how the functions are called

The Network lobby system was a real challenge. It is a hard and complex topic that involves a lot of documentation, time and effort. A lot of tutorials were seen, because most of they built a too much simplistic lobby. For this project, a far more professional lobby system was needed. In the process the whole UI were added too.

With the lobby created, the player can create a server with a custom name or search for existing servers. Once in the server room, the player can choose a colour for his character and change his name, the lobby scene has hooks to pass the necessary information to the main scene.

At this point, a ready check before the match starts was needed. This is done to allow the players to scan the mark and set up the AR scene. Another reason to implement the ready check is that depending of the device, the change of scene can have a variable time, so is required that all the players wait to each other and the match does not start automatically.

While implementing this, a major flaw in Unity Networking was found. On the documentation, Unity says that there is a function to give client authority for Non-Player objects [3], but after a month of work and research, it was found that it does not work. There are several forum threads where the people complains and asks for help [4] [5] [6]. Even a post on the forum help threads was made to see if anyone could help [7]. The final conclusion was that the only objects allowed to call or use any type of Network related functions are the Players (You have to indicate which object will be the player on UNET, and only can be one type of object).

The reason to make a non-Player manage network related functions is to create several managers to make the application flow as elegant and efficient as possible. Due to this limitation on the Unity Networking system, all the commands and code had to be pushed into the Player Prefab, and manually create listeners to check the conditions. On behalf of code efficiency, those listeners are destroyed on runtime once they achieve their proupose.

At this point, the alpha testing started. The core game was already designed, and the network capabilities were functional. The project was built for PC and shared across different close persons to play and test both the multiplayer and the game.

Thanks to the testing phase, it was found out that there was some delay between the clients. Sometimes, in one client, an additional box was destroyed, and this only occurred on that client. When the server destroys a cube, it destroys it on every client, which causes that the raycast of the client finds the space empty and the explosion goes ahead. The solution was to delay the destruction of the cube by 0.01 seconds.

This lag affected too how the player movement was perceived by the other clients. A lot of times the player just started to teleport across the field, or a player saw how he catch an enemy in an explosion, but in fact, the enemy had escaped with no harm. This was very common and gave the players an unfair vision of the game. A whole optimization of the game was made, and manually changed the send rate of information. If too many information was sent, the whole game was slow down, but if the information is not sent at sufficient intervals, the player teleporting problem would occur.

It was clear that UNET is easier to use if you stick to the tutorials and demos but is not a flexible framework and it does not give you too many tools to work with.

Once those problems were solved, the focus was on finishing the game completely for PC, implementing all the functionalities, UI and polishing. By doing this it was achieved to be always working with a polished version of the game in which is only needed to add the new functionalities.

3.4. ANDROID IMPLEMENTATION

Once the PC version of the game was completely functional and could be played without any problem, the next move was to port the game to Android.

The first step was to create and implement the controls with UI Joystick and buttons. Without this, no matter how well goes the port, the game could not be tested.

At this point it was the time to start the port. The process went with no trouble.

The first time to run the game, a problem was found. The player could not be controlled on the horizontal axis. This only occurred on the Android version. On the PC, the joystick worked perfectly. After some time, it was found that the problem was because the joystick moved the rigidbody, but for the rotation was controlled in the transform, so a Gimbal Lock effect was produced.

Then, a major problem was found. If a client player was two to three seconds without pressing the screen, the server will disconnect it from the match. This occurs because UNET detects the mobile device network (even connected to a Wi-Fi network) as a bad network connection and kicks the user of the server.

The way to fix the problem was to change by hand the Timeout Disconnection function inside the Unity networking classes. The fact that it was needed to change a hidden UNET framework class made me conclude that UNET is not suited for Android.

With the problems solved and the game successfully ported to Android, it was time to test the game. In this beta round of testing there were involved testers of all type of skills, ages and genders. People who played videogames of twenty-five years and thirty-five, people who does not play video games of the same ages and people of sixty years that does not play videogames. The results as gameplay experience were incredible positive. All the people were incredibly engaged and easily catched up to the game no matter what skill level they had previously with games. It was always tested with four players simultaneously.

During the testing process it was found that the delay between clients was a problem again. The information send rate was re-calibrated to try to fix the problem and it was mostly solved, however due to the lack of optimization of the Unity Networking framework for mobile devices, it seems inevitable to have some little delay.

3.5. AUGMENTED REALITY

After having a fully functional Android game, it was the time to implement the Augmented Reality functionality.

The main idea was to implement a markerless technology that allowed the players to scan a flat surface and place the game environment on it and play with the sensation that the game was really in front of them, on the real world like a tabletop game.

The markerless AR detection works the same no matter the framework. It uses the gyroscope and selective image stabilization to navigate trough a replica of the real world. The mobile is set as the centre of coordinates, and it starts scanning the area. It creates a cloud of collision points using 3D image comparison to recreate the environment. This cloud points are used as reference between images to navigate better through the environment and provide a extended tracking. When a lot of cloud points are coplanar, a flat surface is detected and uses a postprocessing edge detection to find the boundaries of the plane.

But after some research, it was found that only certain devices can use this kind of technology [8]. The devices need a series of high precise sensors. For Android devices, the Requirements are Android Version: 6.0 (Marshmallow) or newer, IMU w/ Gyroscope Sensors and Selective Image Stabilization.

In behalf of a more approachable software, it was decided to use the former UJI logo as a marker and develop the application with the conventional marker scanning process.

To implement augmented reality into a game, there are some things to take into account.

An AR element has to be created, and it will be the one that appears and disappears depending on whether the mark is being tracked or not. All the game elements that will be augmented have to be child of this AR element to be activated or deactivated. This causes the problem that core elements of the game management can be disabled in the middle of the match, or simply they are not enabled at the beginning. Another thing is that the game state must be kept until all is enabled again. The way to solve tis is by making all the core elements of the game managed by external elements that take into account the current state of the game, if it is detecting and playing or not.

This was a little more difficult that it sounds due to the multiplayer problems mentioned before. Only the player object is able to manage the network information, and they can not do it if they are disabled. The solution for this problem was that when someone lost track of the marker, all the players positions and other game state conditions to consider are saved, then the players are changed to a not augmented object, all their interaction are disabled, and they are moved out of the screen, but never disabled. By doing this, a pause signal can be sent to all the other players until the lost player reconnects. In this way the game will not crash due to the disabling of important network managing elements, and the game could be resumed when everything was right again.

There were too many problems trying to implement the solution above, time was running out, and at this point it only worked for the server. By adding the Augmented Reality load to the game, the problems given by the multiplayer framework worsened to the point of making the game unplayable. A decision had to be made, and despite a single player version was achieved and working properly, the result of this project would be the previous working version of the game, an online multiplayer game without Augmented Reality.

The result single player experience with augmented reality can be seen in the following video: https://vimeo.com/273839361

Illustration 14 shows the look of the AR game.

Illustration 14 Image of the AR Game

4. RESULTS

As a result of the process, a functional online multiplayer game for Android has been achieved, and it can be downloaded on the following link: https://www.dropbox.com/s/dkgahal2yzwajbs/BoomBoomGo.apk?dl=0

Through the game needs of four people to be played, here is a video of the gameplay: https://vimeo.com/275499882

The whole project can be downloaded in this repository: https://github.com/ZoroastrianMK/TFG_NoAr. In this repository there are not all the process, the hole process was originally saved on another repository that was changed when the AR system was removed. You can find the other repository on the Attachments section.

The Augmented Reality capabilities have been possible to be added to the game, but at the expense of having to remove the multiplayer capabilities as shown on the 3.5 section of the document.

The Android version of the game is good enough to be playable and enjoy the game, but it contains certain minor problems. On some occasions, the players can experience some delay on the other players. As explained before, this still happening even after optimizing the game.

Another problem which occurs sometimes is that a client can see how other player suddenly “dies” at the start of the round, or even itself. In fact, the player does not die, the root cause of the issue is that the animation state synchroniser of UNET is not working correctly. This conclusion has been reached after an in-depth debugging. The only part of the whole code that the die animation is triggered, is when the player actually dies. This function is not called by any means at that point of the game, in fact, the player can still play, move and drop bombs. Even when every player is forced to keep Idle at the beginning of the round the problem does not disappear. Not even that seems to override the Animation Networking Synchronization state.

The objectives declared on the Technical Proposal at the beginning of this project has been accomplished as we can see on the 1.5. chapter, even if they have been made individually:

- A fully playable Bomberman like game has been developed.
- A version with an Augmented Reality system using the camera of the mobile device has been achieved.
- Online multiplayer capabilities have been implemented, and four users can play at the same time.
- The characters have been modelled and animated from scratch.

But the results have not been the expected at the beginning of the project, where it was expected to merge the online game with the Augmented Reality system.

The schedule has been respected. Maybe a week of work has been replaced by another task, but overall the initial time assumptions have been correct. The environment was not modelled, so by the week of the 06/05 all the things concerning the game development and the environment modelling was already done.

Original schedule:

Modelling Environment and Memory development – 06/05

Finishing Game Development and Memory corrections – 16/05

Final schedule:

Memory development – 06/05

Memory corrections – 16/05

5. CONCLUSIONS

Last year I made a proof of concept of a simplest online multiplayer game using Photon. In that case, once I breached the problem of the learning curve, I managed to create a game with up to six players could play simultaneously with no delay.

Illustration 15 shows an image of the game end result.

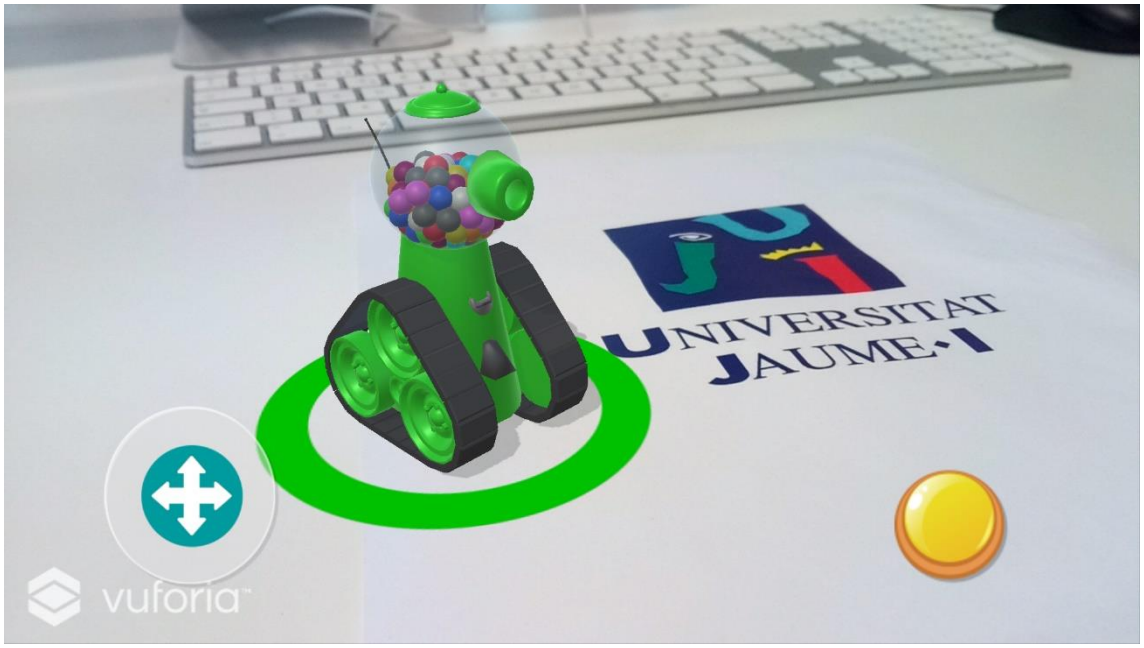


Illustration 15 Caramel Fighters. Demo developed on 2017

As a result of these last months of work, I have been able to develop a playable online multiplayer game for Android with minor problems that does not impact the overall experience of the game, together with a single player AR experience in which the placer can move freely, drop bombs and destroy boxes.

After all this work, I have come to the conclusion that Unity Multiplayer Framework is still under development and seems oriented to small demos rather than to fully games. Before the Unity 5.6 update [9], released on March of 1017, there were no serious network capabilities on Unity. The UNET framework has not been updated since then and even on the UNET forum, they have a section where they ask and encourage developers to leave and collect feedback for further updates.

6. BIBLIOGRAPHY

[1] Google Developers. (2018). Supported Devices | ARCore | Google Developers. [online] Available at: https://developers.google.com/ar/discover/supported-devices.

[2] Kotaku.com. (2018). [online] Available at: https://kotaku.com/game-developers-explain-some-of-their-favorite-ways-to-1798749279.

[3] Unity. (2018). Handling Non-Player objects - Unity. [online] Available at: https://unity3d.com/es/learn/tutorials/topics/multiplayer-networking/handling-non-player-objects.

[4] Feedback.unity3d.com. (2018). Unity Feedback - Multiple Player Objects (with server authority). [online] Available at: https://feedback.unity3d.com/suggestions/multiple-player-objects-with-server-authority.

[5] Unity Forum. (2018). Unity Multiplayer - Owning multiple player objects. [online] Available at: https://forum.unity.com/threads/owning-multiple-player-objects.353690/.

[6] Unity Forum. (2018). Why non-player objects that have client authority must have local player authority ?. [online] Available at: https://forum.unity.com/threads/why-non-player-objects-that-have-client-authority-must-have-local-player-authority.378582/.

[7] Unity Forum. (2018). Unity Multiplayer - How to sync a counter to check if all players are ready. [online] Available at: https://forum.unity.com/threads/how-to-sync-a-counter-to-check-if-all-players-are-ready.525918/.

[8] Library.vuforia.com. (2018). Ground Plane Supported Devices. [online] Available at: https://library.vuforia.com/articles/Solution/ground-plane-supported-devices.html .

[9] Ólafsson, L. (2018). Update on Unity Multiplayer, Current and Future – Unity Blog. [online] Unity Technologies Blog. Available at: https://blogs.unity3d.com/es/2017/03/17/update-on-unity-multiplayer-current-and-future/.

Docs.unity3d.com. (2018). Unity - Manual: Android environment setup. [online] Available at: https://docs.unity3d.com/Manual/android-sdksetup.html.

Docs.unity3d.com. (2018). Unity - Manual: Vuforia. [online] Available at: https://docs.unity3d.com/Manual/vuforia-sdk-overview.html.

Docs.unity3d.com. (2018). Unity - Manual: Multiplayer and Networking. [online] Available at: https://docs.unity3d.com/Manual/UNet.html.

Docs.unity3d.com. (2018). Unity - Manual: Unity User Manual (2018.1). [online] Available at: https://docs.unity3d.com/Manual/index.html.

Library.vuforia.com. (2018). Vuforia Library Documentation. [online] Available at: https://library.vuforia.com/.

Unity Forum. (2018). Multiplayer Networking. [online] Available at: https://forum.unity.com/forums/multiplayer-networking.26/.

Unity Forum. (2018). Unity Forum. [online] Available at: https://forum.unity.com/.

Unity. (2018). Multiplayer Networking - Unity. [online] Available at: https://unity3d.com/es/learn/tutorials/s/multiplayer-networking.

Wiki.unity3d.com. (2018). Unify Community Wiki. [online] Available at: http://wiki.unity3d.com/index.php/Main_Page.

7. ATTACHMENTS

Download APK:

https://www.dropbox.com/s/dkgahal2yzwajbs/BoomBoomGo.apk?dl=0

Gameplay Video: The recording software did not record audio, so the game's music has been added later. Audio effects are not heard.

https://vimeo.com/275499882

AR Video:

https://vimeo.com/273839361

Animations:

https://vimeo.com/273840105

AR Git Hub Repository. In this repository you can find the project from the beginning to the Augmented Reality implementation:

https://github.com/ZoroastrianMK/TFG

Final Git Hub Repository. A parallel repository to work with the final game version:

https://github.com/ZoroastrianMK/TFG_NoAr

Jester 3D Max model:

https://www.dropbox.com/s/2u9cfk0k3osg8m0/Jester.max?dl=0

PSD's of the project:

https://www.dropbox.com/sh/i8lv0vxeko52nty/AABnMBXlCjM8LV7_ZY3f_B1aa?dl=0

Image Gallery of the Art process:

https://www.dropbox.com/sh/rul5e9h8il6uey4/AAAa2jixbJM2eyvK4eNHLixza?dl=0