Universitat de Lleida

TREBALL FINAL DE GRAU

ESCOLA
POLITÈCNICA SUPERIOR
UNIVERSITAT DE LLEIDA
INSPIRING THE FUTURE

Estudiant: Albert Drudis Mola

Titulació: Grau en Enginyeria Informàtica

Títol de Treball Final de Grau: Joc 3D basat en Unity estil Tower Defense

Director/a: Francesc Sebé Feixas

Presentació

Mes: Setembre

Any: 2022

Abstract

In this project we have developed a Tower Defense style video game. The main idea of the game is that a path is given to the enemies to walk through which leads to your base. The objective is to place turrets and other structures to attack the enemies and prevent them from destroying the base.
This game is developed with Unity using as a programming language C# [1].
The main objectives of the project are:

- Creation of the **3D design** which will encase the world design, the non-playable characters and the structures.
- Implementation of the **artificial intelligence** to guide the entities through the map and the aim of the structures.
- Design of the **user interface** that will allow the player to interact with the game and receive the necessary information.

Also we will use and implement some of the features provided by the game engine that facilitate for a better development and design.

[1] C# is a general-purpose programming language.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Unity is one of the biggest platforms of app development that gives a lot of ease of use, a good amount of libraries and resources that help the developer.

There is also a personal motivation to the development of this project, as somebody who has spent countless hours immersed in video games from a very young age. The fact that I never developed one was something that has been in the back of my head for a long time.
This project is the perfect opportunity to dive in this world and start learning some of the most used technologies of game development and start gaining some experience.

1.2 Objectives

The project develops a small 3D game app. It will be developed in unity, using C# as the main coding language.
The main objective is to learn about Unity app development, how its components work and how to implement some of the many features given by the game engine to help developers and to have a more in depth perspective about the world of game development, mainly Unity developed apps.

As for the project the main focuses are to make a graphic design which is intuitive enough for the player to understand what is happening in the game. To implement the behaviours of the different components through scripts so everything works accordingly. And to make a comprehensive enough user interface so an inexperienced player could intuitively understand how to play the game and interact with its components.

2 State of the art

2.1 Game Industry

Nowadays with the use of smart TV's, computers, tablets and smartphones on the rise (Figure 1) as well as a wide variety of software platforms, such as Windows[2], Linux[3], macOS[4], Android[5] it is difficult to develop products for each different platform. So it is very important for the developers to use cross platform frameworks and technologies.
That is why Unity has become very dominant in the game creation world.

Unity offers a very good performance compared to other game engines like Unreal Engine[6] or CryEngine[7]. It is also very cheap for small companies or personal developers. Even big companies are charged quite a low cost fee.

The game industry keeps getting higher values every year, which make the predictions skyrocket, we are past the point where the industry is the biggest market in the media sector with earnings estimated at around 100 billion dollars generated in revenue at 2020 it is also important to break down how the shifting growth within the market is going, as the mobile is gaining over PC and console.

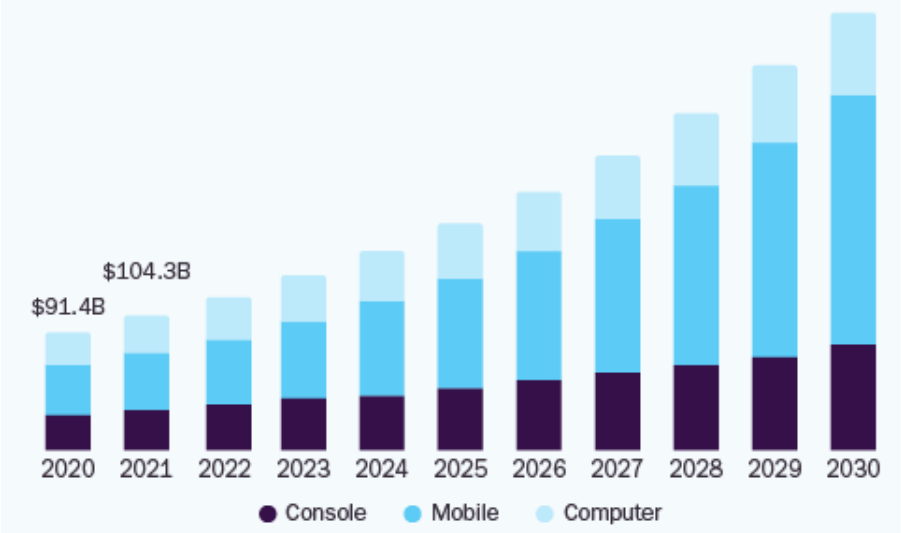

Figure 1: Game industry estimated earnings.

[2]Windows is a graphical operating system published by Microsoft.
[3]Linux is an open source operating system released under the GNU General Public License.
[4]macOS is a graphical operating system published by Apple
[5]Android is a mobile operating system based on a modified version of the Linux kernel.
[6]Unreal Engine is a game development engine developed by Epic Games.
[7]CryEngine is a multi-platform game engine developed by Crytek.

2.1.1 Mobile Game Industry

The mobile game industry has been gaining a big part of the gaming market since the iPhone launch getting close to the 50% of the yearly revenue.

It is expected to register 12.5% CAGR[8] during the 2020-2027 period.
Since mobile games can reduce stress and social connection, they played an important role during the COVID pandemic, positively affecting the industry.

Throughout the recent years the focus point has shifted and the industry started spending more and more in advertising attracting more users. To reduce the cost of development they started using the CTR[9] test, helping to get an early overview on what the user is focused.

As we can see in the Figure 2, mobile games has been growing at an amazing rate with respect to the rest of platforms.

There is also the fact that the smartphone hardware has increased greatly during the last couple of years. Processors with higher performance give the game developers the opportunity to increase the graphics fidelity, and long lasting batteries allowing users to spend time in games without worrying constantly about charging the phone.

[8]The compound annual growth rate (CAGR) is the annualized average rate of revenue growth between two given years

[9]Click-Through Rate is a metric used mainly for advertising that measures the number of clicks advertisers receive on their ads per number of times the ad is shown.

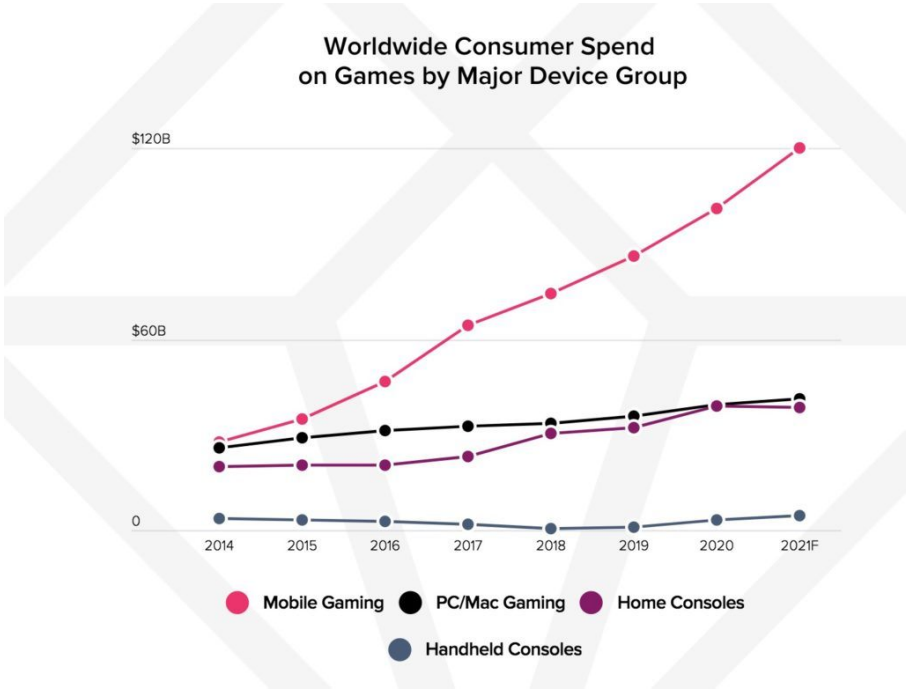

Figure 2: Worldwide consumer spending on games.

2.2 Unity

The main tool used for this project is Unity. It is mainly used to develop games and it is very popular due to its ease of use and the very powerful engine that gives developers a wide variety of features and functions to use during the creation process.

2.3 Game Engines Overview

A game engine is like an integrated development environment with reusable software components and a ready made suite of visual components which help make complex development tasks simpler by providing an abstraction layer.
These game engines are used by developers to create games for personal computers, consoles and mobile devices.

Game development engines are also being used in other fields such as training and military simulation due to their high fidelity.

2.3.1 Components of a Game Engine

- **Input:** Game engines provide support for a phletopn of input devices such as mouse, gamepad[10], touchscreen joysticks[11] etc. They provide many ways of handling inputs with tools like **events** and **polling**.

  Input **events** are actions captured by the computer, like a click of the mouse or a press of a key, that trigger certain parts of the code based on which was the pressed key.

  Polling is used to get values from the input devices like the tilt angle of a game stick or the coordinates of the mouse.

- **Graphics:** 3D assets are previously designed in external modeling and rendering programs, like Blender, and imported into the game engine. Then the engine implements effects like shadows and lighting to make them look real.

- **Physics:** The generation of physics is made by the **Physics Engine**. Physics Engines are a sub-component of the game engine used to simulate of real-life events like the movement of a rigid body, the behaviour of mass and velocity, fluid dynamics etc. These are very complex engines used when games portray any sort of real-life simulation.

- **Artificial intelligence:** Artificial intelligence or AI is playing an ever increasing role in game development. It is mainly used to create an environment where the player can get a feeling of immersion by creating non-playable characters that act according to their surroundings.

  The implementation of AI in games is usually made by specialised software engineers who create the scripts that are later included in the game.

- **Sound:** The audio is another sub-component of the game engine using rendering engines to control the sound effects, providing a software abstraction of the graphics processing unit using the multi rendering API[12] like Open-AL[13] or SDL audio audio[14].

- **Networking:** Nowadays most gaming engines have complete support for online connectivity to make the implementation of online multiplayer and social connectivity inside the game possible.

[10] A gamepad is a type of video game controller held in two hands, where the fingers are used to provide input.

[11] A joysticks is an input device that can be used for controlling the movement of the cursor or a pointer in a computer device

[12] API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other.

[13] As the abbreviation of Open Audio Library, it is a cross-platform API

[14] SDL acronym for Specification and Description Language is a program design and implementation language that is used to build real-time event-driven systems that involve parallel processing.

2.3.2 Unity Engine

Unity is mainly used to create 2D and 3D games using API C# or Mono as a scripting, it also implements a drag and drop functionality to make the development process more visual and easier.

Unity also provides support for a long list of platforms:

- **Mobile:** iOS[15], Android, tvOS[16].
- **Desktop:** Windows, Mac[17], Linux.
- **Web:** WebGL[18].
- **Console:** PlayStation[19], Xbox[20], Nintendo Switch[21], Stadia[22].
- **Virtual Reality and Augmented Reality:** Oculus Rift[23], PlayStation VR[24], Google's ARCore[25], Apple's ARKit[26], Windows Mixed Reality[27], HoloLens[28], MagicLeap[29], Steam VR[30], Google Cardboard[31].

[15]iOS is a mobile operating system created and developed by Apple
[16]tvOS is an operating system developed by Apple that runs the Apple TV digital media player
[17]Mac is a line of computers developed by Apple
[18]WebGL acronym for Web Graphics Library is a JavaScript API for rendering interactive 2D and 3D graphics.
[19]Playstation is a video game console developed by Sony.
[20]Xbox is a video gaming brand created and owned by Microsoft.
[21]The Nintendo Switch is a hybrid video game console developed by Nintendo
[22]Stadia is Google's gaming platform that lets you instantly play video games on compatible screens you already own.
[23]Oculus Rift is a virtual reality headset that is designed to connect to a high-powered PC to enable advanced computations and graphics renderin
[24]PlayStation VR is a virtual reality headset designed by Sony
[25]ARCore is Google's platform for building augmented reality experiences
[26]ARKit is Apple's platform for building augmented reality experiences
[27]Windows Mixed Reality is a platform which provides augmented reality and mixed reality experiences desingned by Microsoft.
[28]Hololens is Microsoft's headset for augmented reality that displays holograms that can be used to display information, blend with the real world, or even simulate a virtual world.
[29]Magic Leap is an augmented reality headset designed to amplify enterprise productivity.
[30]SteamVR is a tool for experiencing VR content on most VR headsets.
[31]Google Cardboard is ave handheld device that powers a virtual reality experience using almost any smartphone running Cardboard-enabled apps.

2.3.3 Unity Engine Market

As of today, the Unity game engine is the most dominant game development engine. As we can see it has taken over the game development market taking almost half of the shares i.e. Figure 3.

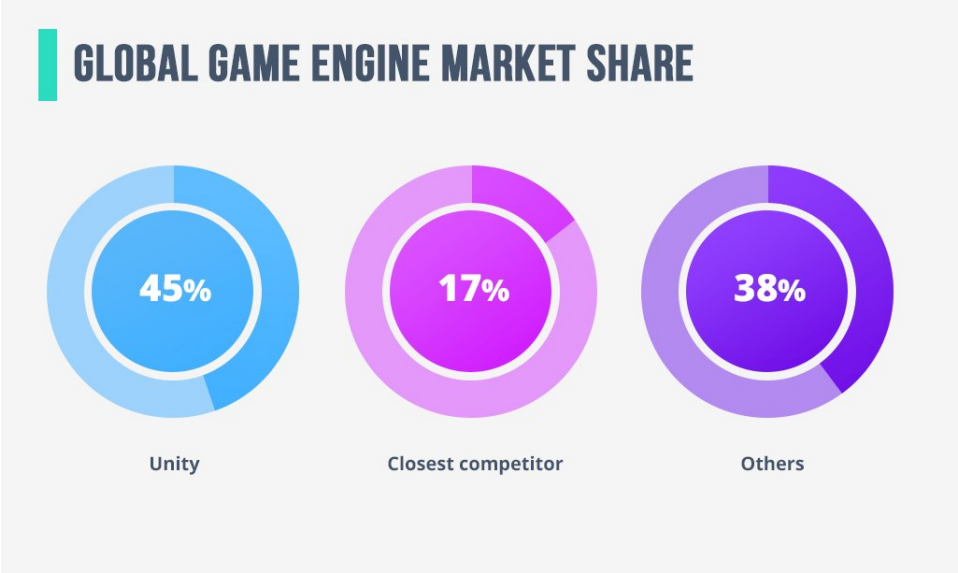

Figure 3: Worldwide shares on game engines.

3 Development process

This is a small project orientated mainly to the learning and integration of the technologies used in 3D games. Since the learning and the development will be simultaneous some rollbacks and corrections are to be expected and the requirements will be discovered and updated during the development.

Taking that into account the project development has been distributed between seven phases considering the limited time available.

3.1 Time management

To better understand how the focuses and the tasks of the project have been divided, we have a breakdown of the time in these two tables.

We first show a snapshot of how the main focuses have been divided (Table 1):

	Month 1				Month 2				Month 3			
Activity	Weeks				Weeks				Weeks			
	1	2	3	4	5	6	7	8	9	10	11	12
Learning	X	X	X	X	X	X	X	X	-	-	-	-
Implementation	-	-	X	X	X	X	X	X	-	-	-	-
Testing	-	-	-	-	X	X	X	X	X	X	-	-
Documentation	X	X	X	X	X	X	X	X	X	X	X	X

Table 1: Time distribution of the main focuses.

With that in mind, this will be the task break down (Table 2):

Task	Time allocation
Initial document and planification	Week 1
Learning Unity and the environment	Week 2
Design turret model and map	Week 3
Aiming system	Week 4
Enemy creation and map interactions	Week 5-6
Testing and fixing errors	Week 7-9
Documentation	Week 11-12

Table 2: Time distribution of the tasks.

3.2 Learning Process

The main focus of learning is how to use and implement the unity tools.

3.2.1 Hierarchy

The hierarchy window (Figure 4) is used to visualize every GameObject in a project such as models, cameras or prefabs.

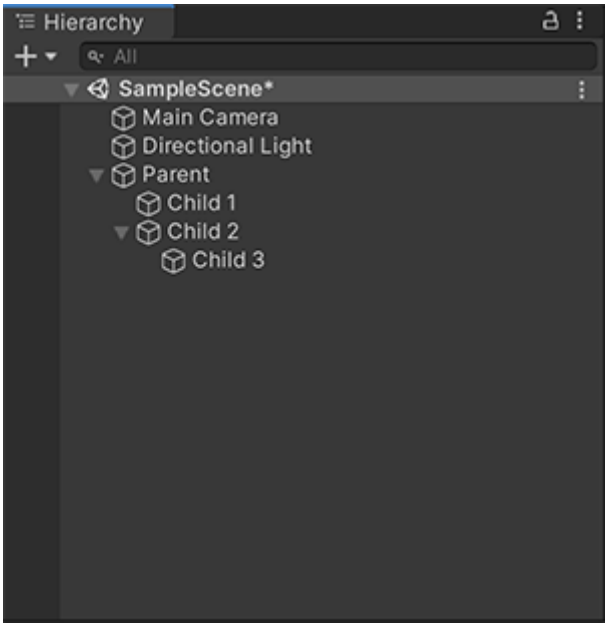

Figure 4: Representation of the hierarchy in the Unity editor inspector.

3.2.2 Inspector

The inspector window is used to visualize and edit most of the elements in the Unity Editor such as GameObjects, components, assets, materials, etc.

3.2.3 GameObjects

The **GameObject** is the main concept in Unity game development. Every object in Unity is a GameObject. It is the fundamental object that Unity uses to represent anything from a scenery to a character or a light source.

Its main purpose is to act as a container for **Components** which implement the actual functionalities.

3.2.4 Components

Components are elements containing the actual functionalities within a GameObject. They contain certain editable properties that define the behavior of the GameObject. In the inspector, a list of the attached components can be visualised and edited (Figure 5).

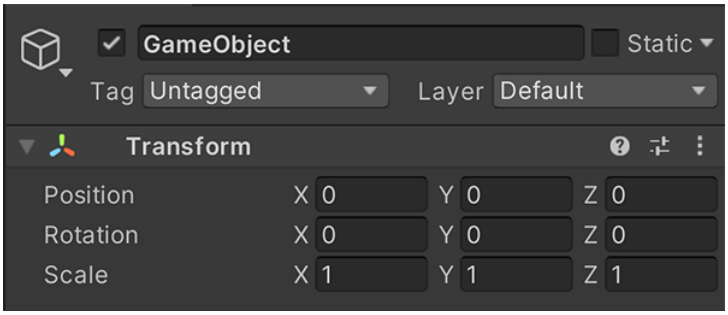

Figure 5: Representation of a component screen in the Unity inspector.

3.2.5 Transform

A **transform** is the component which defines the position, rotation and scale of an object. Every transform can have a parent, that would allow to make a position relative to another transform. This creates a hierarchy that values like sale and rotation can be inherited.

3.2.6 Scripts

The **script** is the component that allows to link code implemented functionalities to a GameObject. In the inspector the developer can also visualise certain properties of the object as well as how they are modified in real time during the execution (Figure 6). That proves to be really helpful as it can be used as a debugging[32] tool during the development process.

[32]The process of identifying and removing errors from computer hardware or software.

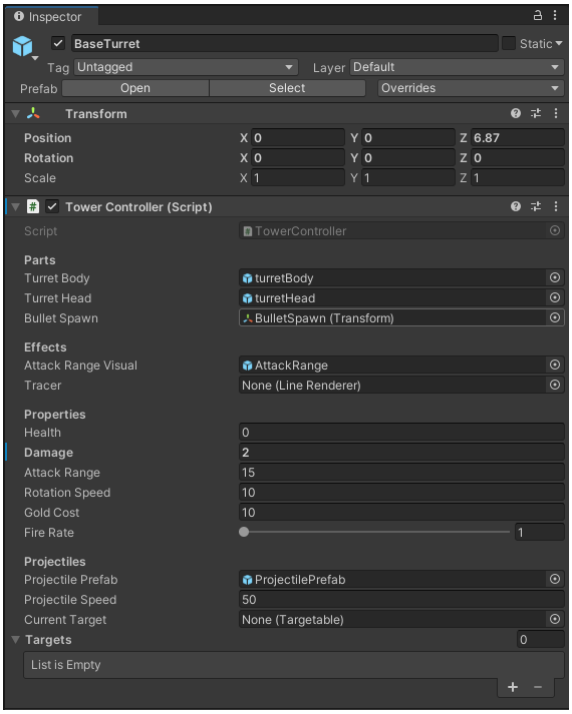

Figure 6: Representation of a script component in the Unity editor inspector.

3.2.7 Coroutines

In Unity, a **coroutine** allows you to execute tasks across frames. In most situations, a method runs and returns the control to the calling method when it finishes. Hence, the execution must take place within a single frame update.

So a coroutine can be used as a process running in the background while other executions take place.

3.2.8 Vector3

Vector3 are a representation of vectors and points in 3D space. Its main purpose is to represent 3D positions and directions in the game.
It supports basic arithmetic operators like sum, difference, product, etc.

3.2.9 Colliders

Colliders are components that define the physical limits of the GameObject body. So a collision between GameObjects will be defined by the shape of their colliders instead of the visual **mesh**[33].

3.2.10 Primitive objects

The Unity engine editor has an array of 3D models of different shapes that can be used to model the created objects as well as objects that can be created with external modeling software.
From the editor, the following objects can be created:

- **Cube:** The default **Cube** is a white cube with 1x1 sides. The texture is defined so that it is repeated on each side (Figure 7).

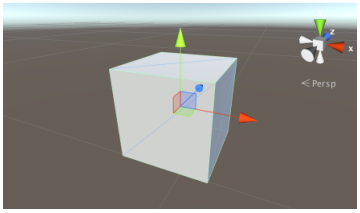

Figure 7: Representation of a cube.

- **Sphere:** The default **Sphere** primitive is a white sphere with a 0.5 radius and textured so that the texture image wraps around the sphere (Figure 8).

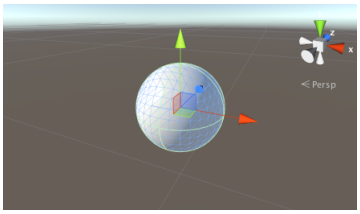

Figure 8: Representation of a sphere.

- **Cylinder:** The default **Cylinder** primitive is a cylinder of 0.5 diameter and 2 height, It is textured so that the image wraps around the tube and then it is repeated separately on the two flat ends (Figure 9).

[33] A mesh consists of triangles arranged in 3D space to create the impression of a solid object.

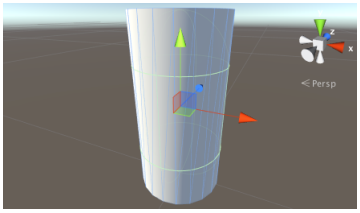

Figure 9: Representation of a cylinder.

- **Capsule:** The **Capsule** primitive is a cylinder with hemispherical caps at each end. It is one unit in diameter and two units high with a 0.5 radius on each cap. It is textured so that the image wraps around once and the edges pinch together at the apex of each hemisphere (Figure 10).

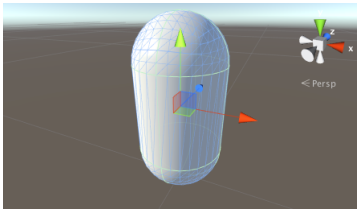

Figure 10: Representation of a capsule.

- **Quad:** The default **Quad** primitive is a thin square divided into two triangles. Each edge is one unit long (Figure 11).

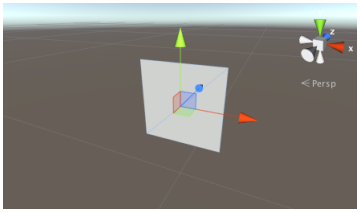

Figure 11: Representation of a quad.

- **Plane:** The default **Plane** primitive is a flat square with edges being ten units long. It is divided into 200 triangles (Figure 12).

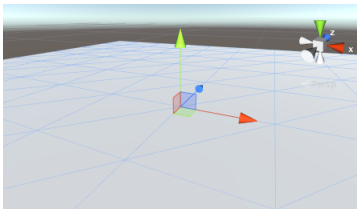

Figure 12: Representation of a plane.

4 Entities

Entities are objects that interact with the game, that can be either player inputs or other entities. Each entity has its own properties as well as behaviours.

4.1 Turrets

The turrets are the main entity of the game, they are the only way for the player to interact with the game to target and kill enemies.
Turrets have a system to target enemies within a certain range. While an enemy is inside the specified range the turret will automatically face the enemy and start shooting (Figure 13). Every turret has the following properties:

- **Attack Range:** The attack range is the distance of effectiveness of the turret, once an enemy is within that range the turret will aim at the enemy and attack it.
- **Rotation Speed:** How fast the head of the turret can turn to aim at an enemy.
- **Damage:** The damage of the turret is the amount of damage an enemy will take from every shot of the turret.
- **Fire Rate:** The fire rate imitates how fast the turret shoots. More exactly the time that has to pass between two shots.
- **Projectile Speed:** How fast a projectile shot by the turret moves along its trajectory.

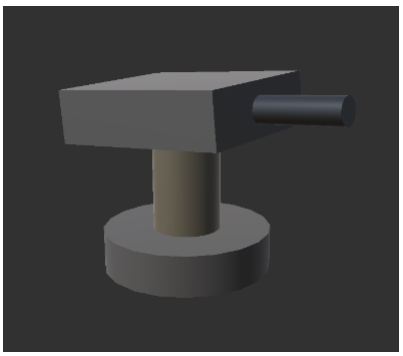

Figure 13: 3D model of the turret.

4.2 Projectiles

A projectile is shot by a turret and it impacts the enemy and causes some damage to it (Figure 14).
Projectiles have the following properties:

- **Damage:** The damage will be inherited from the turrets damage.
- **Projectile Speed:** How fast a projectile shot by the turret moves along its trajectory.

Figure 14: 3D model of a projectile.

4.3 Enemies

The enemies are the main threat of the game. They follow a previously set path until they get to the base. This path is set by points set in order. The enemies will go to a given point and once reached they will move to the next. Once an enemy reaches the base it will deal damage to it (Figure 15). Enemies have the following properties:

- **Health:** The amount of damage an enemy can take. Once it reaches zero the enemy is destroyed.
- **Speed:** How fast the enemy can travel through the map.
- **Damage:** The damage it will inflict once it reaches the base. An enemy is destroyed upon reaching the base and dealing damage to it once.
- **Gold Value:** How much gold the player receives upon destroying the enemy.
- **Way Point:** The next point of the map where the enemy is going to.

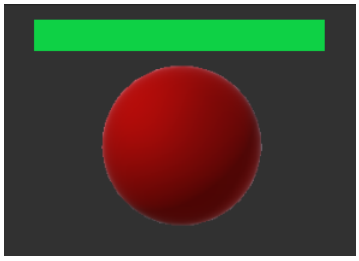

Figure 15: 3D model representation of an enemy.

4.4 Base

The base is the structure the enemies are moving to. The player's goal is to protect it by killing the enemies before they can reach it (Figure 16). The base has the following properties:

- **Health:** It is displayed on the screen. Once it reaches zero the base is destroyed. Therefore the game is over.
  The player has to protect it by creating turrets that kill the enemies before they can reach it.

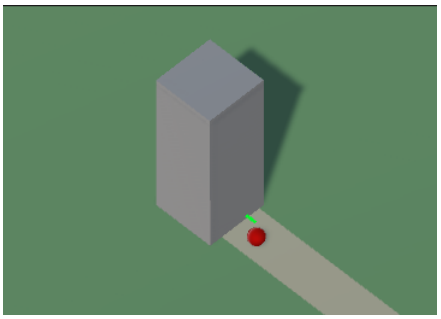

Figure 16: 3D model representation of the base .

5 Interactions

When a behaviour includes two or more different game objects an interaction between them occurs. This is also true when a player interacts with a GameObject. Here is a detailed explanation of the relevant interactions of the game.

5.1 Aiming

When an enemy enters the range of a turret, the turret will calculate the direction it has to face to point directly at the enemy. Then it will rotate the head of the turret until it reaches that position.

5.2 Projectile impact

When a projectile impacts an enemy. It causes a certain amount of damage to that enemy.That amount of damage will be given by the turrets damage at the moment that the projectile was fired. Once the damage has been dealt the health bar of that enemy will decrease that amount of the damage. Then the projectile will be destroyed. If upon impact the health of the enemy becomes zero or lower, the enemy is killed.

Since enemies carry a certain amount of gold with them, upon death that amount of gold will be given to the player and the counter of player gold will be increased by that amount, then the enemy will be destroyed.

5.3 Enemy reaches base

When an enemy reaches the base it will deal damage to it. That damage will depend on the enemy's strength. When damage is applied to the base, the main health bar is decreased by that.

5.4 Turret placement

When the player selects a turret from the overlay button, all the buttons will be disabled. At that point, the selected turret will appear on top of the mouse and will follow it. Once an available point of the map is clicked, the turret will be placed and enabled there.

All turrets have a certain price, so if the player does not have enough gold to build it, the button appears to be disabled. If a turret is successfully built, the corresponding amount of gold is decreased from the player gold balance.

6 Overlay

The overlay is divided in three sections, the **top left**, the **top right** and the **bottom center** (Figure 17):

- In the **top left** we have the gold display. This indicates how much gold we have at any given time. It is updated every time an enemy is killed or when a structure is purchased.
- In the **top right** corner we have the main health bar with a number indicating exactly how much health we have left. This is the health of the base, meaning that once it reaches zero the game is over.
- In the **bottom center** of the screen we have the structure placement buttons. These buttons will display the kind of turret assigned to each as well as the amount of currency needed to construct it. Once the button is pressed the border will darken indicating that a turret is about to be placed. A red border will appear if we do not dispose of enough currency to build the structure, at which point the button will be disabled until we gather enough.

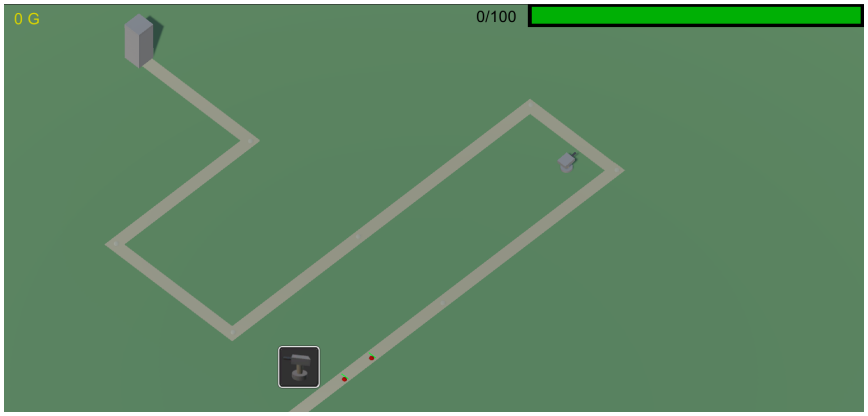

Figure 17: Representation of the game overlay with the gold and health at the top, and the turret placer at the bottom.

7 How to play

The game will give the player some time to prepare its strategy before starting. A small amount of gold will be given at the start to put some structures before the first wave of enemies comes. Once the player is ready the first wave of enemies will come and the player will have to use the gold given by the slain enemies to keep putting structures to defend the base.

7.1 Controls

The player will be able to interact with the camera with the **arrows** to look around the map. The camera can also be zoomed in and out with the **wheel scroll**.

7.2 Selection

This will be the main way for the player to interact with the objects in the game. By clicking on a structure, a menu will be displayed which will allow to interact with the selected structure.
If an enemy is selected it will display its current stats.

7.3 Leveling a Turret

When clicking on a turret, a small menu will appear. The the player will be able to increase the turret power in exchange for gold.

7.4 Strategy

The best strategy for beating a tower defense is resource management. Knowing when and where to spend the resources is key to the success of the game.
Since the path that the enemies follow is always the same there will be certain places that allow a turret to cover more space. Choosing better places for a turret gets more value out of the resources spent in its building.

8 Software Development

The software development in this project has consisted of two main parts: one being the visual part including the creation and modeling of 3D objects. The other has been devoted to the coding of the behaviour of each object and the interactions between them.

8.1 Base turret model

The base turret model was made manually by assembling and editing cubes and cylinders as it can be seen in Figure 13.

8.2 Turret Behavior

The main behaviour of the turret is to aim at the enemies when in range to then fire at them

8.2.1 Enemy Detection

When a turret is created a coroutine is called which will be constantly looking for enemies.

This is achieved by creating an **OverlapSphere**[34] around the turret being the range of the turret its radius. Then the enemies are found by doing a **Select** for searching GameObjects that have the enemy's Component and then transform the result to a **List** for better management.

This way the coroutine will constantly detect and store all the enemies inside that OverlapSphere.
For the sake of performance since the enemies do not have very high speeds, this will refresh every half a second instead of every frame.

[34]Method from Unity Physics library that computes and stores colliders touching or inside the sphere.

8.2.2 Aiming

Once we have stored all the enemies within the turret range, if the enemy list is not empty, a coroutine will be called to do the aiming.

The first thing needed to aim is to know the enemy position relative to the turret position. For that, the Vector3 position of the turret is subtracted from the Vector3 position of the enemy.

With the relative position it is possible to calculate the aiming rotation.
By using the **RotateTowards**[35] method. This is done by providing the transform.forward of the turret, the previously calculated target direction and the turrets rotation speed, which is calculated by multiplying the rotation speed property by the **Time.deltaTime**[36]. A Vector3 is obtained with the rotation that the turret will have to perform.

To apply this rotation to the turret, the rotation has to be transformed into a **Quaternion**[37]. For that purpose the **LookRotation**[38] method is used. The resulting Quaternion will be applied to the turret's transform rotation property.

8.2.3 Shooting

Once the turret is aiming at a target, if it was not already shooting, it can start shooting.
To do that, as long as there are enemies in range another coroutine will be called.

The shooting method will first instantiate a projectile, which is a prefab previously created. To give the sense of realism, it will be placed inside the turret's barrel. So as for the projectile to go in the same direction as the one the turret is aiming, the turret's rotation property is applied to the projectile's rotation property.

Then the necessary information is stored in the projectile. The target will be the one that the turret is aiming at. The speed will be given by the turret, as the turret is the one doing the shooting, and the damage that it will inflict on the target is also given by the turret.

[35] Method from Unity Vector3 that rotates a vector towards a target.
[36] The interval in seconds from the last frame to the current one.
[37] Quaternions are used to represent rotations in Unity.
[38] Quaternion method that creates a rotation with the specified forward and upwards directions.

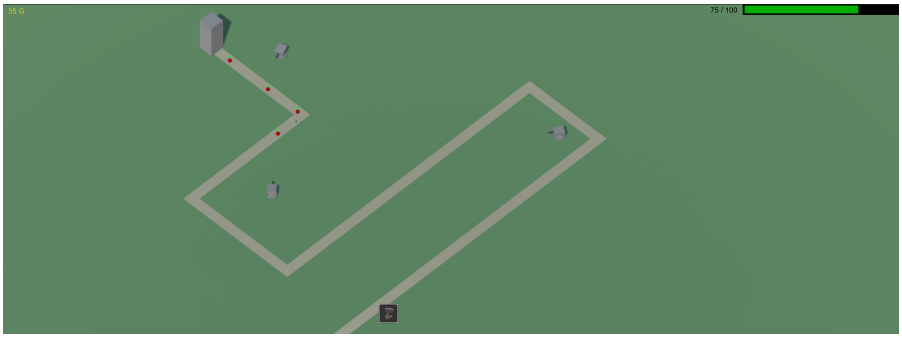

Figure 18: Live game of the turrets shooting at the enemies while enemies attack the base.

8.3 Projectile Behavior

A projectile moves form the turret to the enemy and, on impact, it applies some damage to the enemy.
To do that, as soon as the projectile is created a coroutine is called that manages the process.
First, a fail-safe is implemented so that in case that anything went wrong the projectile will be destroyed after five seconds. This is done to prevent possible performance issues in the case that projectiles did not collide against my object.

8.3.1 Movement

Projectiles are moved by a loop where the **Translate**[39] method is called giving as parameter the Vector3.forward multiplied by **Time.deltaTime** multiplied by the projectile speed.

8.3.2 Trajectory adjustment

After that, a second fail-safe is implemented. Since some of the projectiles would not make contact with the target, a small adjustment to the projectile trajectory is made so that it never misses the target. This is done by using the method **LookAt**[40] providing the enemy's position as a parameter.

8.3.3 Impact calculation

To calculate the impact of the projectile with the target, the distance between both objects is calculated at every frame using the method **Distance**[41] by giving the position of the enemy and the position of the projectile. When the distance reaches is we have an impact.

Upon impact the damage of the projectile is inflicted onto the target by subtracting

[39] Moves the transform in the direction and distance of translation.
[40] Rotates the transform so the forward vector points at the target's current position.
[41] Returns the distance between a and b.

it from it's health parameter.Then, the projectile is destroyed.

When this happens, the health of the enemy is checked so that if it reaches zero or below it is destroyed. That will trigger a method which will increase the player's current available gold by the value of the enemy's goldValue parameter.

8.3.4 Impact adjustment

Sometimes a projectile would move across its target within a single frame so the distance between both objects would not be zero, to fix that an **Epsilon Distance**[42] was implemented giving enough margin of error so that the projectile could never go through the target.

If the distance between the two objects is smaller that the Epsilon Distance we consider it as an impact. The value of the Epsilon Distance is always bigger than the distance that a projectile can travel during a frame, assuring that it can never move across its target.

[42] A small margin of error within which if a projectile is found it wold count as a hit.

8.4 Enemy Behavior

Enemies walk though a set path until they reach the base, and apply damage to it.

8.4.1 Pathing

The path is constructed with a set of way points. Each of these points contains as a property the next point in the path, creating a **linked list**[43] in this way.

As soon as an enemy is created a coroutine is called to make it move.
The enemy will go to the first way point, once it reaches it, it gets to the next point and so on.

8.4.2 Movement

To make the enemies move first the position of the way point is stored in a Vector3 data structure. Then the distance is calculated with the **Distance** method by giving the position of the enemy and the position of the way point. With that distance the duration of the travel is calculated dividing it by the enemy's speed.
While the traveled time is smaller that the calculated travel time, the enemy will keep moving.

The movement will be created using the **Lerp**[44] method. It will be called in a loop while the time traveled is smaller than the total travel time. The method Lerp works by providing the current position of the target, the position of the enemy and an interpolation ratio, which is calculated by dividing the time traveled by the total duration of the travel. Then, after calling Lerp, the time passed will be calculated and the loop will keep going.

With the only way of knowing if the enemy reached the target way point being that the traveled time is smaller than the total travel time another overlook was found, similar to the one with the projectile going past the target. So as to avoid that, instead of simply subtracting the duration from the time passed, we use **Min**[45] method, by giving it the deltaTime and the calculation of the time passed it will return the smallest of the two values, so it is more accurate. This is used to avoid overshooting.

[43] A linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list.
[44] Linearly interpolates between two points.
[45] Returns the smallest of two or more values.

8.4.3 Projectile impact

As for the health of an enemy when a projectile impacts on it a method is called which subtracts the amount of damage of the projectile to the enemy's current health. When that happens, this is to be reflected in the enemy's health bar. So the bar is updated according to the remaining health of the enemy after the impact.

When an enemy finally reaches the base it behaves similarly to a projectile upon impact. The damage of the enemy is inflicted upon the base's current health and the enemy is destroyed.
Health bar management is done by modifying the **offsetMax**[46] parameter. The x parameter of the offsetMax maximum value is multiplied by the percentage of current health.

[46]RectTransform property: offset of the upper right corner of the rectangle relative to the upper right anchor.

8.5 Overlay Behavior

8.5.1 Overlay Visualisation

The overlay has three main parts. The gold display is the one showing the player how much gold is currently available. This value is updated every time an enemy dies, adding the amount of the enemy's worth to the available gold or when a turret is placed, subtracting the gold price of the turret from the currently gold balance.

The health bar has two components. The first shows the current and the maximum health, and a health bar representing the amount of current health. The health bar management is done by modifying its **offsetMax** parameter. The x parameter of the offsetMax maximum value is multiplied by the percentage of current health.

Finally, there are the turret placing buttons. These are used by the player to place turrets on the field.

8.5.2 Turret placing

When clicking on a turret the game will change to turret placing mode. This action will disable other turret buttons and a model of the turret will be placed on top of the mouse position, moving with the mouse, to help visualise where the turret is to be placed.

When the turret placing button is clicked, the position of the mouse is recovered in a **Vector3** using the **mousePosition**[47]. With that position, a **Ray**[48] is generated using **ScreenPointToRay**[49]. This will create a ray going from the camera going trough the mouse. To know if the ray is touching the ground, the **Raycast**[50] method is used. This will return true if the ray intersects with a Collider. If the ray is intersecting with the ground, the point on the ground where the ray is intersecting is calculated by getting its coordinates.
To do that, the **RaycastHit**[51] returned by the Raycast method is used and a new Vector3 is generated by getting the x and z coordinates of the RaycastHit with the y coordinate being zero.

[47] A Unity's Input method that returns a Vector3 of the current mouse position in pixel coordinates.
[48] A ray is an infinite line starting at origin and going in some direction.
[49] A Unity's Camera method that returns a ray going from camera through a screen point.
[50] Returns true if the ray intersects with a Collider, otherwise false.
[51] Structure used to get information back from a raycast.

Having the intersecting point on the ground a temporary instance of the turret is placed on it.
To make the turret model follow the mouse continuously this method is placed inside the Update. Since it will be called every frame, it will look as if the turret moves as with mouse.

While in turret placing mode, when the left mouse button is pressed the turret is placed at the spot where the temporary turret model was being visualized.
This is done with **GetMouseButtonDown**[52] instantiating a turret at the position where the temporary turret was.

[52]A Unity's Input method that returns true during the frame the user pressed the given mouse button.

9 Conclusions

The idea of this project was simply to make a Tower Defense because I always enjoyed the style of the game. It turned out to be a good decision because it made me discover Unity. This has been a big factor for me, since Unity has proven to be a great tool for beginners because after a bit of investigation the layout becomes very intuitive. I was also able to get started in C# which is a very commonly used programming language.

There have been many problems during the development, most of them during the development of the behaviour of the objects, because not only the right tool for the job has to be used, but it also has to be implemented correctly.

The result of the project was quite good, getting a fully functional version of the envisioned game.

Now it can be improved upon, some of the first things to add is an enemy progression system, controlling how many enemies come at each wave as well as how strong they are.
Also more enemies can be added with different characteristics, like being faster or with more health.
Another improvement would be to add turrets with special effects with different sprites and animations.

10 Bibliography

https://unity.com/our-company
https://bsic.it/in-depth-analysis-of-the-gaming-industry-1/
https://www.grandviewresearch.com/industry-analysis/video-game-market
https://www.mordorintelligence.com/industry-reports/mobile-games-market
https://www.studytonight.com/3d-game-engineering-with-unity/game-engine
https://finance.yahoo.com/news/game-engines-market-2022-leading-120700809.html
https://www.innovecsgames.com/blog/unity-2019-update/
https://www.businessofapps.com/news/consumer-spending-in-apps-40-higher-than-before-pandemic/
https://docs.unity3d.com/*
https://www.mulesoft.com/resources/api/what-is-an-api