CT6COPRE REPORT

Tudor-Cristian Ion (825241)

Coordinator teacher: Mattew Higgins

Observation: Most of the photos are using gifs, which are going to be presented just as pictures, if you see this report from a pdf or word application. Thus, here is the link to the exact same document, but in its original state, as a google doc in which the gifs can be seen properly

Number of Words: 3845

Table of Contents

Introduction

The first idea of the project was to implement a VR game mainly because developing on this platform was never experienced by me and it was in a good timing with another project which required VR development as well. Unfortunately, due to the unexpected beginning of this year, having in the main cast, the notorious Covid-19 which excluded the possibility of using the VR headsets available in the university. Thus the project idea was changed towards another unexplored field which is mobile development.

A mobile application is defined by *a software program that runs on a mobile phone* (Cambridge Dictionary, 2020). Before the advent of mobile phones, the only mobile way to communicate was by car phones which were invented based on a phone network invented in 1947 by an engineer at Bell Labs.

Fig 1. Carphone, 1947

Unfortunately at that time the technology and infrastructure necessary for running such a network didn't exist therefore various limitations such as limited simultaneous usage, thus waiting lists which varied from 1 to 5 years were beginning to be formed to schedule the usage of the car phones(Techinsider, 2017) .Thanks to Martin Cooper and Motorola, the future was introduced in 1973 when they introduced Motorola DynaTac 8000X (the first handheld cell phone). (TechInsider, 2017)

Fig 2. Martin Cooper & Motorola DynaTac 8000X, 1973)

The first so called smartphone was announced by IBM in 1993, called IBM Simon. This device differentiated itself by the rest of its competitors because it introduced the first phone with a touch screen and software applications embedded and from there the mobile development witnessed an exponential evolution being direct proportional with the evolution of infrastructure. (Tech, 2015)

Fig 3. (IBM Simon, 1993)

This big impact of the hardware technology has made possible the manufacture of a phone which is structured on up to 16 GB of RAM and 512 GB of storage (Samsung Galaxy S20 Ultra, 2020). This for me personally seemed something hardly possible keeping in mind that this phone has the specifications required to play the most recent AAA games such as Red Dead Redemption 2(Rockstar, 2019)

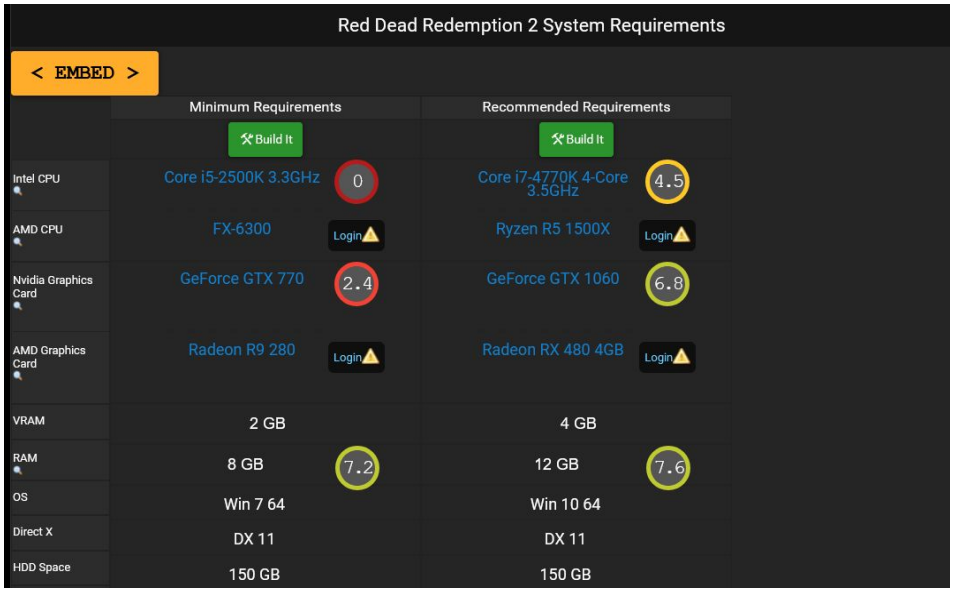


Fig 4. Red Dead Redemption System Requirements (GameDebate, 2019)

Furthermore, having better hardware specifications to work with has brought the attention to a high percent of developers, based on Statista, the numbers of applications available on the play store in December 2009 was 16 000, since then until, 20th of March 2020, there are 2 870 000 apps available(Statista, 2020).

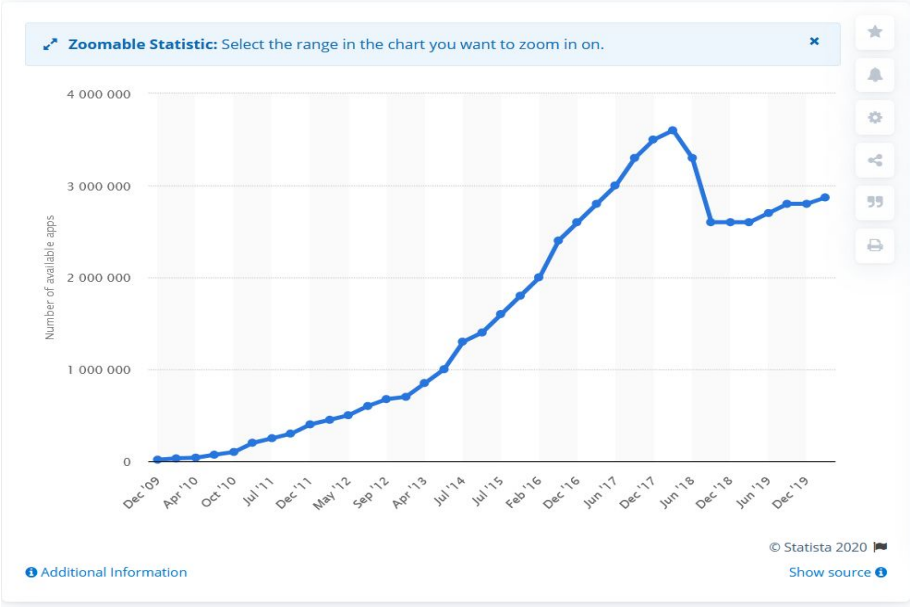


Fig 5. Number of apps on play store (Statista, 2020)

In *Mastering Android Game Development with Unity*(Siddharth Shekar, Wajahat Karim, 2017) there are presented the basics into developing games for the Android platform in there it can be seen the development of the Android operating system and the specific improvements that each update bring

Introduction to Android Game Development with Unity3D

2.2.x	Froyo	USB tethering, Hotspot support, Adobe Flash, Voice Dialling	8	May 2010
2.3.x	Gingerbread	New Copy/Paste, WebM, NFC, Front Camera	9, 10	December 2010
3.x	Honeycomb	3D Graphics, Redesigned UI, Video Chatting, Bluetooth tethering, 3G, 4G	11, 12, and 13	February 2011
4.0.x	Ice Cream Sandwich	Virtual buttons, Face Unlock, Native Camera Features, Face Detection, Android Beam, Wi-Fi Direct	14 and 15	October 2011
4.1 - 4.3	Jelly Bean	Expandable Notifications, Google Now	16, 17, and 18	July 2012
4.4	Kit Kat	Major Design Interface Update, Translucent Status bar, Immersive Mode, Wireless Printing	19 and 20	October 2013
5.0	Lollipop	Redesigned UI with Material, Lock Screen Notifications, Guest mode, Battery Saver mode	21	October 2014
6.0	Marshmellow	Fingerprint security support, Doze mode for battery saving, App standby mode, Enhanced App permission	23	October 5, 2015
7.0	Nougat	Multi window view, VR support	24, and 25	August 22, 2016

Fig 6. Android OS Upgrades Mastering Android Game Development with Unity(Siddharth Shekar, Wajahat Karim, 2017)

In the same book, a game engine is defined as a *software framework designed for the creation and development of video games.* Game engines became more and more popular as every company would create one as they see fit based on their game designs. For example Ubisoft made use of its DUNIA Engine for creating the last series of their notorious Far Cry franchise, *Far Cry 5*(Ubisoft, 2018). This engine is bended towards the necessities of the game design, having dynamic weather, volumetric lighting, dynamic fire propagation, Non-scripted enemy A.I and much more(Giantbomb, XXXX). Of course these types of game engines are available only within the companies grasp and is not for public use, fortunately Unity3D comes to the rescue with functionality to sustain all aspects of games ranging from simple 2d platformers to massive online role-playing games(Learning Unity Android Game, Thomas Finnegan, 2015).

Having experience with this great game engine and a market with over 30 million devices waiting to play games, seemed the idea of making the project structured towards android platform in Unity engine, the most viable option thus the game development plan began anchored on this idea.

While having the right tool and the right market to deploy and develop a game seems all you need to start developing, there is one more step necessary, knowing the limits that occur when working on this platform:

- Screen Compatibility
    - Not all the phones have the same screen size, so is essential to scale everything accordingly, fortunately Unity3D, has special functionality to achieve this
- Battery
    - Battery life is the most important aspect of the mobile user experience (Android Developers, 2020). The most problems regarding the battery are being aroused by bad design of the process necessary to run the app. For instance, the android documentation describes the design, every developer should know when developing on android, *Lazy First* in which the user is first making the game run as he sees fit and from there, the testing begins and is looking for ways to reduce and optimize the operation time. (Android Developers, 2020)
- Background execution limits
    - In the documentation it states that since Android 8.0, the system started to preserve more battery life by receiving location updates less frequently thus affecting the dependent APIs.(Android Developers, 2020)
- Lower memory
    - The obvious fact that on average, android phone is not having the same specifications

With all this in mind, the project development stage ought to commence.

Design and Planning

First Idea

As stated in the introduction, the first idea was to implement a VR game but due to Covid-19 and the lack of necessary hardware, the project changed its platform to Android.
Having no experience working on the Android platform made its presence severely in the design and planning of this project. The first idea of the project after changing its platform was to implement a 2D platformer similar to games such as Celeste (MattMakesGames, 2018) but with different goals. In this game the player is a pizza in a pixel art world and its objective is to gather the requirement ingredients on the map and to jump itself in the oven, the player performance is based on the amount of time necessary to complete the level, the score varying itself from 1 pizza slice(poor) 2 pizza slices(mediocre), three pizza slices(perfect)

Fig 7. Celeste Level Layout Example (Celeste, 2018)

First Idea Development Process

The first step consisted in creating the artwork. All of it was based on PixelArt due to my low level of drawing skills.

Artwork

First, the main character was designed to be half of pizza.

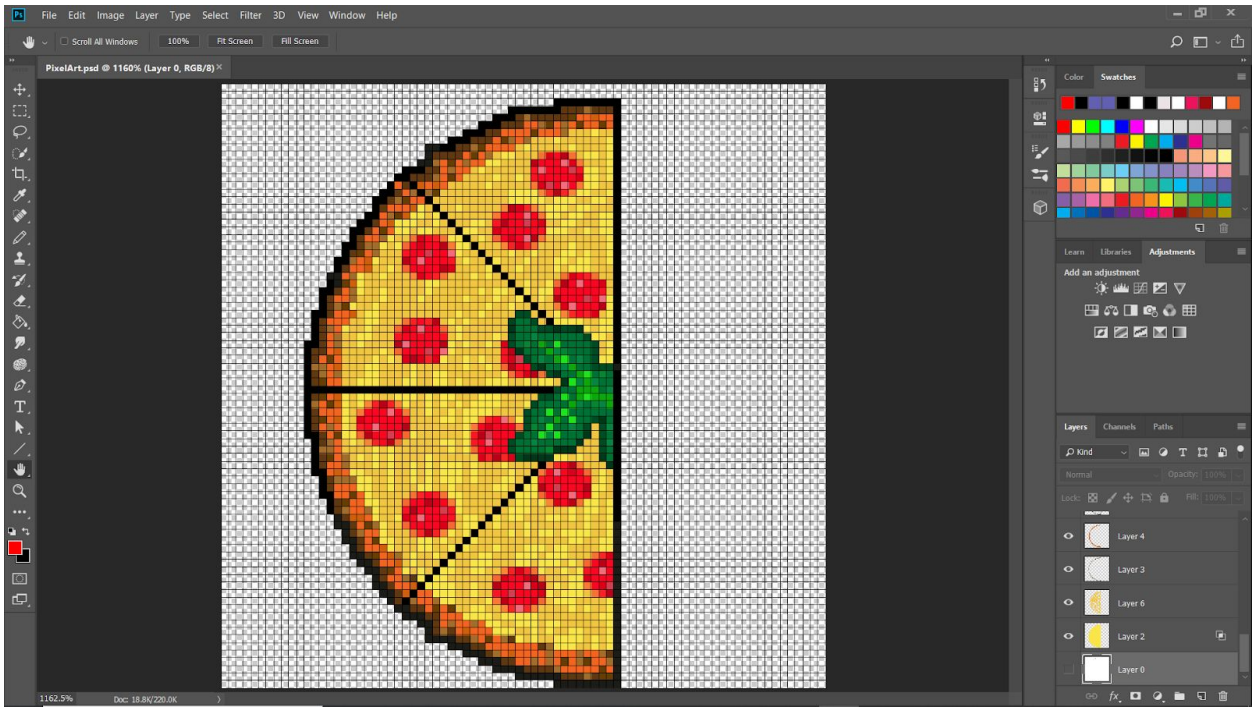


Fig.8 Player Artwork

Having this as the first level layout:

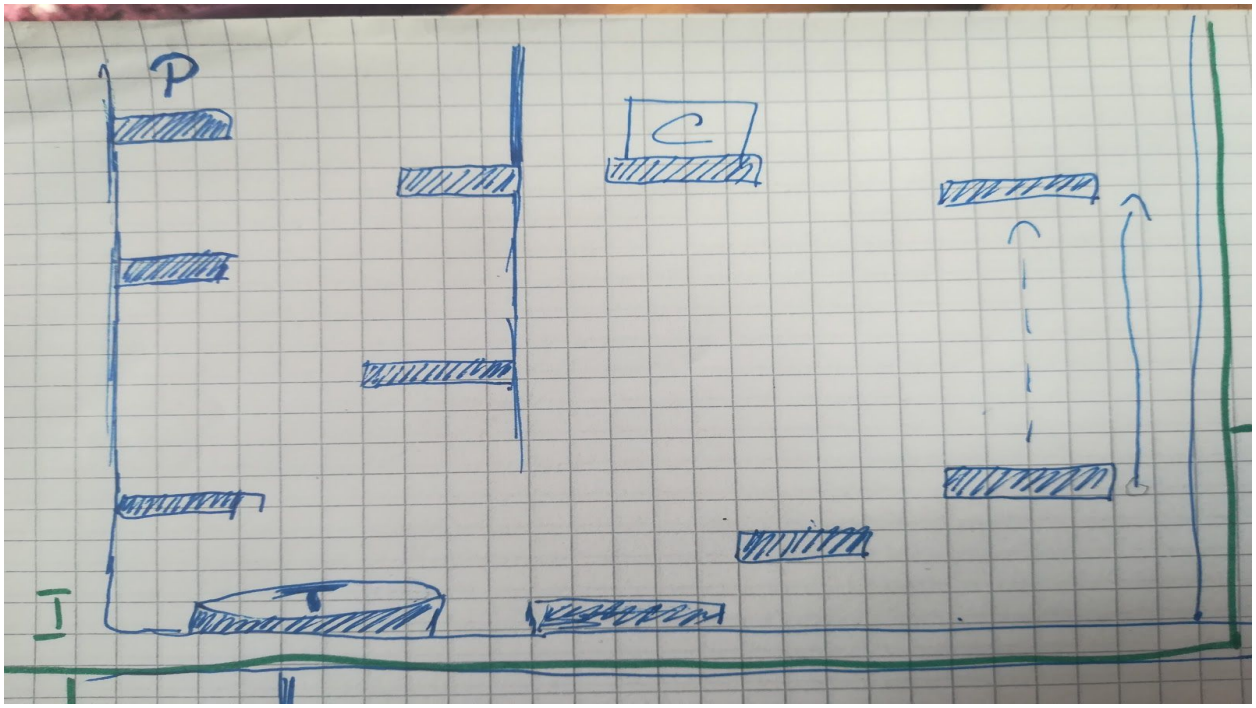


Fig.9 First Level Design

The next step consisted of making the ledge on which the player can jump on.

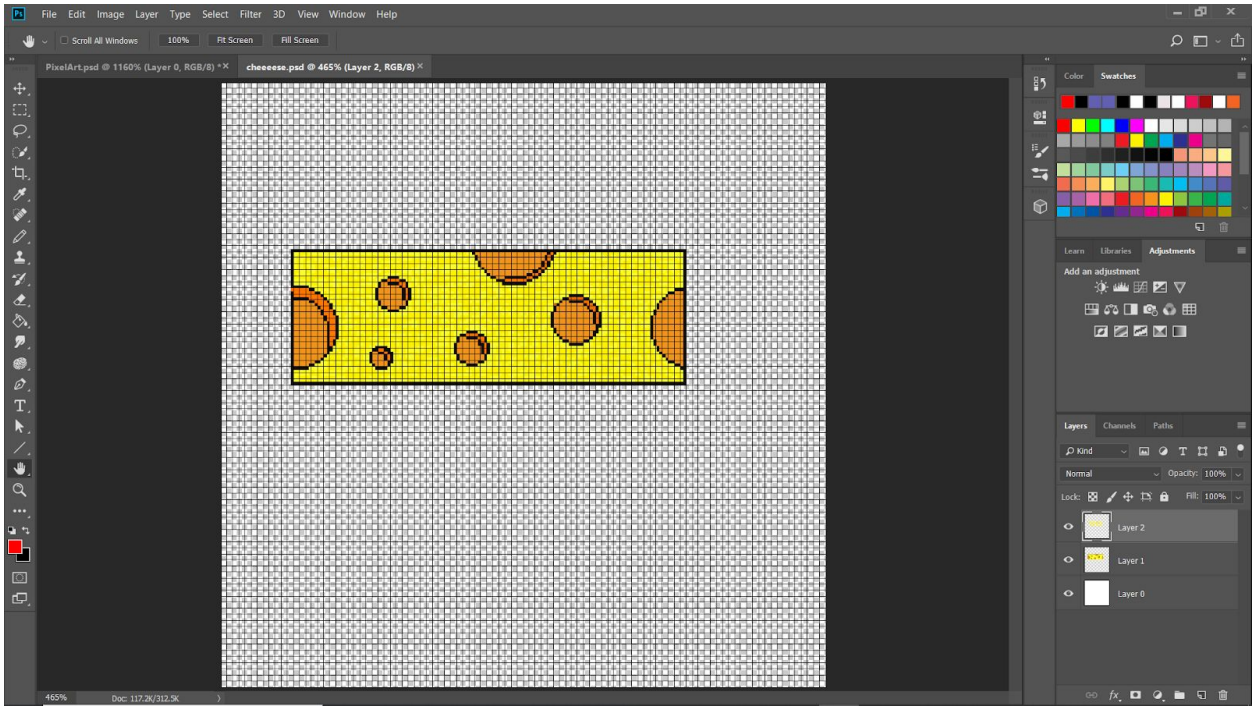


Fig.10 Ledge Design

Followed by a brick oven in which the pizza has to reach to successfully finish the level

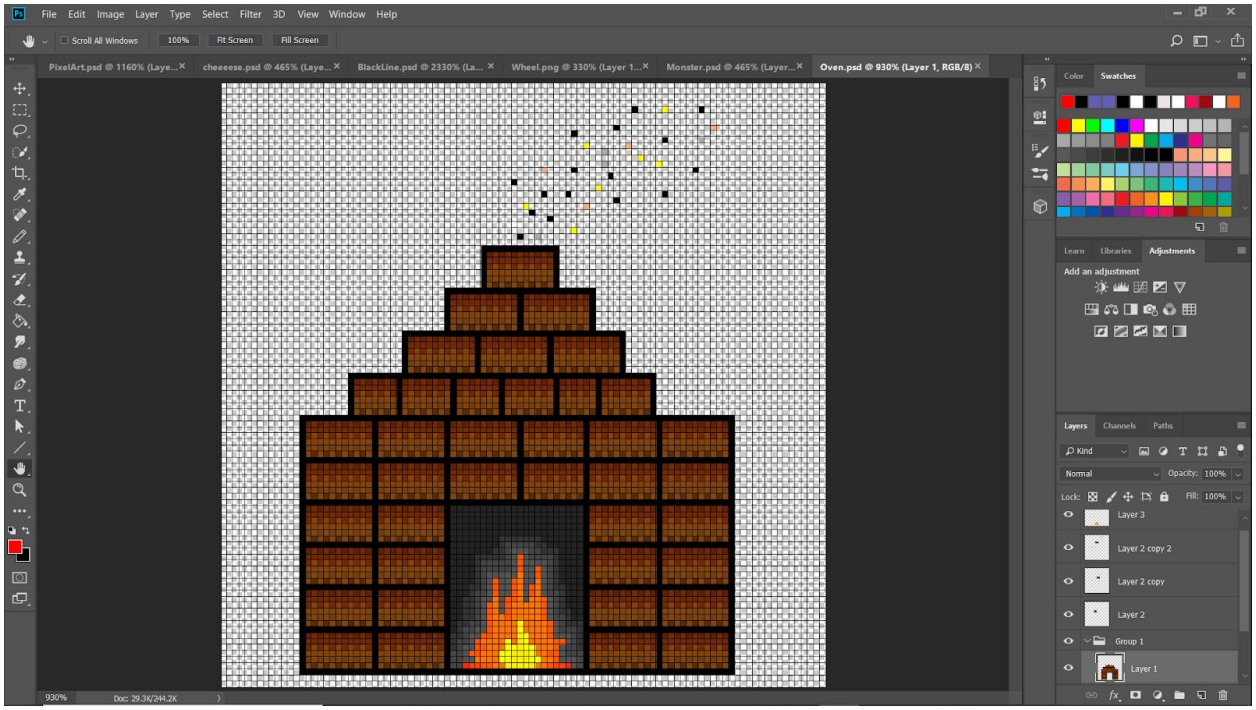


Fig 11. BrickOven

And two pieces of food that the player has to collect before jumping inside the oven:

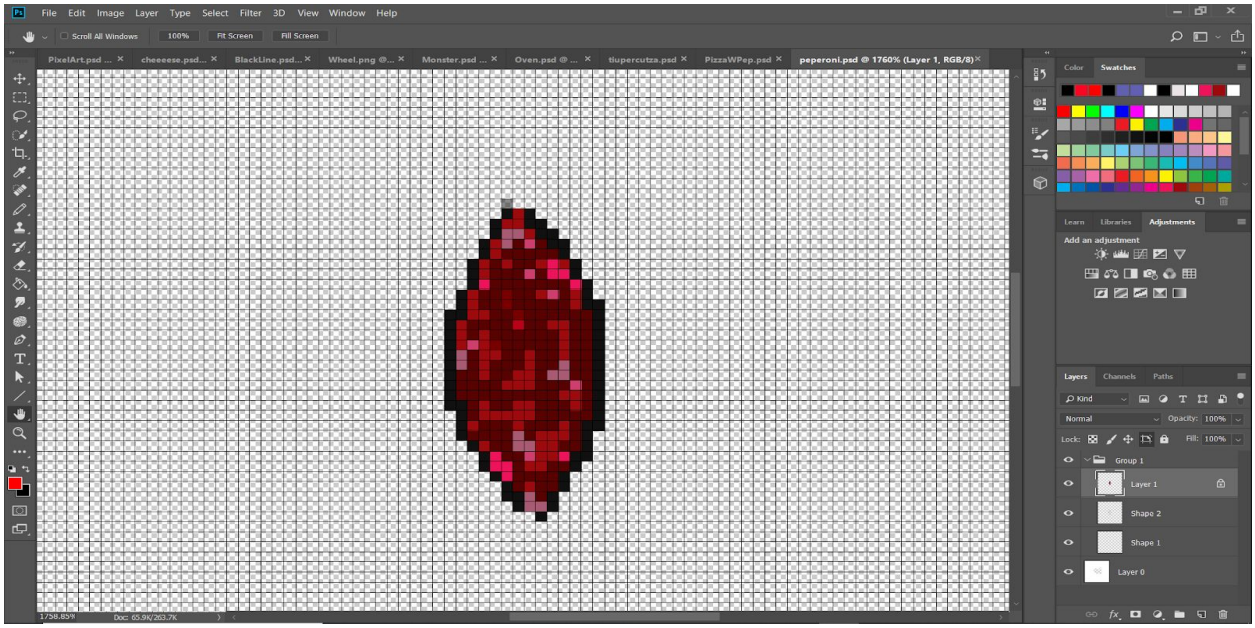
- Pepperoni
- 

  Fig 12. Pepperoni

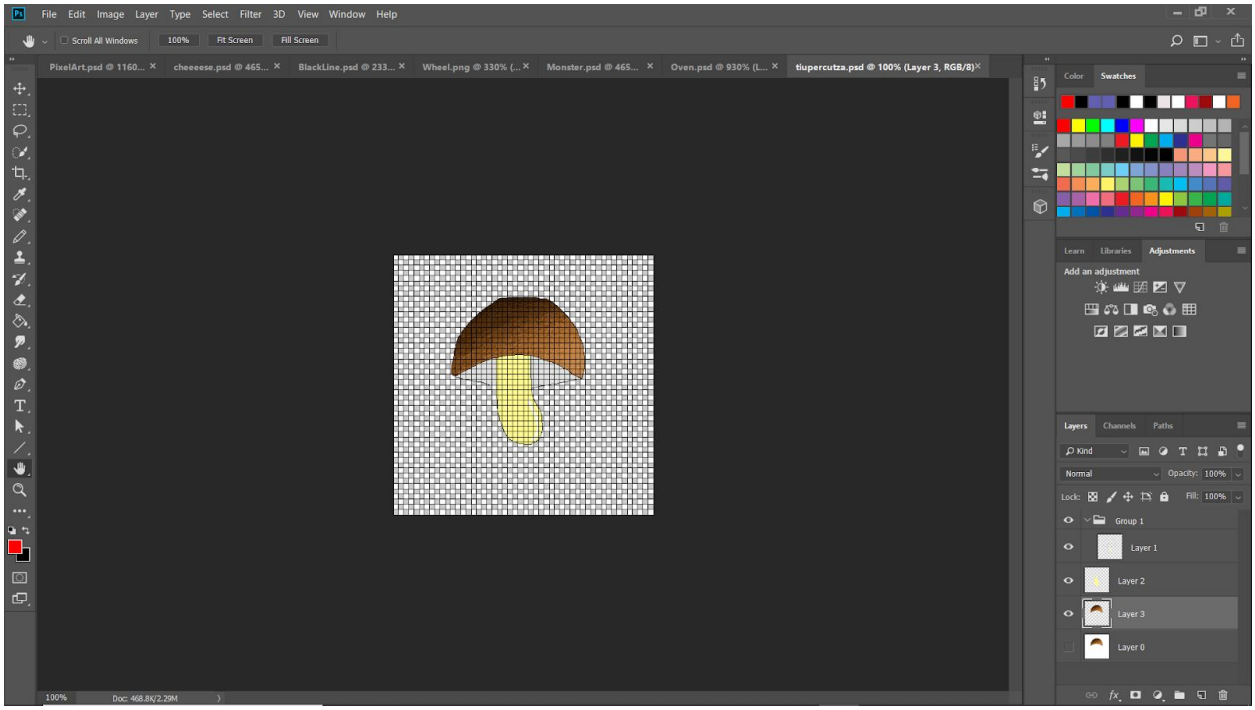
- Mushroom
- 

Fig 13. Mushroom

As well as a wheel used as an obstacle which was animated to move up and down

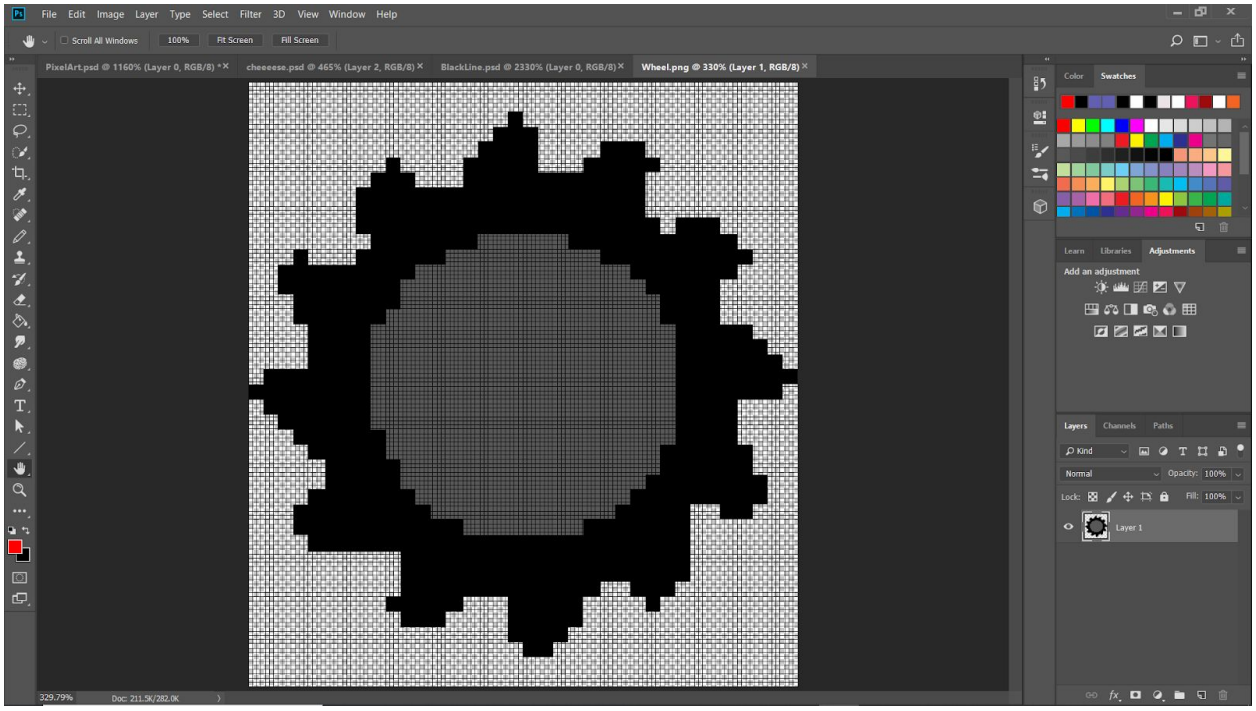


Fig 14. Wheel

All of this hooked to unity resulted the following result:

First idea gameplay

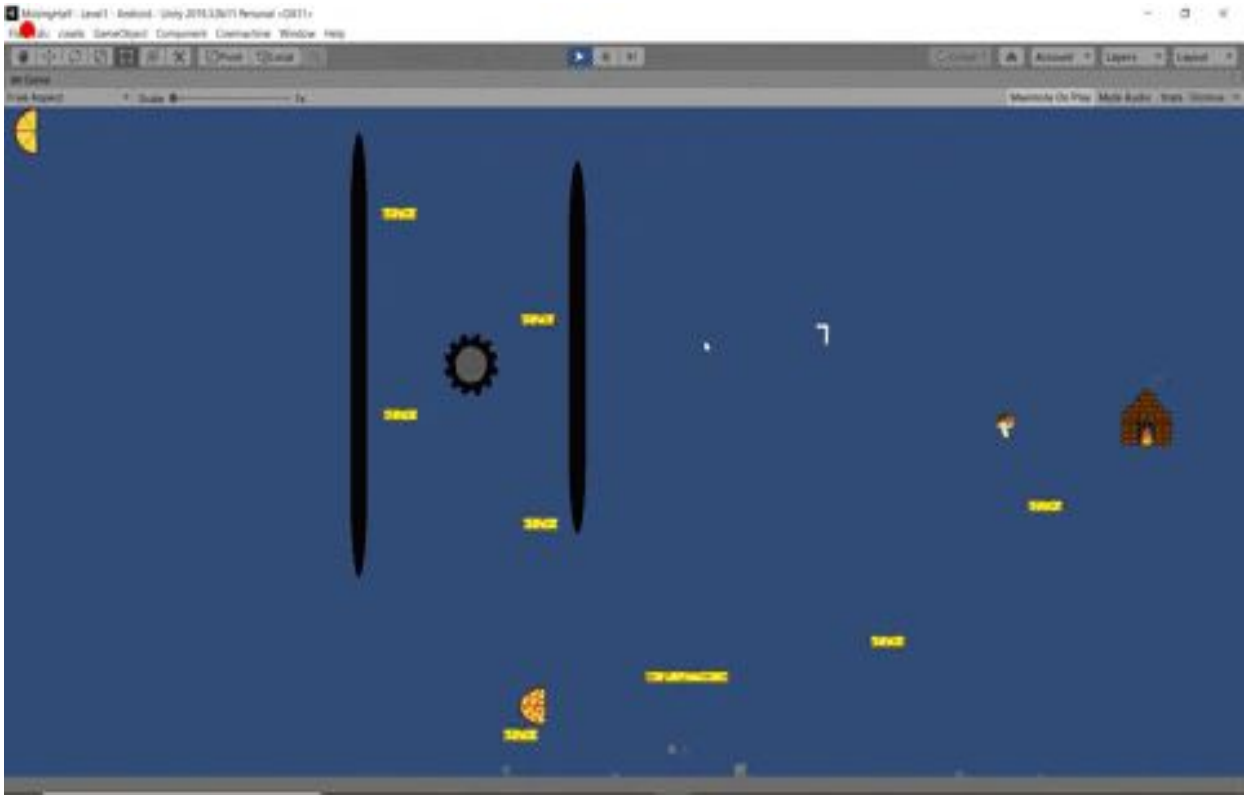


Fig 15. First Idea Level Gameplay

Then the development plan continued by creating a doodle for the enemy

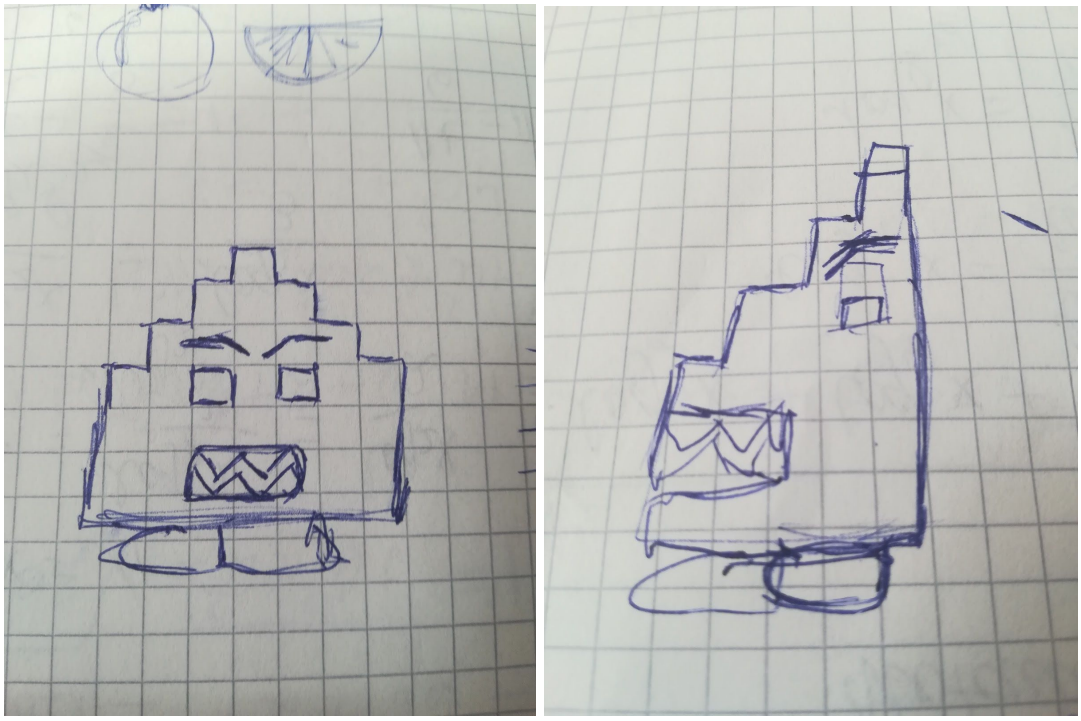


Fig 16. Monster Concept Art

Which ended up with this form:

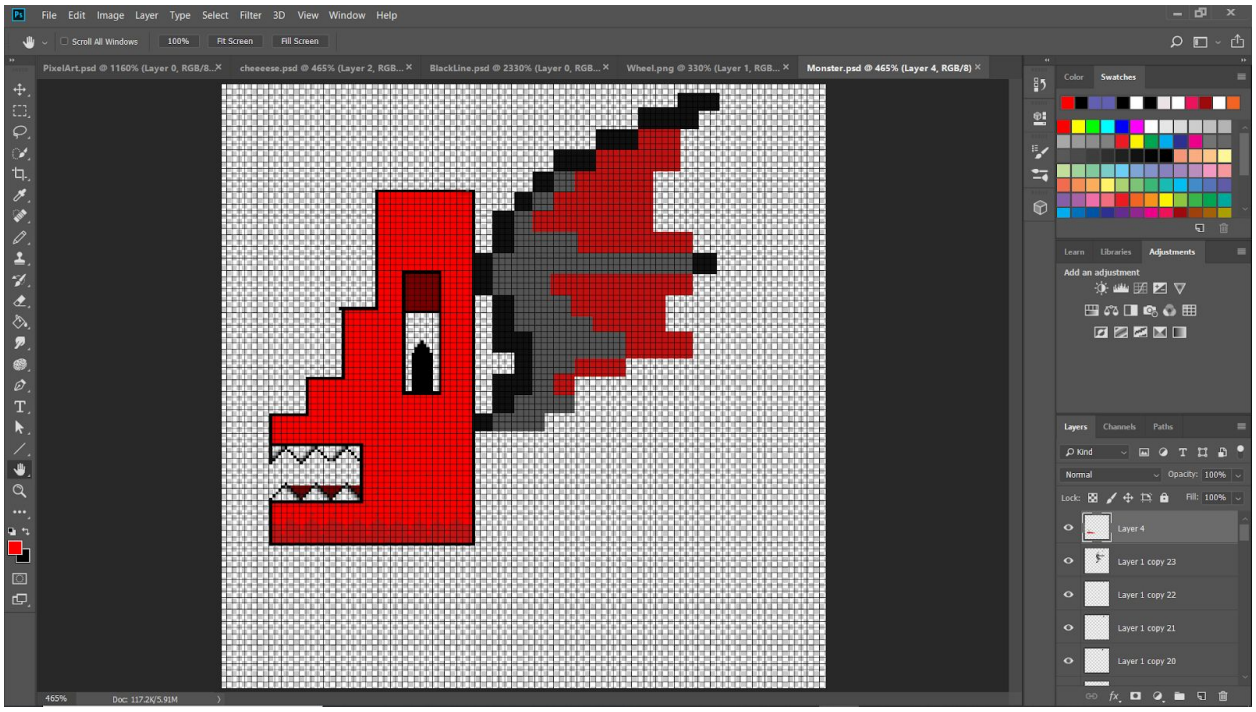


Fig 17. Monster Final Concept Art

First Idea A* Pathfinding

On this enemy was attached the A* pathfinding technique required from the free version of Aron Granberg's asset in the unity asset store, to make it able to track the player in the scene.

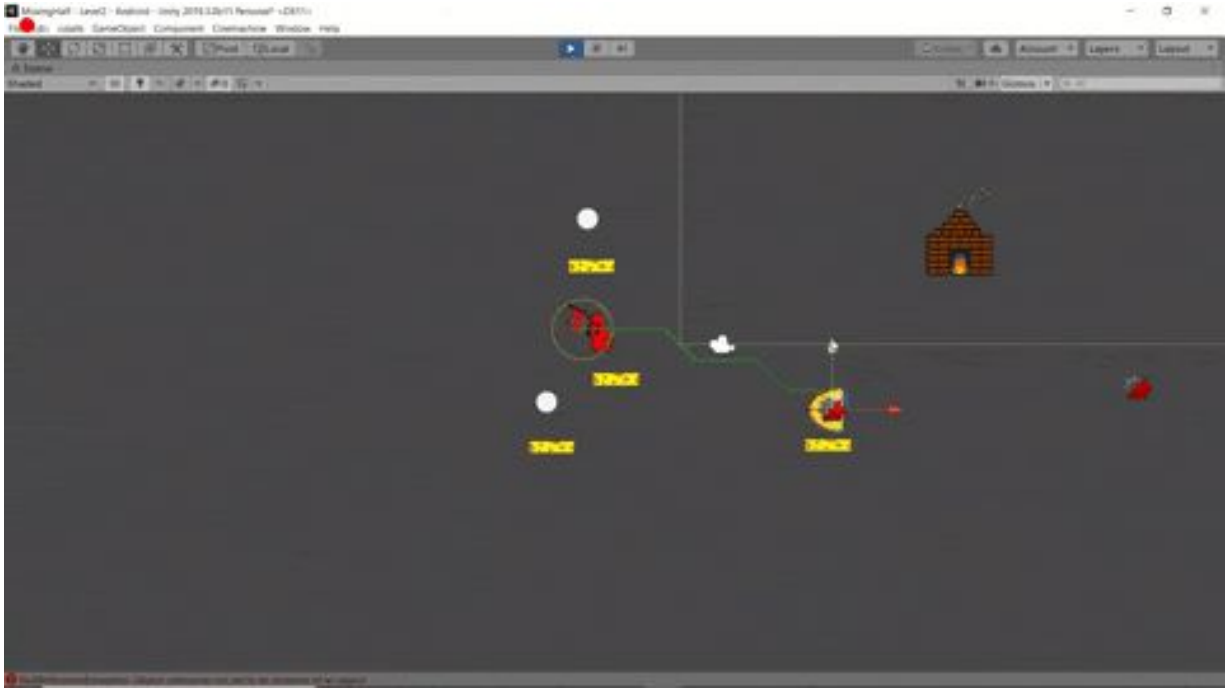


Fig 18. First Idea Enemy Pathfinding Demonstration

Idea Change

After many days of being stuck on the level design and having just one working prototype level, the game idea changed drastically to a clone of Doodle jump (Lima Sky, 2009) based on the time left to finish the project as well as the rest of the projects.

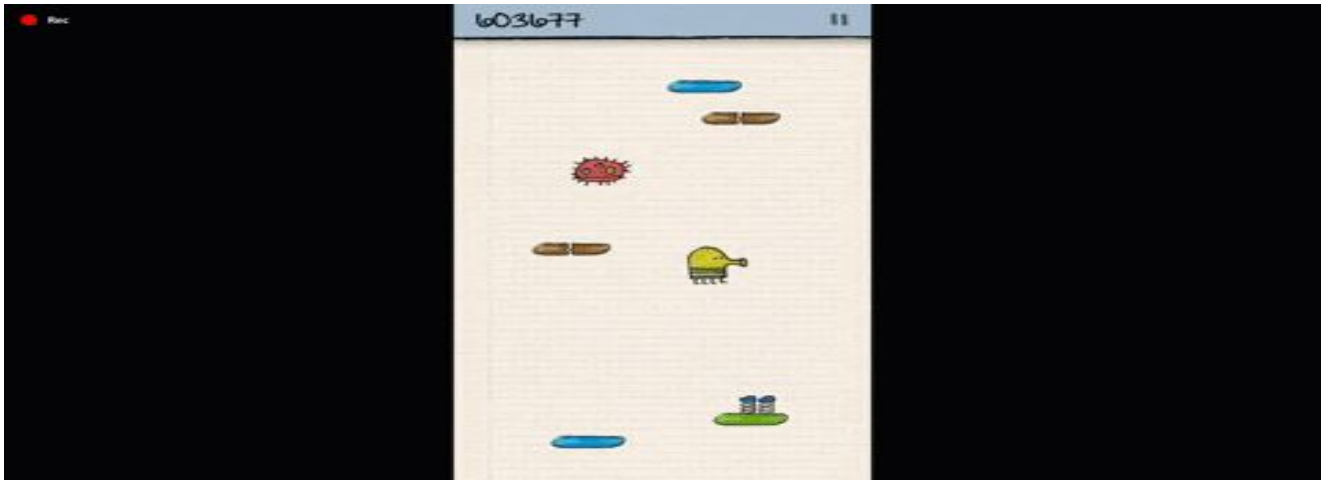


Fig 19. Doodle Jump Gameplay

Development Steps of the new idea

Movement

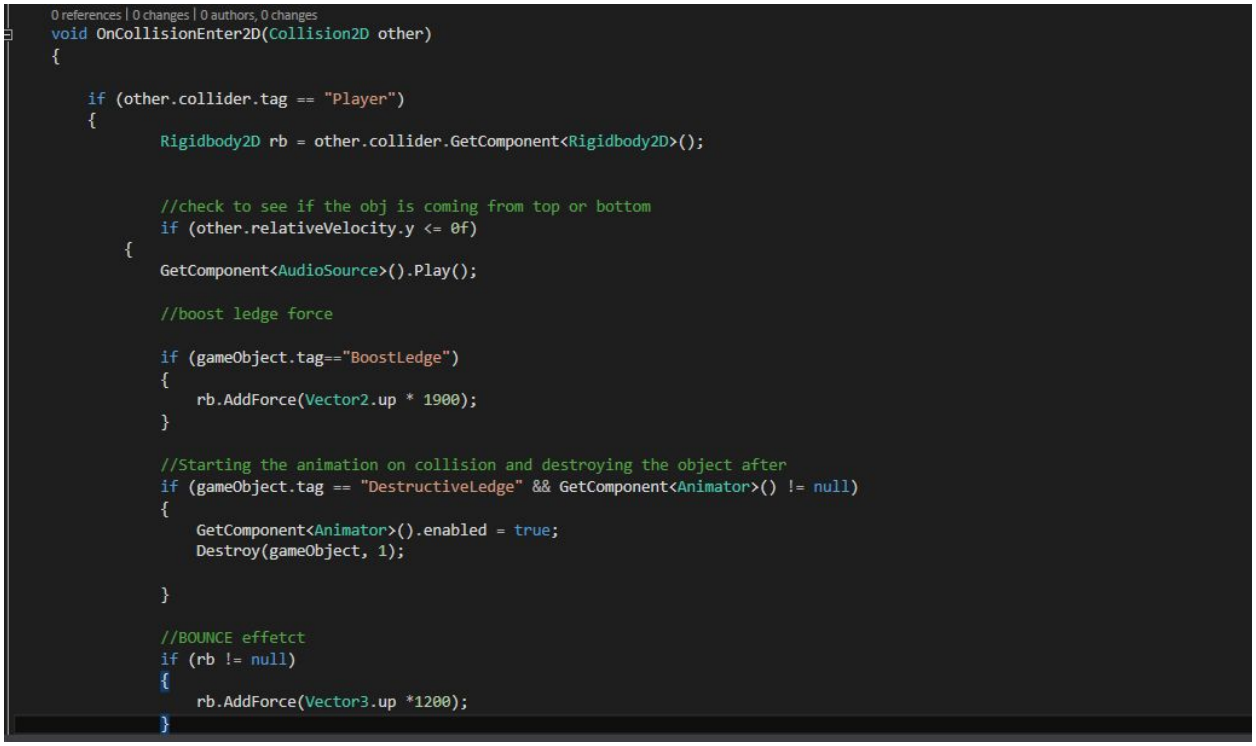
First, the movement of the player was made using the velocity component of the rigid body upon which a velocity vector was passed consisted of the Input acceleration on the x which gives the x value of the accelerometer of the phone and multiplied by a scalar named speed. The velocity vector would update itself every frame in the update function and is passed to the velocity of the rigid body in the fixed update which is a special function implemented by Unity3D for performing physics on objects.

```
0 references | 0 changes | 0 authors, 0 changes
void Update()
{
    //velocity vector
    dirX = Input.acceleration.x * player.speed; ;
}

0 references | 0 changes | 0 authors, 0 changes
void FixedUpdate()
{
    //left -right accelerometer movement
    if (dirX!=0)
    {
        Vector2 velocity = rb.velocity;
        velocity.x = dirX;
        rb.velocity = velocity;
    }
    //keyboard movement for testing
    else
    {
        Vector2 velocity = rb.velocity;
        velocity.x = player.movement;
          rb.velocity = velocity;
    }

}
```

Fig 20 Player Movement Code

Bounce Effect

The next step consisted of making the bouncy effect which was done using a special component from Unity called *Platform Effector 2D* which is providing proper functionality for platform style effects such as one way collision, removal of side-friction/bounce etc (Unity Manual, 2019). This component combined with an edge collider 2D has made possible the bounciness effect as well as the one way collision.The bounce effect was achieved by adding a force on the Y axis of the rigid body.



```
0 references | 0 changes | 0 authors, 0 changes
void OnCollisionEnter2D(Collision2D other)
{

    if (other.collider.tag == "Player")
    {
            Rigidbody2D rb = other.collider.GetComponent<Rigidbody2D>();


            //check to see if the obj is coming from top or bottom
            if (other.relativeVelocity.y <= 0f)
        {
            GetComponent<AudioSource>().Play();

            //boost ledge force

            if (gameObject.tag=="BoostLedge")
            {
                rb.AddForce(Vector2.up * 1900);
            }

            //Starting the animation on collision and destroying the object after
            if (gameObject.tag == "DestructiveLedge" && GetComponent<Animator>() != null)
            {
                GetComponent<Animator>().enabled = true;
                Destroy(gameObject, 1);

            }

            //BOUNCE effetct
            if (rb != null)
            {
                rb.AddForce(Vector3.up *1200);
            }
```

Fig 21. Bounce effect Code

Shooting Mechanics

The shooting mechanics were based on Instantiating a bullet prefab consisting of the pepperoni slice(Fig x) on which a script was attached to control the direction of the bullet, analogue as for modifying the direction of the player, by adjusting the velocity component of the rigid body existent on the bullet with the normalized difference between the mouse position and player position.

All of this was tested on the enemy concept made for the previous attempt on the project alongside a* pathfinding for creating a 2d grid graph and fetching the enemy the right path to follow. This was done by importing the free A* Pathfinding Project which holds functionality for creating a 2d/3d grid graph which is responsible for calculating a path which is being fetched inside my script for the enemy to follow in order to find the player. In this script the generating path of the enemy is done in the fixed update where a direction to the first waypoint is being fetched in a force vector in which the direction is multiplied by a speed variable to apply the steering behaviour to the NPC and the waypoint list is updated by a invoke repeating function which is a function premade by unity which runs a specific function in a fixed time in seconds, then repeatedly every rate in seconds(Unity Manual, 2019).

```
//check to see if a path is available
if (path == null)
{
    return;
}

//check to see if there are more waypoints or not to stop the player when neccessary
if (currentWaypoint >= path.vectorPath.Count)
{
    reachEndPath = true;
    return;
}
else
{
    reachEndPath = false;
}

//move the enemy
//get the direction to the next waypoint
Vector2 direction = ((Vector2)path.vectorPath[currentWaypoint] - enemyRB.position).normalized;

//get a direction force to be applied on the enemy
Vector2 force = direction * speed * Time.deltaTime;

//add force
enemyRB.AddForce(force);

//distance to next waypoint
float distance = (enemyRB.position - new Vector2(path.vectorPath[currentWaypoint].x, path.vectorPath[currentWaypoint].y)).sqrMagnitude;

if (distance < nextWaypointDistance*nextWaypointDistance)
    currentWaypoint++;

//snap the position of the sprite towards the moving direction
//if its going right
if (force.x > 0)
   monsterGFX.transform.localScale = new Vector3(-1f, 1f, 1f);

// going left
else if (force.x < 0)
   monsterGFX.transform.localScale = new Vector3(1f, 1f, 1f);
```

Fig 22. EnemyAI pathfinding code

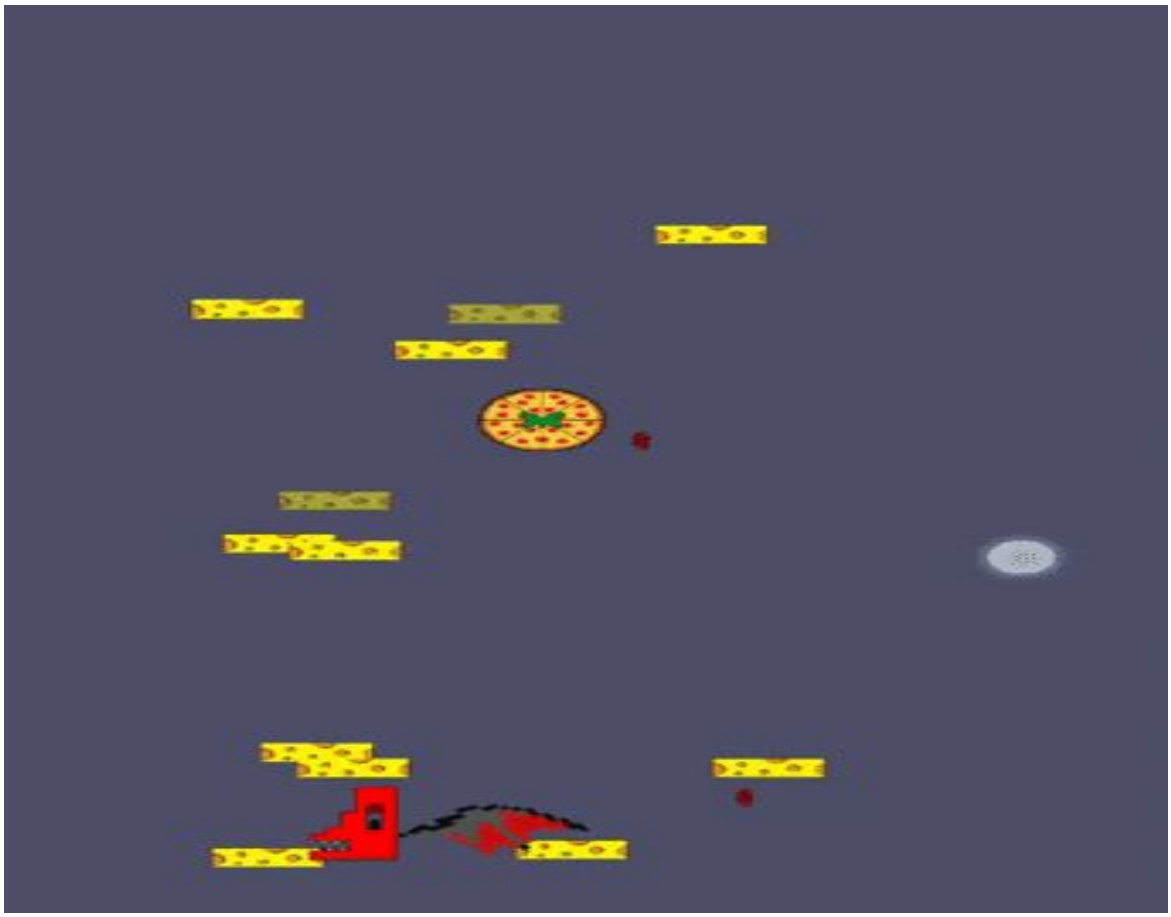


Fig.23 Shooting Mechanics + Enemy pathfinding

Generating Platforms

The next step was to find a way to generate the platforms, based on the tip suggested by Android Development on their blog, this problem was approached with the *Lazy first* concept, thus, at first the level generating was done in the Start method by having a vector position and a for loop at the start of the function in which the x and y values of the vector would be filled with a random value between the level width and height and then being instantiated at the position of the vector for an amount of times. This function is running first at the start of the function then it recalls itself when the player has reached the final ledge generated, when this happens, the level procedurally instantiates new ledges and deletes the old ones.

```
void Start()
{
    //Starting the profiling
        sampler = CustomSampler.Create("LEVEL GENERATOR");
        sampler.Begin();

    //position to instantiate
    Vector3 instantiatePos = new Vector3();

    //loop through the number of wanted platforms and instantiate it
    for (int i = 0; i < numberOfPlatforms; i++)
    {
        instantiatePos.x = Random.Range(-levelWidth, levelWidth);
        instantiatePos.y = Random.Range(minY, maxY);

        Instantiate(platformPrefab, instantiatePos, Quaternion.identity);

    }
    //end zone of profiling
    sampler.End();
```

Fig 24. Platform instantiating generation Function

Camera Follower

The next step was creating a camera follower. This was achieved by importing the Cinemachine package from unity which holds a camera component with functionality to follow a specific target.

Then the project development continued by animating the characters, thus the proper packages for rigging the 2D character were imported (2D animations, 2D IK) in the 2D IK package there is functionality to enable the rigging based on a sprite sheet which was exported from photoshop as a .PSB to enable unity the ability to sort the sprites based on the photoshop layers. The rigging was done by adding bones to the necessary parts of the character and updating the geometry of them to see the actual movement. All of this was then recorded inside the unity's animation system by adding keyframes in the timeline in which changes were made to the specific bones. The available animations of the enemy are :

1. Flying

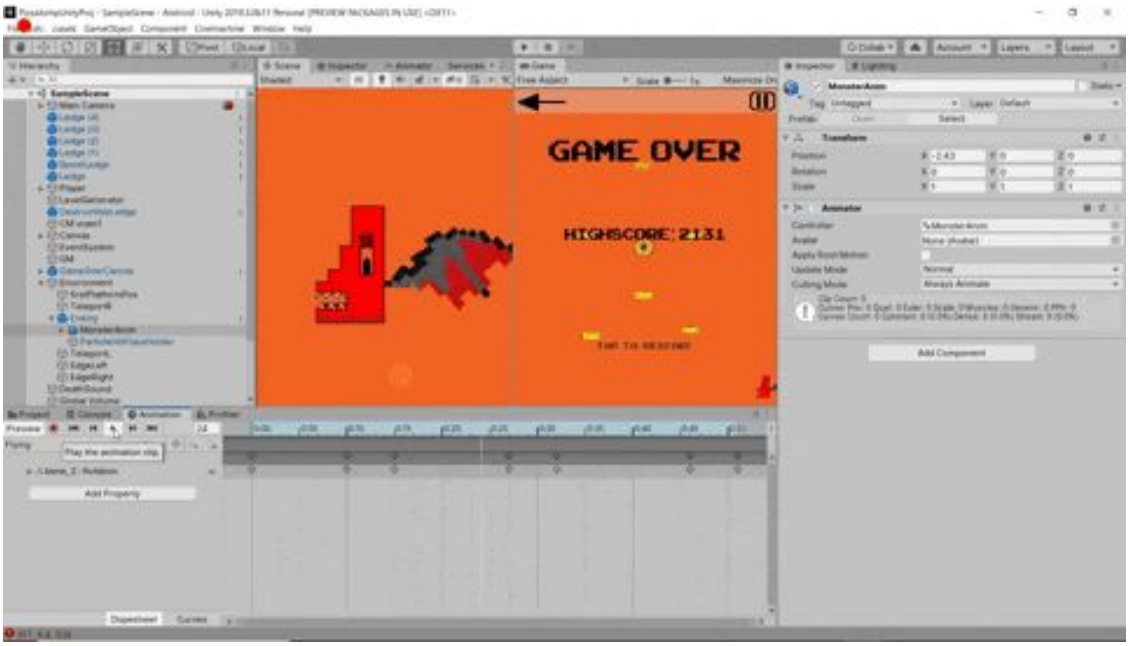
a. 

Fig.25 Flying Animation

2. Eating

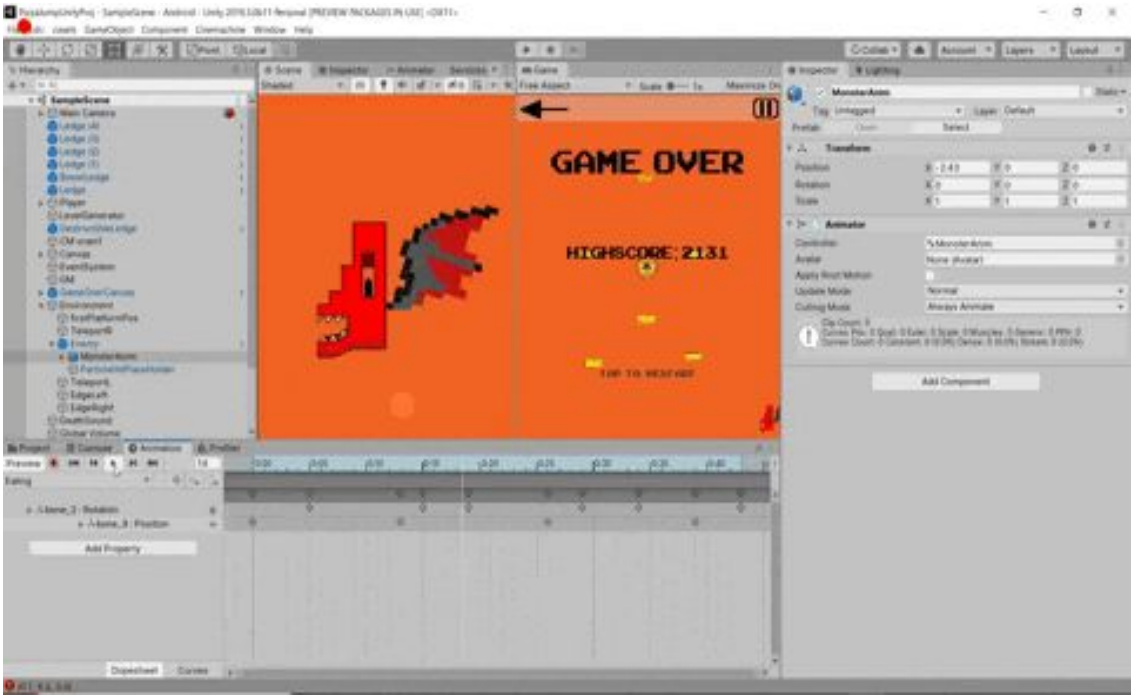
a. 

Fig 26. Eating Animation

Score

Furthermore, the score was made using the PlayerPrefs component which stores and access player preferences between game sessions.(Unity Manual, 2019). There are two values for the score, one is the current score and the high score. The score function is designed to set the

score as the high score after the first round and then the new score is being checked with the last set high score, choosing the max value from both.

```
0 references | 0 changes | 0 authors, 0 changes
void Update()
{
    //if the game is finished
    if (player.GetComponent<Player>().gameOver)
    {
        //disable the unnecessary things
        //enable the game over canvas
        DisableComponents();

        //store the score
        score = Mathf.RoundToInt(player.GetComponent<Player>().topScore);

        //compare the values and assign the highscore to the maximum one
        if (score > highScore)
        {
            highScore = score;
            PlayerPrefs.SetInt("HIGHSCORE", highScore);
        }

        //update the text
        scoreTxT.text = "SCORE: " + score.ToString();

        //reload the scene
        if (Input.GetMouseButtonDown(0) || Input.touchCount>0)
            SceneManager.LoadScene(1);
```

Fig. 27 Scoreboard Code

Audio

The Audio was done by using samples from the free pack called *The Essential Retro Video Game Sound Effects Collection* (SubspaceAudio, 2016). The game isn't going to be published using any of these soundtracks, it is purely for the sake of demonstrating this artefact and is not to be available to download/play anywhere.

Shader

Furthermore, a shader meant to replicate a dissolve effect was made using the ShaderGraph component, a volume framework which holds the property to make specific objects bloom, which was used to enhance the effect brightness all of this are rendered by the help of a custom render pipeline asset. The shader graph is making use mostly of noise effect and step nodes to create at a certain position which is then being adjusted every frame by using a fade float modifier. All of these are being rendered on a material in which the fade value can be easily adjusted using the *SetFloat* built in function(Unity Manual, 2019).

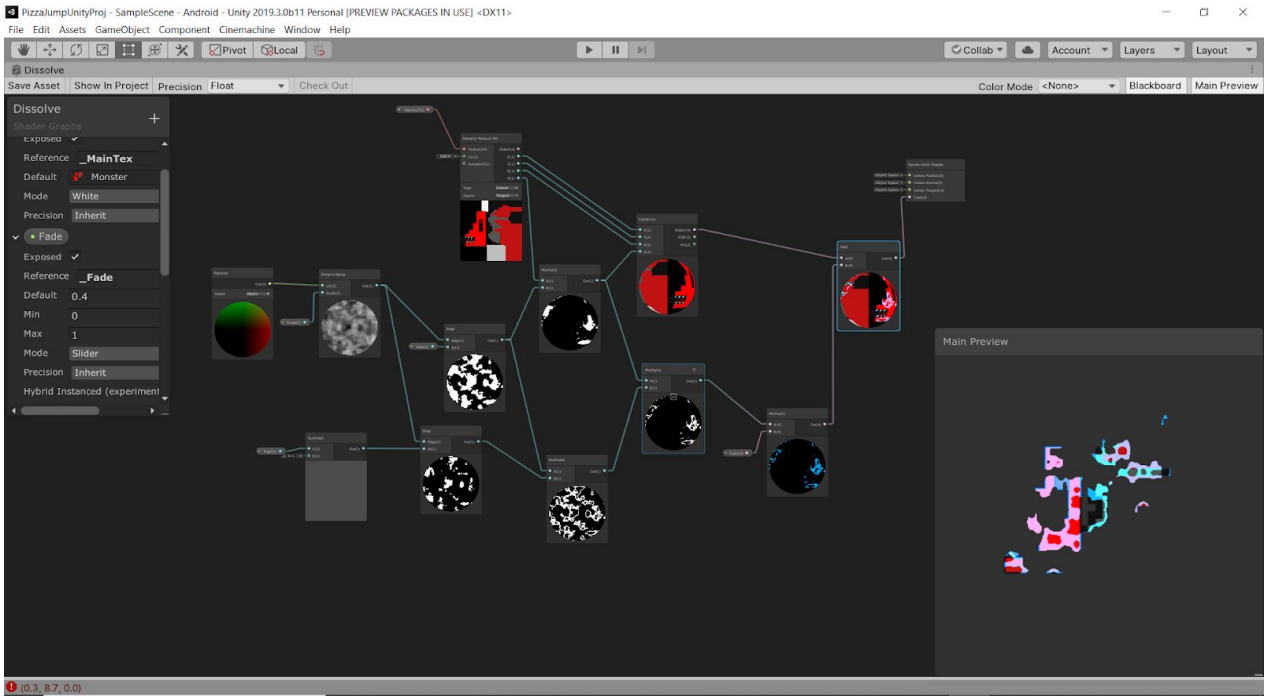


Fig 28. Dissolve Shader Graph

Canvas Scaler

The UI was adjusted accordingly to fit any device screen size thanks to unity's properties within the canvas which has a Canvas scaler component capable of scaling the canvas to the screen size which the running device is having.

Fig 29. Canvas Scaler Property

Finally, the color of the background was adjusted by editing the RGBA values based on the position of the player and multiplied by a multiplier value each frame.

GameFlow

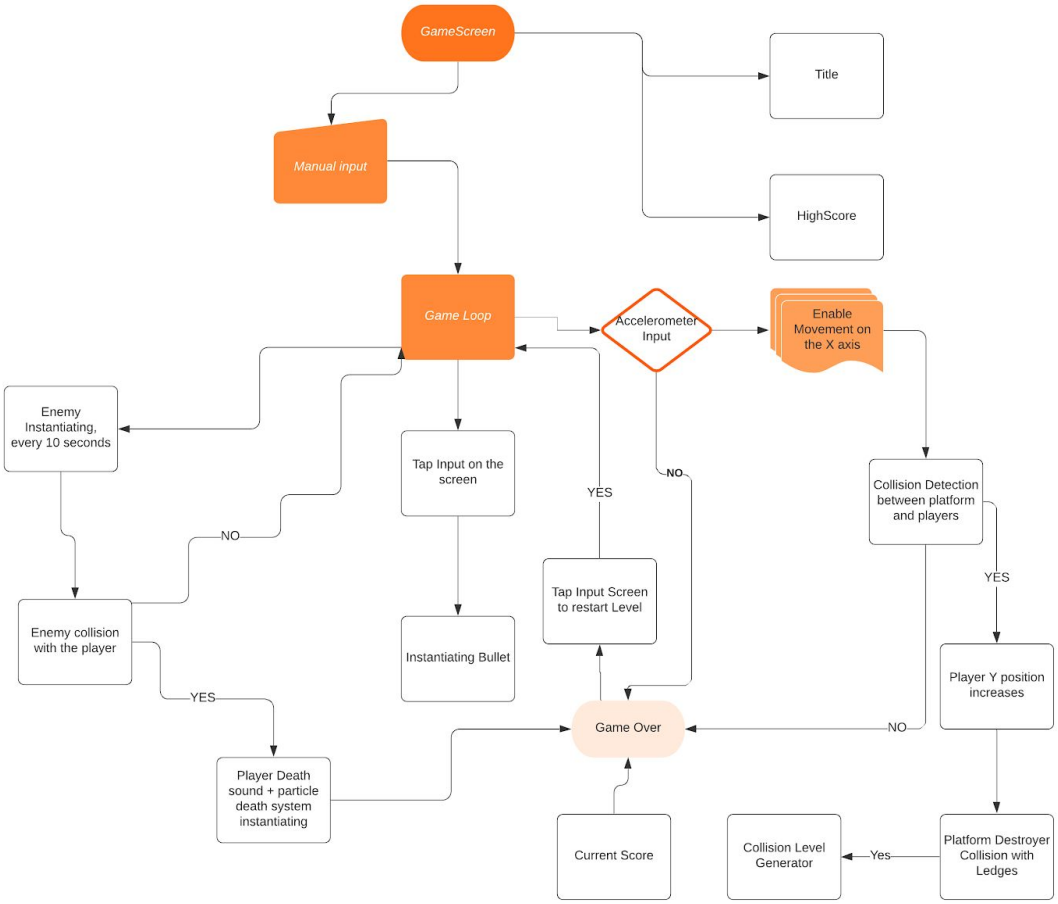


Fig 30. GameFlow

Profiling

A crucial part when developing games is profiling. Jan Bosch defines profiling in his book called "*Continuous Software Engineering"* as: "*Profiling prerelease software product performance is the measurement and representation of such properties of a product, which enables evaluation of that product's excellence before delivery"*. Us as humans tend to analyse ourselves on our schedules involving for instance the amount of time required to execute a task such house chores or even on the profesional side, when wondering how optimal is your work? The same mechanism applies on software products to ensure the optimal execution of the processes required to run the game, giving at the same time a good experience to the player.

The running time of the game was measured using Unity's profiler which has the ability to show the runtime of all aspects in the game in every frame.
By profiling, the aspect which needed the most power to run was spotted to be the shader which needed a custom render pipeline in order to run properly.
And the whole runtime of the game in one frame with the custom render pipeline was 290.56ms.
The game was tested running with the shader on a Huawei P10 Lite running Android 8.0, having a Octa-Core 4 x 2.1 GHz Cortex-A53 + 4 x 1.7GHz Cortex-A53 and 3GB RAM memory. The CPU runned all the processes from the game within a frame in 882.35 ms, having as the biggest one the Custom Render Pipeline with 177.09 ms and 69.1 KB memory used.

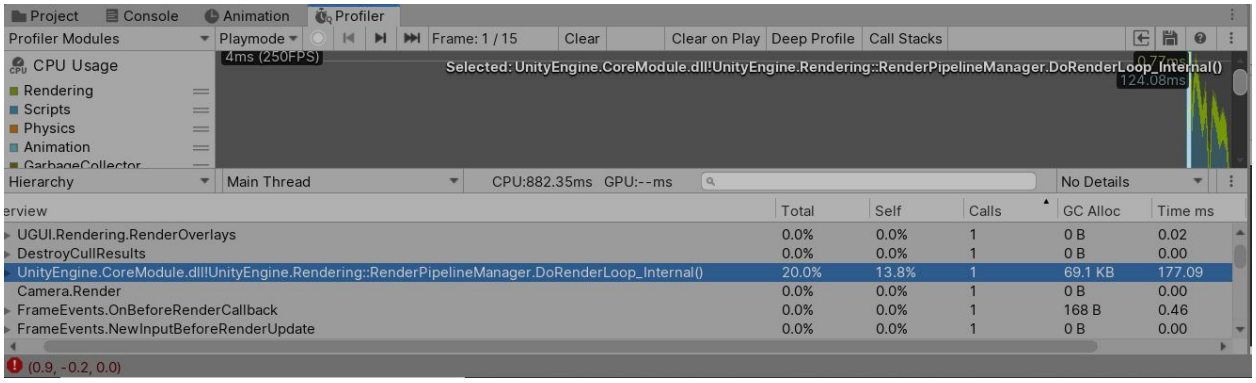


Fig 31. Custom Render Pipeline Asset Running Time

In this criteria, while testing on that phone, the game wasn't running as tested from the editor computer, it was visible differently, having varying local lag.

In the case of the shader, there was no way to dodge this, whether it is worth it to use it or not. So it was decided to exclude the shader and replace it with a prefab from a free package of particle systems which was instantiated by using a collision checking between the enemy and the bullet. retrieved from the unity store.

Another area which took quite a lot of time and space to compute was the level generator of the jumping platforms.The function was running at a time rate averaging 6.82ms out of 5 tests.The measurement of the function was possible by making use of the sampler class provided by Unity within the UnityEngine.Profiler header and is consisted of a custom sampler which has functionality to create custom marker (In->Out) inside the profiler to measure a specific function within the game. (Unity Europe, 2017).

```
2 references | 0 changes | 0 authors, 0 changes
public void InstantiatePlaftorms(int numberOfPlatforms, Vector2 lastPos)
{
    sampler = CustomSampler.Create("InstantiatingFunction");

    //begining of profiling
    sampler.Begin();

    //position to instantiate
    Vector3 instantiatePos = new Vector3();
    instantiatePos.y = lastPos.y;
    //loop through the number of wanted platforms and instantiate it
    for (int i = 0; i < numberOfPlatforms; i++)
    {
        instantiatePos.x = Random.Range(-levelWidth, levelWidth);
        instantiatePos.y += Random.Range(minY, maxY);

        instantiatedPlatform.Add ( Instantiate(platformPrefab, instantiatePos, Quaternion.identity));

        if (i == numberOfPlatforms - 1)
            instantiatedPlatform[i].name = "lastPlatform";
    }
    //end area of the profiler
    sampler.End();
}
```

Fig 32. Instantiating Level Generator

Level Generator Instantiate		
Test	Time(ms)	Average
1	6.53	6.82
2	6.6	
3	6.63	
4	6.2	
5	7.25	

Fig 33. Instantiating Level Generator Average Time

Having the game running at this timerate was not acceptable thus the level generator technique was adjusted using a *platform destroyer* empty game object, parented to the player, on which was added a box collider to keep track of the ledges that were passed and changing their position and based on their tag, instantiate special platforms based on a random value.between the width of the level and the distance between the platform destroyer collider and the object platform that collided with, added to the y position of the player. The runtime for this method was averaging 1.374 ms which compared to the previous way of generating the level which measured on average 6.82 ms, 5.446 ms margin between these two.The reason is based on the fact that the Lazy First method was generating 100 ledges at once then regenerating new ones at certain times and deleting the previous ones. On the other side, the time value is slow because it only happens the platform destroyer intersects a platform and in this time the function is not instantiating a new platform, it just changes its position thus having a significant amount of objects to deal with.

Level Generator Collision		
Test	Time(ms)	Average
1	0.7	1.374
2	1.41	
3	0.63	
4	3.72	
5	0.41	

Fig 34. Level Generator Collision Average Runtime

```
//margin between the distances
offset = other.transform.position.y - transform.position.y;

//the position to instantiate at
Vector2 posToInstantiate = new Vector2(Random.Range(-levelWidth, levelWidth),offset+ GetComponentInParent<Player>().transform.position.y);

//Starting the profiling
sampler = CustomSampler.Create("Collision Instantiating");
sampler.Begin();

if (other.tag == "Ledge")
        other.transform.position = posToInstantiate;

if (other.tag == "DestructiveLedge")
{
    Debug.Log("ELSE");
    Destroy(other.gameObject);
    Instantiate(destructivePlatformPrefab, posToInstantiate, Quaternion.identity);
}
if (Random.Range(1, 50) == 5)
{
    instantiatedObj = Instantiate(boostPlatformPrefab, posToInstantiate, Quaternion.identity);

    if (x == 0)
    {
        Instantiate(destructivePlatformPrefab, posToInstantiate, Quaternion.identity);
        x = 1;
    }
}
if (other.tag == "BoostLedge")
    Destroy(other.gameObject);

//end zone of profiling
sampler.End();
```

Fig 35. Collision Level Generator Runtime

Reflection

To summarize the analysis of the project, this entity is a clone of doodle jump all assets being personal, apart from the pathfinding and two particle system, respective the one for the player death as well as enemy death. I tried as possible to stick to my own assets because there will be no possible way to make a living out of making games by using some other people's assets and also there is no much learning and satisfaction in that. I don't necessarily believe that making clones of games is a bad thing, in order to improve, one must learn, and to learn he has to follow some guidelines and all the learning lies within the process of creating that particular piece of software, not in creating the idea of the next big original thing. For example, think about game jams, by definition a game jam is defined by Global Game Jam, which is one of the biggest contests in the world as "[...] *The goal is to come together and make a video game, or a non-digital game like a board game or card game*".(Global Game Jam, 2019). Thus in its essence, a game jam is seen as a contest in which the participants has to design an idea based on a given theme and to administer that idea into something playable, of course marks are given for originality as well but if your goal is first to get experience, to be able to make greater things in the future, learning some fundamentals first by making several clones, might be a good route for success.

A thing which I was extremely surprised that I managed to make it work was the whole process of animating the enemy character. I was surprised because I never worked with this and also my fellow artist/modellers colleagues, didn't make it quite a good advertisement, also maybe because they had to do a lot more powerful animations than mine was.Another good thing that I can fairly say I learnt how to implement is creating shaders respective materials based on that shader using shader graphs. All of this thanks to the tutorials made by Brackeys on his youtube channel (Brackeys, 2018).

The major problems that I encountered along the way with the techniques implementation were inside the level generating function, which was the one consuming the most computational power. The struggle happened when I was trying to find an alternative way to generate the platforms and was able finally to find one which obviously came with a problem as well. The new level generator is making use of a platform destroyer collider attached to the player and the problem triggered when the player would move too fast for the collision to happen. This was fixed using the unity's manual which suggested changing the collision type of the rigid body to continuous and thus the problem was fixed.

Another big problem was the frame drop rate of the game when playing it from the Huawei P10 Lite, due to the use of a shader, respective custom render pipeline asset with a volume framework for creating the bloom effect. As there was no adjustment found to change this It was decided not to use it. And finally the last major problem was with the shooting mechanism which based on my personal mistake which involved not normalizing the velocity vector of the bullet was made the shooting mechanism acting as following:

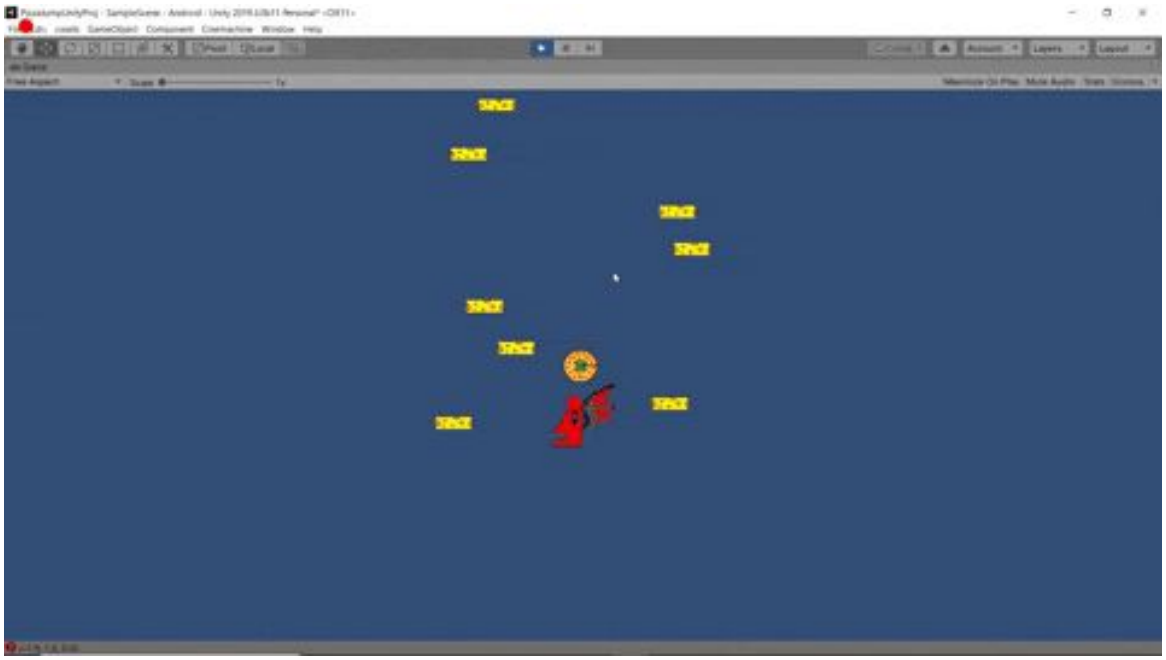


Fig 36. Shooting Problem

Making the bullet to go faster if the user would touch on a further position in the world (15, 0 ,0) rather than a closer one (3,0,0).

On reflection , I would have loved to make an artefact using the oculus rift s headset as the VR universe is one at the verge of being dived deep into it with games such as Half Life Alyx, the continuation of the good well known franchise called Half Life, made by Valve in 2020. But all in all, the process of making this game was quite enjoyable as I haven't worked on this platform before as well and got the chance to learn multiple new things about the aspects that can make a difference between a forgotten app sitting in the play store and one that sits in the top 50. Which in my opinion is all about game design and smooth mechanics and if I would have to do this again I would definitely plan the risks of my game more thoroughly as the first idea was dropped due to the lack of game design and this made me restructure the whole backbone of the game.

References

2017. History Of Cellphones And How Drastically They've Changed. [online] Available at: <https://www.youtube.com/watch?v=nrdNdprcYls> [Accessed 15 May 2020].

Dictionary.cambridge.org. 2020. MOBILE APPLICATION | Meaning In The Cambridge English Dictionary. [online] Available at: <https://dictionary.cambridge.org/dictionary/english/mobile-application> [Accessed 15 May 2020].

Rajput, M., 2015. Tracing The History And Evolution Of Mobile Apps. [online] Tech.co. Available at: <https://tech.co/news/mobile-app-history-evolution-2015-11> Seams, C., 2013. Car Phone Driven From Curiosity To Commodity To Collectible. [online] Petrolicious. Available at: <https://petrolicious.com/articles/car-phones>

En.wikipedia.org. 2011. IBM Simon. [online] Available at: <https://en.wikipedia.org/wiki/IBM_Simon>

En.wikipedia.org. 2012. Motorola Dynatac. [online] Available at: <https://en.wikipedia.org/wiki/Motorola_DynaTAC>

Giant Bomb. n.d. Dunia Engine (Concept) - Giant Bomb. [online] Available at: <https://www.giantbomb.com/dunia-engine/3015-3977/>

https://developer.android.com. 2020. Android 8.0 Behavior Changes. [online] Available at: <https://developer.android.com/about/versions/oreo/android-8.0-changes>

Shekar, S., 2017. Mastering Android Game Development With Unity. Birmingham: Packt Publishing Ltd.

Youtube.com. 2017. Unite Europe 2017 - Practical Guide To Profiling Tools In Unity. [online] Available at: <https://www.youtube.com/watch?v=OSlOwJP8Z14>

2018. Far Cry 5. Ubisoft.

Clement, J., 2020. Google Play Store: Number Of Apps 2019 | Statista. [online] Statista. Available at: <https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/> [Accessed 15 May 2020].

Finnegan, T., 2015. Learning Unity Android Game Development. Birmingham: Packt Publ.

Global Game Jam. 2020. Global Game Jam. [online] Available at: <https://globalgamejam.org/> [Accessed 15 May 2020].

Granberg, A., 2019. A* Pathfinding.

Sky, L., 2009. Doodle Jump. Lima Sky, GameHouse, Innovative Concepts in Entertainment, Real Arcade.

Technologies, U., 2020. Unity - Manual: Unity User Manual (2019.3). [online] Docs.unity3d.com. Available at: <https://docs.unity3d.com/Manual/index.html> [Accessed 15 May 2020].

Thorson, M., 2020. Celeste. Matt Makes Games.

Youtube. 2018. *Basics Of Shader Graph - Unity Tutorial*. [online] Available at: <https://www.youtube.com/watch?v=Ar9eIn4z6XE> [Accessed 15 May 2020].

SubSpaceAudio. 2020. *512 Sound Effects (8-Bit Style)*. [online] Available at: <https://opengameart.org/content/512-sound-effects-8-bit-style> [Accessed 15 May 2020].