Polytechnic Institute of Cávado and Ave

Advanced 3D Programming

2D Game

Course leader:

Bruno Freitas Ferreira

Student:

Romano Keser

Academic year 2020/2021

June 2021

CONTENT

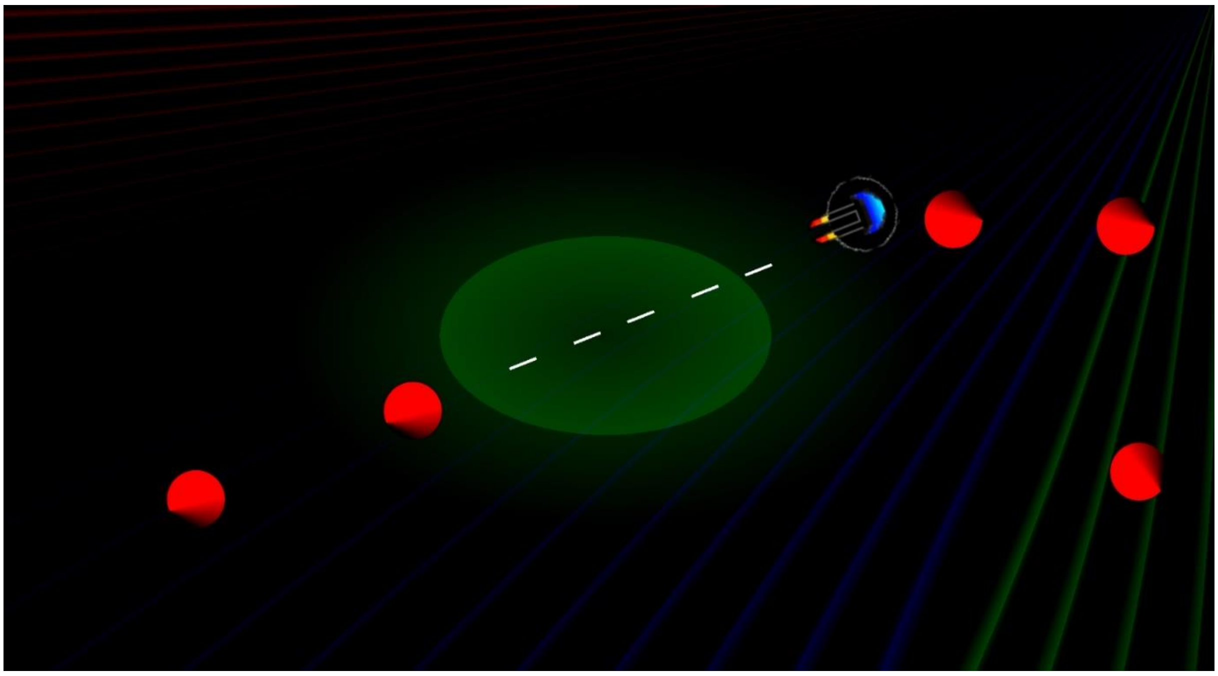
Table of Contents

1. Brief introduction 3
2. How it all works? 4
2.1 Player model 5
2.2 Enemy model 6
2.3 Background 7
2.4 Enemy spawning positions 8
3. Scripting 9
3.1 Shader 10
3.2 Flash interval 12
4. Conclusion 14

1. Brief introduction

The 2D Game made in Unity is about avoiding other enemies (obstacles). The player can shoot at the enemies and avoid their attack by moving around. The whole playground is followed by different visual effects made in HLSL Shaders.

Gameplay is very easy to learn as there are only a few things to learn. The player can also die if it gets "eaten" by enemies more than five times. The player can move around with WASD keys and in order to shoot, the player has to click the left mouse click.



Picture 1) Gameplay

2. How it all works?

There are four main objects: background, player, spawn positions and enemy. The background is just a plane with defined box colliders around it and applied shader to it. Player model is a sprite made in Photoshop with "PlayerTip" attached later on. Spawn positions are defined around the background plane where enemies are randomly spawning throughout the game.

2.1 Player model

Player model is a sprite made in Photoshop and it looks quite simple.

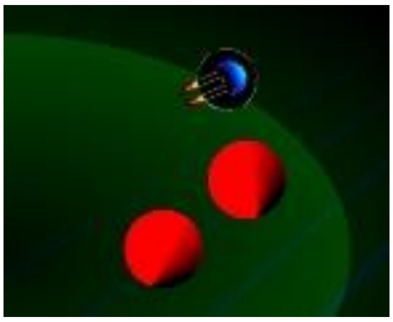
Picture 2) Player model

The top "candles" are where the bullets are coming from when shooting. They are also small sprites and only the script is attached to them, no visual effects.

The script "playerMovement" is attached to the player model where simple functions are made for the movement, but more on that later.

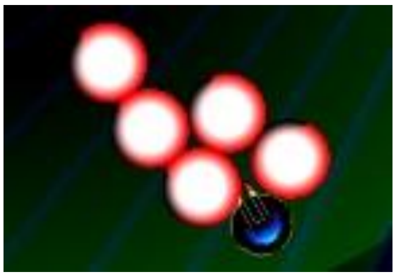
2.2 Enemy model

Enemy model is following the same logic except the Shader is applied to them. They are spawned outside of the background sprite and they're spawning every two seconds.



Picture 3) Enemy model

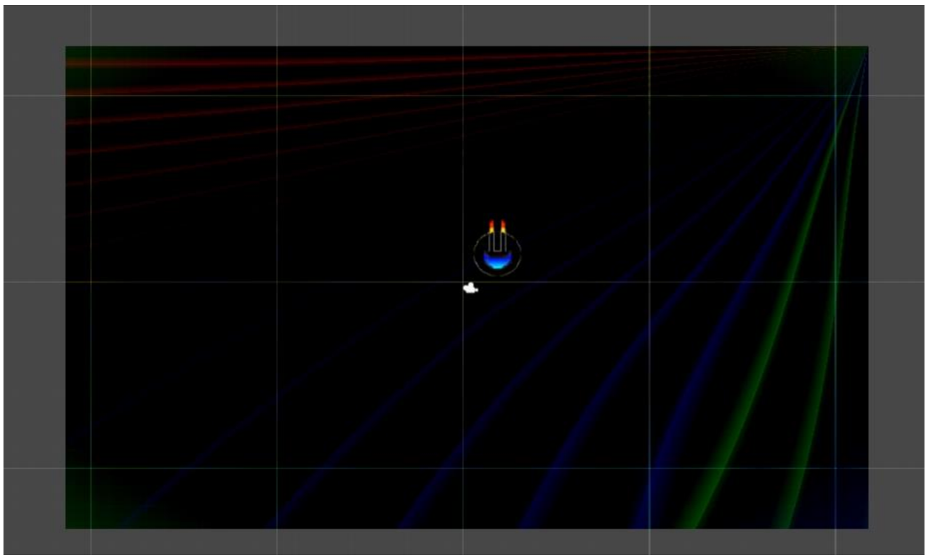
On this picture (Picture 2 – enemy model) you can see they are painted with red colour with white ring around them. The white ring around them is more frequent as the enemies are getting closer to the player (Picture 3).



Picture 4) White ring impulses are more frequent

2.3 Background

The background sprite is a simple sprite to which is added a Shader and Material.

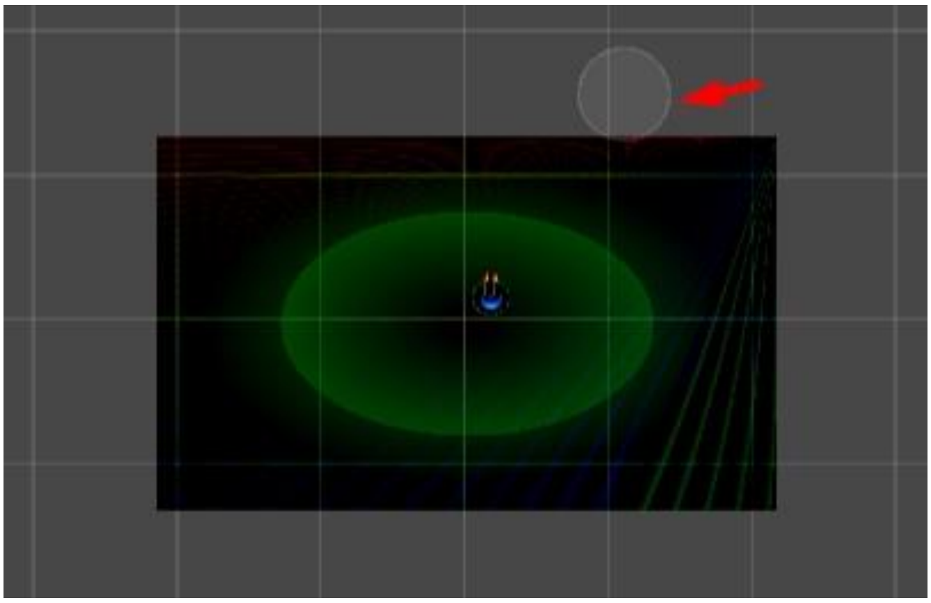


Picture 5) Background

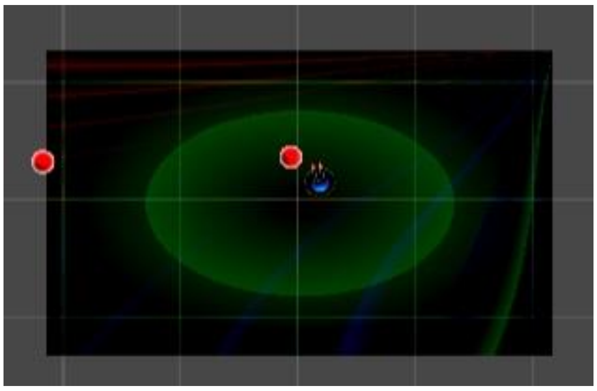
The colours of the background are constantly changing and the values can be changed in the Inspector, but more on that later.

2.4 Enemy spawning positions

There are 8 spawning positions around the playground. Enemies are spawning in random order on these positions. That way the enemies are attacking the player from multiple directions.



Picture 6) On of the spawn positions



Picture 7) Enemy spawning

3. Scripting

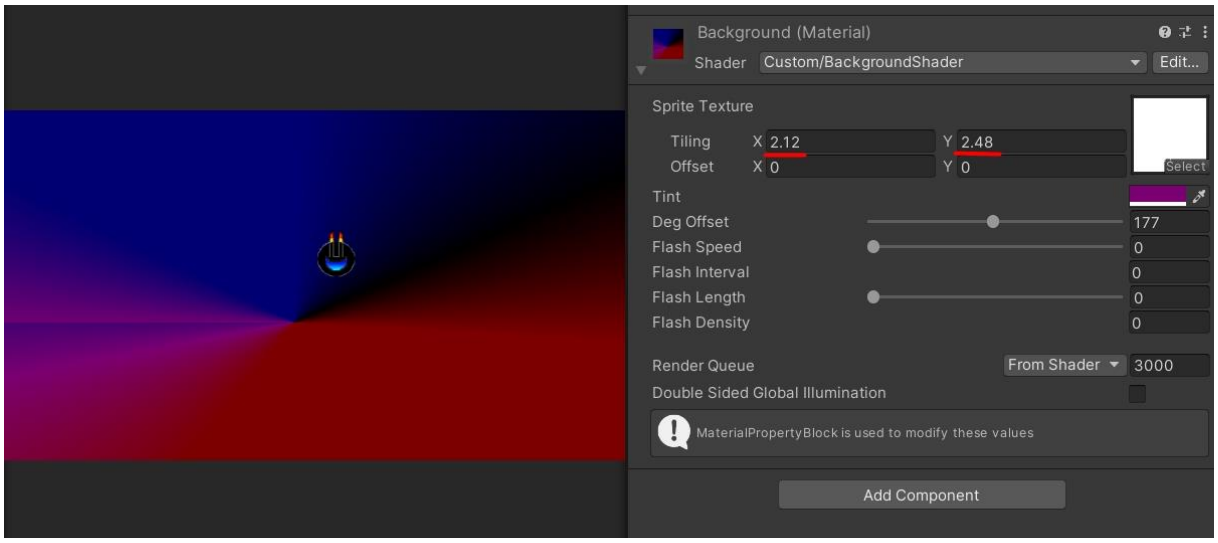
There are in total six scripts that are controlling the gameplay. To the player model there is two scripts attached, *characterMovement* and *playerAttack.* To the enemy model, there are *enemyAttack* and *enemy* that are controlling the enemy health status and *enemy* script that is calling the shader to change the colors and circle vibration effect. Shader material is attached to the background plane that is giving the glowing colour effect.

3.1 Shader

There are nine properties in the shader for colour rendering and five properties for the flash effect.

```
        [SerializeField] _MainTex("Sprite Texture", 2D) = "white" {}
        [SerializeField] _Color("Tint", Color) = (1,1,1,1)
        [HideInInspector] _StencilComp("Stencil Comparison", Float) = 8
        [HideInInspector] _Stencil("Stencil ID", Float) = 0
        [HideInInspector] _StencilOp("Stencil Operation", Float) = 0
        [HideInInspector] _StencilWriteMask("Stencil Write Mask", Float) = 255
        [HideInInspector] _StencilReadMask("Stencil Read Mask", Float) = 255
        [HideInInspector] _ColorMask("Color Mask", Float) = 15
        [HideInInspector][Toggle(UNITY_UI_ALPHACLIP)] _UseUIAlphaClip("Use Alpha Clip", Float) = 0
```

For the easier colour and 3D effect on the background handling, I made Sprite Texture and *Tint* properties (SerializedField) so they could be changed in the inspector.



Background (Material)
Shader Custom/BackgroundShader Edit...

Sprite Texture
Tiling X 2.12 Y 2.48
Offset X 0 Y 0
Tint
Deg Offset 177
Flash Speed 0
Flash Interval 0
Flash Length 0
Flash Density 0

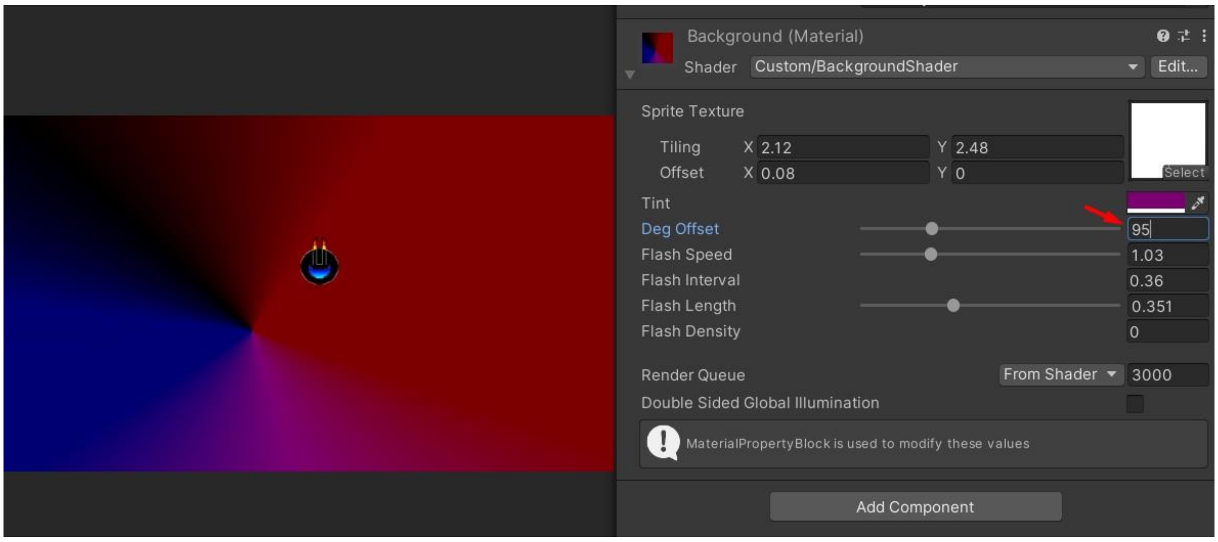
Render Queue From Shader 3000
Double Sided Global Illumination

MaterialPropertyBlock is used to modify these values

Picture 8) Changing the x and y values in the inspector

```
_DegOFfset("Deg Offset", Range(0, 360)) = 0
```

DegOffset is defined as a float and its range is from 0 to 360, as its values are changing the degrees of the colours and the light in the background.


Background (Material)
Shader Custom/BackgroundShader Edit...
Sprite Texture
Tiling X 2.12 Y 2.48
Offset X 0.08 Y 0
Tint
Deg Offset 95
Flash Speed 1.03
Flash Interval 0.36
Flash Length 0.351
Flash Density 0
Render Queue From Shader 3000
Double Sided Global Illumination
MaterialPropertyBlock is used to modify these values

Picture 9) DegOffset set to 90 degrees

```
            _DegOFfset("Deg Offset", Range(0, 360)) = 0
           _FlashSpeed("Flash Speed", Range(0, 4)) = 1
           _FlashInterval("Flash Interval", Float) = 4
           _FlashLength("Flash Length", Range(0, 1)) = 0.1
           _FlashDensity("Flash Density", Float) = 4
```

Flash Speed, Flash Interval, Flash Length and *Flash Density* are all available properties in the Inspector.

3.2 Flash interval

Flash interval is disabled in the background, rather it's created for enemies. As the enemies are getting closer to the player, the the flash effect gets more intense.

```
color.xyz += float3(1, 1, 1) * saturate(abs(abs(distance(float2(0.5, 0.5),
IN.texcoord) - _Time.y * _FlashSpeed * color ) % _FlashInterval - _FlashLength
) + _FlashLength) * _FlashDensity;
```

Where *abs* is HUE and it is converted to RGB with this function:

```
float3 HUEtoRGB(in float H) {
      float R = abs(H * 6 - 3) - 1;
      float G = 2 - abs(H * 6 - 2);
      float B = 2 - abs(H * 6 - 4);
      return saturate(float3(R, G, B));
}
```

This properties and functions are then called to *Enemy.cs* script and it is transfered to the enemy behaviour.

```
float distance = Vector2.Distance(player.transform.position,
this.gameObject.transform.position);
        float b = Mathf.InverseLerp(distance, 0.1f, 4.0f );
        material.SetFloat("_FlashSpeed", b * 1.0f);
    }
```

float direction calculates the distance between the enemies and the player, and if it gets lower than 4.0f, the flash speed gets higher. This is also adjustable in the inspector.

Similiar thing is made for enemies health. Here it is calculated if the enemy's health goes lower than 30, their colour is changed to red.

```
void Update()
    {
        ShaderTest();

        Vector3 direction = player.position - transform.position;
        float angle = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg;
        rb.rotation = angle;
        direction.Normalize();
        movement = direction;
    }
private void ShaderTest()
    {
        if (enemyhealth <= 30f)
        {
            material.color = Color.red;
        }

        float b = Mathf.InverseLerp(distance, 0.1f, 4.0f );
        material.SetFloat("_FlashSpeed", b * 1.0f);
    }
```

In the *Void() function we're calling the public* function *ShaderTest()* and it is referenced to the class *ShaderTest()* where the enemies health is calculated. If enemies health goes bellow 30f, enemies' colour is set to red.

4. Conclusion

Altough this is a simple game with the simple shader and visual effects, it has potential to later be modified. Basic UI has to be added. Game score and health bar has to be added too.

The classes are not very well organised and the whole project has to be more optimised.

The shader's elements can be included in the game settings so the user can play around with this properties. Interesting effects in the background can be done. From basic colour changing to the glowing effects and lights.

Currently the enemy's ability to switch the colour to red when they are low on health is working, but it gets affected to every enemy on the playground.

There are more things to be done in the future.