ECS7003P – Multi-Platform Game Development 2018/19

Planetary Defender: Game Description Document

John Allen - m.j.d.allen@se18.qmul.ac.uk

Gameplay screen capture – Planetary Defender

Gameplay video - https://youtu.be/afhx5Txeb1g

Preface

The following document outlines and describes the list of technical implementations made in Planetary Defender. The function, of each implementation is explained in full, as well as the motivation for including it. Each aspect of the game is also critiqued according to how impactful it is on the player's experience and whether this impact resembles the intended impact of the aspect upon design. The remainder of this document is split into five sections; environments, gameplay features, non-playable features, audio-visual and overall game quality. Planetary Defender was developed in Unity 2018.3.2f1 and scripts were managed using Visual Studio for Mac version 7.8.3.

1. Environments

Planetary Defender contains only one environment, this is mainly due to the fact that the player cannot move within its environment. This means there is no need for any terrain or navigation rules. The single environment in Planetary Defender is rudimentary and sees the player in space adjacent to an Earth like planet and its moon. This environment was simply created using a custom skybox component [1] attached to the Player Camera game object, a child of the Player object in the Game scene. Figure 1 shows the custom skybox features and how the skybox is rendered to the camera. The game environment is best viewed by watching the gameplay video listed on page 1.

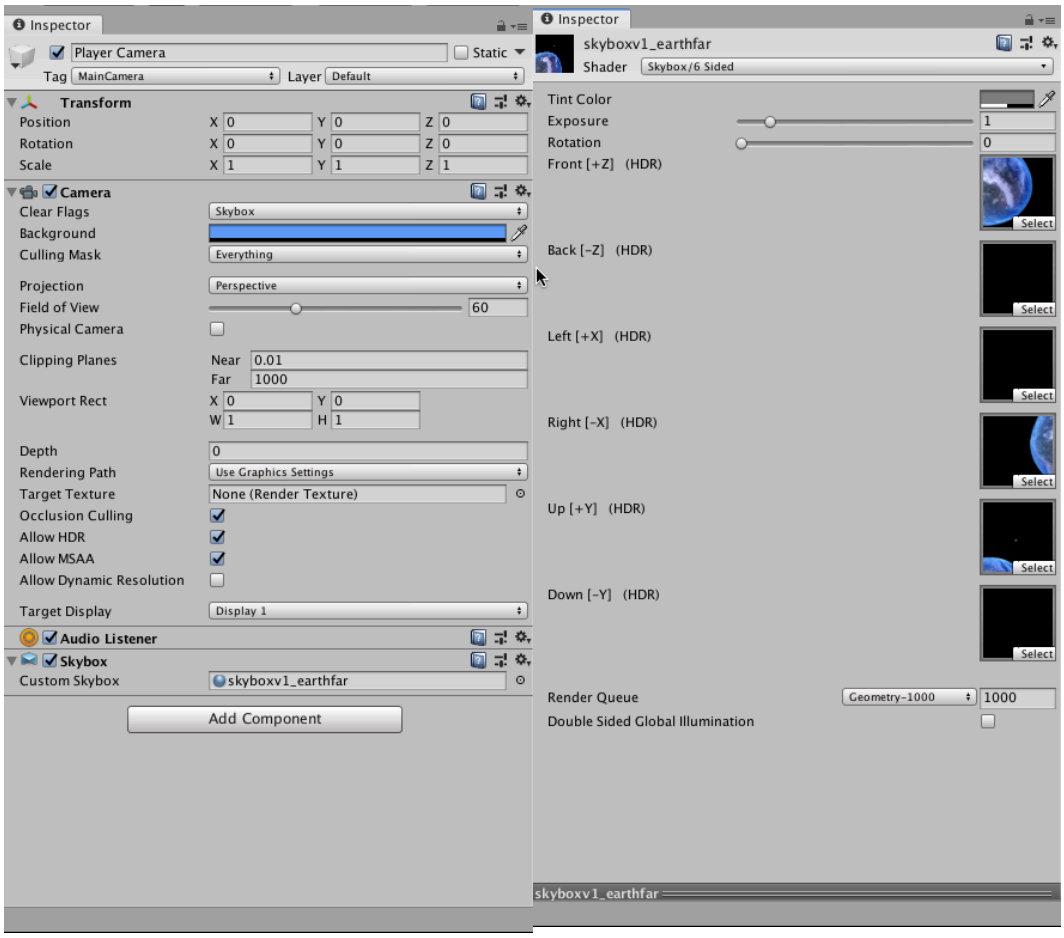

Figure 1 *– Screenshots of the skybox features implemented in the Unity editor.*

2. Gameplay Features

Planetary Defender is a first person shooter crossed with a tower defence game. The narrative of the game follows a wave like structure, where after a short acclimatisation time the waves begin. Upon each wave, enemies will begin to spawn around the player and it is the objective of the player to destroy the enemies before they destroy the players' ship.

2.1. Player Controlled Features

2.1.1. Aim

The player is located in a fixed position in the world space and the only adaptations that can be made to the players' transform component during gameplay are that of rotations executed by changes in mouse cursor position. The player has full 360° freedom of rotation about it's own up axis, 130° freedom of rotation about the players' right axis between the angles of 85° aiming up, toward the up axis and 45° aiming down, toward the negative up axis. Rotation about the players' forward axis was locked. The player was given a full view of the surroundings from all directions about its up axis because the intent was that enemies would spawn and approach the player from all directions. This attack was designed to feel somewhat overwhelming and frantic as per the design document. The players' aim up and down was locked as previously mentioned to ensure that, upon aiming down, the player was not able to look at an angle to steep as to be looking inside the players ship's hull. Upon aiming up, the rotation was locked such that the player could not aim so far up as to be looking behind them in the same fashion as a 180° turn about the up axis and 180° turn in the forward axis. The rotation about this forward axis was locked for this same reason. The code controlling the aim movement is displayed below and can be found between lines 59 and 62 of the Player Controller script, where the variable 'sensitivity' is a public float and was optimised during testing for the easiest aim.

```
horAim += Input.GetAxis("Mouse X") * sensitivity;
verAim += Input.GetAxis("Mouse Y") * -sensitivity;

transform.eulerAngles = new Vector3(Mathf.Clamp(verAim,85,45),horAim,0);
```

2.1.2. Changing weapons

There are three weapons available to the player in Planetary Defender; laser, machine gun and rockets. At the beginning of the game, the player has the laser equipped. The weapon system game object contains all three weapons with spacing between them of one unit in the up direction. To change weapons, the player uses the space bar and upon a space bar press, the weapon system object, which is a child of the player camera and is in turn a child of the player. Upon pressing space, the weapon system is displaced either one unit down or two units up to align the selected weapon with the players view. There is a cool down time of 2 seconds between weapons changes to ensure the human player can acclimatize to weapon after each change. Code controlling the weapon changes is displayed overleaf and can be found between lines 93 and 121 in the Player Controller script attached to the Player game object. The first if statement displayed below controls the weapon switch from laser to machine gun, the second controls the switch from machine gun to missiles and the last control the switch from missiles back to laser.

```
if (Input.GetKeyDown("space") && laser && nextMove <= Time.time)
{

 nextMove = Time.time + coolDown;
 laser = false;
 machineGun = true;
 weaponSystem.transform.position += weaponSystem.transform.up;

}

if (Input.GetKeyDown("space") && machineGun &&nextMove<=Time.time)
{

 nextMove = Time.time + coolDown;
 missiles = true;
 machineGun = false;
 weaponSystem.transform.position += weaponSystem.transform.up;

}

if (Input.GetKeyDown("space") && missiles && nextMove <=Time.time)
{

 nextMove = Time.time + coolDown;
 laser = true;
 missiles = false;
 weaponSystem.transform.position -= weaponSystem.transform.up * 2;

}
```

2.1.3. Quitting to main menu and restarting game

At any time during the game the player can choose to quit to the menu or restart the game from the beginning. This can be executed by pressing the ‘r’ key to restart the game or the ‘q’ key to quit to the menu. Code controlling this is displayed below and can be found between lines 123 and 129 of the Player Controller script. In the below code; scene 1 represents the game scene and scene 0 represents the menu scene.

```
if (Input.GetKeyDown("r"))
        {
            SceneManager.LoadScene(1);

        }

if (Input.GetKeyDown("q")) { SceneManager.LoadScene(0); }
```

2.1.4. Firing

In order to destroy enemies in Planetary Defender the weapons must be fired at the enemies. For all three weapons the left mouse button controls when each weapon is fired. When fired, the laser casts a ray from each of the barrels to a maximum distance of 5000. The code controlling the rays casted using the laser can be found between lines 29 and 54 of the Laser Behaviour script, a component of the two Laser Beam line renderers, children of the left and right laser objects. This code is also displayed overleaf, where the ‘laserFire’ variable is a Boolean used to determine if the mouse is currently being pressed with the laser equipped.

```
RaycastHit target;

if (laserFire)
        {
if (Physics.Raycast(transform.position, transform.forward, out target))
        {
if (target.collider)
        {
    laserRenderer.SetPosition(1, new Vector3(0, 0, target.distance * 5));

if (Time.time > nextDamage && target.collider.gameObject.tag == "Enemy")
        {

            nextDamage = Time.time + 0.5f;
            GameObject Enemy = target.collider.gameObject;
            EnemyHealth enemyHealth = Enemy.GetComponent<EnemyHealth>();
            enemyHealth.EnemyDamage(gameObject);

         }
         }
         }
else laserRenderer.SetPosition(1, new Vector3(0, 0, 5000));
         }
else laserRenderer.SetPosition(1, new Vector3(0, 0, 1));
```

The laser fires while the left mouse button is being pressed down and ceases to fire when the left mouse button is released or the weapon is changed. The Laser Behaviour script calls the 'laserFire' variable set in the Player Controller script. The variable value is changed according to whether or not the left mouse button is being pressed down or not and if the laser is the current equipped weapon. The machine gun fires a burst of rounds each time the left mouse button is clicked, however there is a cool down time of 2 seconds between machine gun bursts. The ammunition fired from the machine gun is instantiated via co-routine as well as the velocities of the expelled projectiles. This co-routine contains an iterator which incrementally creates and fires the machine gun round according to the fire rate of the machine gun set using the public float 'nextRound'. This co-routine is displayed below and can be found between lines 133 and 157 in the Player Controller script. The missiles are fired in the same way as the other two weapons, using the left mouse click, a 'barrage' object is instantiated to give the effect of each of the four silos firing a missile. Missile firing also has a cool down time of 2 seconds and one barrage is sent each time the weapon is fired.

```
IEnumerator BurstFire() {

        nextMove = Time.time + 1.8f;

        gunSound.Play();

        for (int i = 0; i <= shots; i++)
        {

            GameObject roundLeft = Instantiate(round, muzzlePositionLeft + gunLeft
.transform.forward * 1.0f, gunLeft.transform.rotation);
            GameObject roundRight = Instantiate(round, muzzlePositionRight + gunRi
ght.transform.forward * 1.0f, gunRight.transform.rotation);

            Rigidbody roundLeftRB = roundLeft.GetComponent<Rigidbody>();
            Rigidbody roundRightRB = roundRight.GetComponent<Rigidbody>();

            roundLeftRB.velocity = aim.forward * muzzleVelocity;
            roundRightRB.velocity = aim.forward * muzzleVelocity;

            yield return new WaitForSeconds(nextRound);

        }
        gunSound.Stop();
    }
```

Below is the code controlling the mouse inputs for the firing, which call on the code snippets displayed on the previous page. This code can also be found in the Player Controller script between lines 64 and 91.

```
if (Input.GetMouseButtonDown(0) && laser)
{ laserFire = true; ; laserSound.Play(); }
if ((Input.GetMouseButtonUp(0) && laser) || (!laser && laserFire))
{ laserFire = false; laserSound.Stop(); }


        if (Input.GetMouseButtonDown(0) && nextShot < Time.time && machineGun)
        {
            nextShot = Time.time + coolDown;

            flashLeft = Instantiate(muzzleFlash, muzzlePositionLeft,
gunLeft.transform.rotation);
            flashRight = Instantiate(muzzleFlash, muzzlePositionRight,
gunRight.transform.rotation);

            flashLeft.transform.SetParent(weaponSystem.transform);
            flashRight.transform.SetParent(weaponSystem.transform);

            StartCoroutine(BurstFire());

        }

        if (Input.GetMouseButton(0) && missiles && nextMissile < Time.time)
        {

            nextMissile = Time.time + coolDownMissiles;

            GameObject Barage = Instantiate(barrage, player.position +
player.transform.forward * 0.8f, player.rotation);
            Rigidbody BarageRB = Barage.GetComponent<Rigidbody>();
            BarageRB.AddForce(transform.forward * thrust * Time.deltaTime);

        }
```

2.2. Non-Player Controlled Features

2.2.1. Enemy Behaviour

There are three kinds of enemies in Planetary Defender, each type of enemy is a different size, ship design and has a different amount of health upon spawn. The small enemies have 50 health, medium enemies have 150 health and large enemies have 250 health. When an enemy spawns, regardless of its class, it will emerge from a portal which is destroyed after 10 seconds. The portal spawn locations are determined by randomly generating a point in cylindrical coordinates within the range that can be seen by the players' camera. A portal then spawns with an enemy behind, which then emerges from the portal. The class of enemy spawned is decided upon by generating 3 random values assigned to each enemy class and the highest being selected as the enemy to be spawned. Each wave spawns more enemies than the last and depends on which wave the player is currently on, there is also a 5 second wait time between enemies spawning and a 20 second wait between waves. The code controlling this can be found overleaf and between lines 45 and 92 of the Game Controller script. After the enemies have spawned, two scripts control the behaviour of the enemies. One script named 'Enemy Trajectory' controls the enemy navigation and another named 'Enemy Combat' controls how attacks the player. Enemy Trajectory moves the enemy toward the players' ship location until it reaches a distance of 20 units from the players' ship at a speed of 12 units per second. Enemy Combat instantiates projectiles from the enemy, directed toward the players' ship every second with projectiles taking 2 seconds to reach the target once they are fired. All enemies have the same settings for the trajectory and combat scripts regardless of class.

```
IEnumerator SpawnWave()
    {
        yield return new WaitForSeconds(WarmUp);

        for (int n = 0; n < waveCount; n++)
        {

            enemyCount = (waveCount + 1) * 2;

            for (int i = 0; i <= enemyCount; i++)
            {

                enemySmallVote = Random.value;
                enemyMediumVote = Random.value;
                enemyLargeVote = Random.value;

  if (enemySmallVote >= enemyMediumVote && enemySmallVote >= enemyLargeVote)
 { enemy = EnemySmall; }
  else if (enemyMediumVote >= enemySmallVote && enemyMediumVote >= enemyLargeVote)
 { enemy = EnemyMedium; }
                else { enemy = EnemyLarge; }

                d = Random.Range(90, 110);
                h = Random.Range(-30, 80);
                phi = Random.Range(0, 360);

                xPos = d * Mathf.Sin(phi);
                zPos = d * Mathf.Cos(phi);

                portalPos = new Vector3(xPos, h, zPos);
                direction = Player.transform.position - portalPos;
        GameObject Portal = Instantiate(portal, portalPos, Quaternion.LookRotation
(direction));

                shipXpos = 1.1f * d * Mathf.Sin(phi);
                shipZpos = 1.1f * d * Mathf.Cos(phi);

                Vector3 spawnPos = new Vector3(shipXpos, h, shipZpos);
                Vector3 shipDirection = Player.transform.position - spawnPos;

           GameObject Enemy = Instantiate(enemy, spawnPos, Quaternion.LookRotation
(shipDirection));

                yield return new WaitForSeconds(5);
            }
            yield return new WaitForSeconds(WaveWait);
        }
    }
```

2.2.2. Dealing damage

When the player fires their weapons at the enemy ships, if the attack reaches the enemy, the attack will deal damage to the enemy. Each of the three weapons deals damage differently and at differing rates. The laser deals damage every half a second it spends hitting an enemy such that 30 damage is dealt to enemies per second while the laser is hitting them. The code controlling this damage is displayed below and can also be found between lines 39 and 47 of the Laser Behaviour script. The Enemy Health script controls the amount of damage done by each weapon a small section of which is displayed overleaf.

```
if (Time.time > nextDamage && target.collider.gameObject.tag == "Enemy")
     {
        nextDamage = Time.time + 0.5f;
        GameObject Enemy = target.collider.gameObject;
        EnemyHealth enemyHealth = Enemy.GetComponent<EnemyHealth>();
        enemyHealth.EnemyDamage(gameObject);
     }
```

```
if (Other.tag == "Barrage") { health -= barrageDamage; }
if (Other.tag == "Bullet") { health -= gunDamage / 40; }
if (Other.tag == "Laser") { health -= laserDamage / 4; }
```

The machine gun deals damage for each shot that collides with the enemy. Each burst contains 20 shots for each gun, with a total damage available to deal on each burst of 60. The bursts fire with a velocity of 50 units per second and contain a collider component acting as a trigger. The code controlling this behaviour can be found at the bottom of page 5. The missiles deal 100 damage for each barrage which collides with an enemy, the barrage also contains a collider which acts as a trigger. In order to move the barrage of missiles away from the player, a force is added to the barrage over time to give a more realistic rocket propelled effect. This code is also displayed at the bottom of page 5. As well as the player dealing damage to enemies, enemies can also deal damage to the players' ship. The ship initially has 10,000 health and takes 20 damage each time an enemy projectile collides with the ship. When either an enemy or the ship's health drops to 0 or below they are destroyed and are no longer active in the game hierarchy. Rounds fired from the machine gun have a lifetime of 2 seconds and barrages have a lifetime of 5 seconds if they do not collide with an enemy.

3. Non-Playable Features

3.1. Main Menu

The main menu is the first thing a player encounters when playing Planetary Defender, it gives the player the option to being the game or to quit the application and exit. On the top left of the screen the controls are outlined and the title of the game is shown in the centre. The background of the main menu shows the players' ship and the planet the ship is defending. The menu screen is displayed below in figure 2 (screenshot taken from Unity editor).

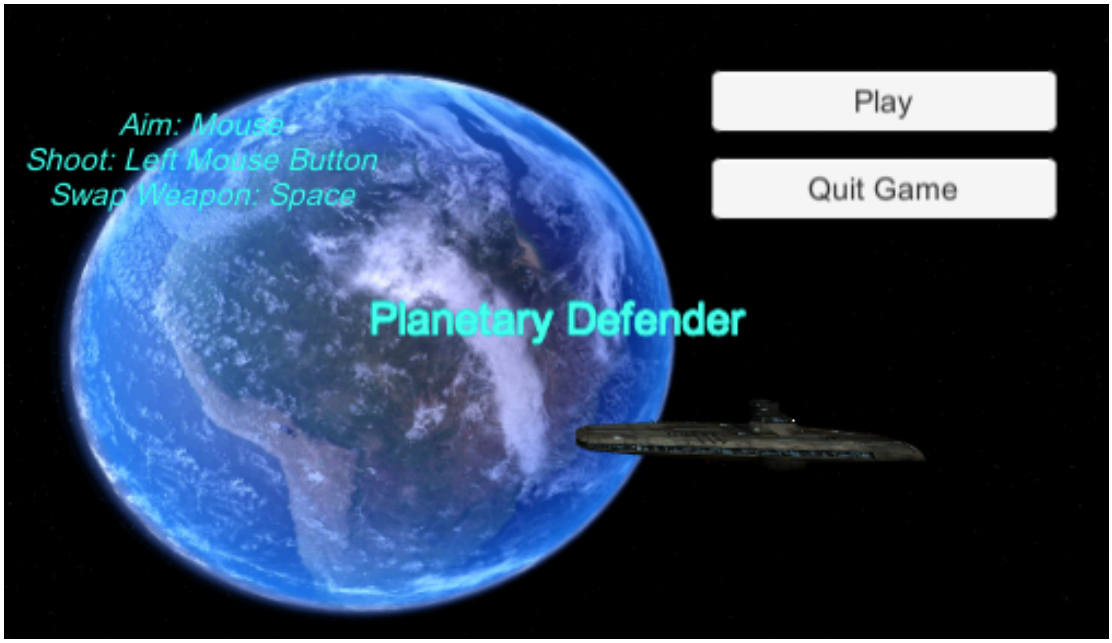

Figure 2: *Screenshot of menu screen taken from Unity editor*

3.2. Game canvas

In game a canvas overlays information about the environment to the player. This includes the ships' health, the score and the crosshair for aiming. The crosshair is a small red cross located at the centre of the players field of view. The station health is displayed at the bottom centre of the screen and the game score (number of projectiles landed on enemy ships) is displayed at the bottom left of the screen after the first enemy is destroyed. Figure 3 shows a screenshot of the in game canvas.

Figure 3: *Screenshot of in game screen showing crosshair, score and ship's health.*

4. Audio-Visual

The only continuously playing audio is the sci-fi style background music [2] played on loop during gameplay and while the player is at the menu. When each weapon fires, its own unique sound plays according to the duration of the weapon firing. While the laser is firing, a space like laser sound [3] plays on repeat and a blue laser line shown in figure 4 is displayed on screen, when the laser is turned off, the sound stops playing and the laser line is no longer rendered. While bullets are being instantiated during a machine gun burst, a machine gun sound [4] plays until the end of the burst as well as a muzzle flash particle effect [10] (shown in screen capture on page 1) which runs while rounds are spawning for that burst. When firing a barrage of missiles, a sound attached to the barrage game object plays a bazooka sound [5] and a flame particle effect is run from each missile in the barrage [11] and shown in figure 5. Upon the spawn of an enemy portal, a banging sound [6] is played once to signal the opening of the portal. When en enemy projectile collides with the players ship, a clinking sound [7] is played to confirm the ships taking of damage. This collision between the ship and the enemy projectile also runs a particle effect mimicking the impact of the projectile into the ship [12]. Upon destruction of an enemy, an explosion sound [8] is played to notify the player of the enemy destruction. An explosion particle effect is also run upon enemy destruction [13] this explosion is shown in figure 6. Finally upon destruction of the players ship, an echoing explosion sound [9] is played.

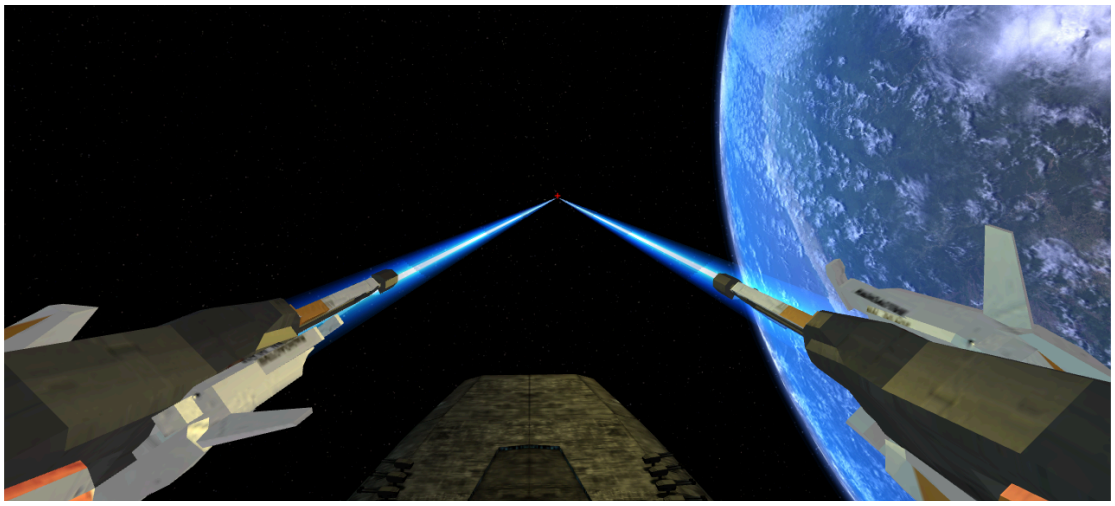

Figure 4: *Screenshot of laser line emanating from laser barrels.*

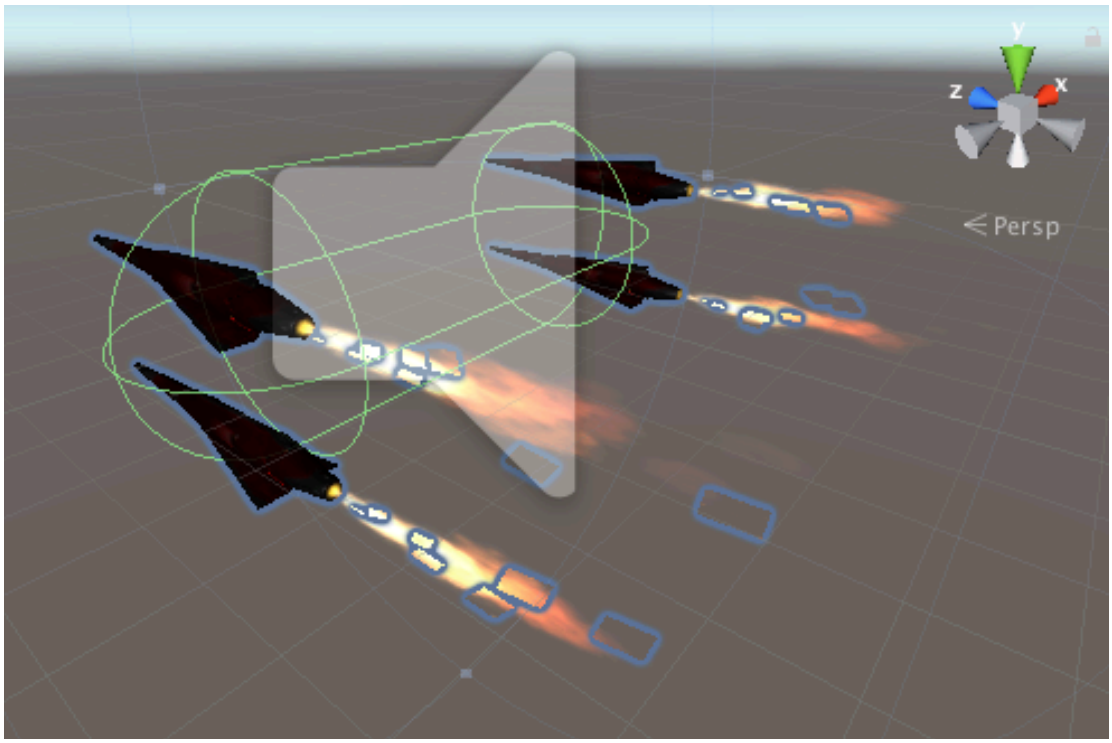

Figure 5: *Screenshot of flame particle effects from missile barrage.*

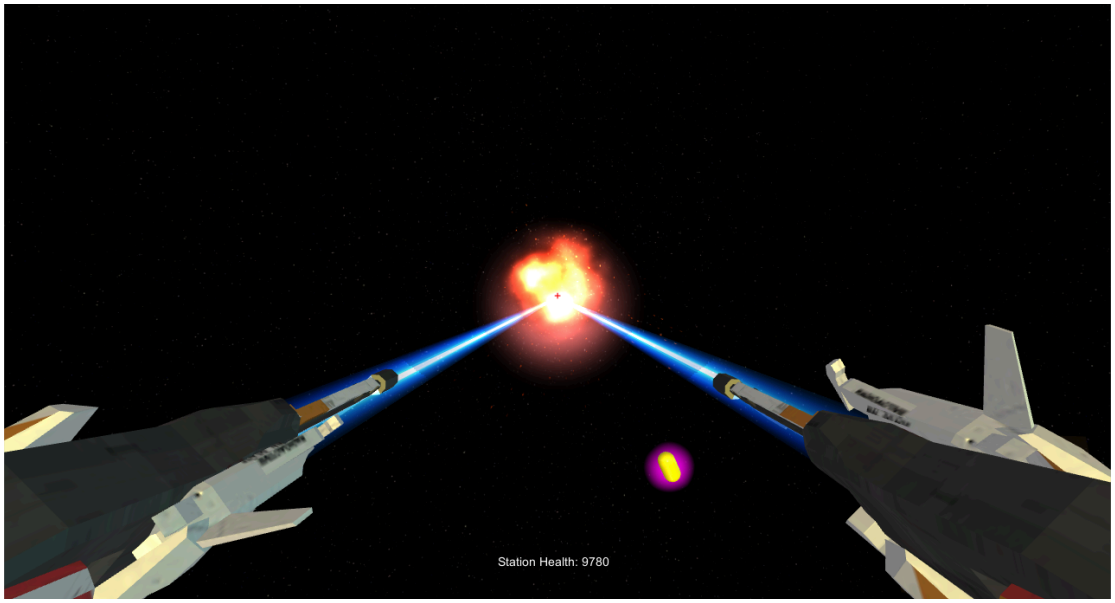

Figure 6: *Screenshot of enemy destruction explosion*

5. Game Quality

Overall, the final game did not reach the high standard outlined in the design document. The range of weapons and enemies had to be reduced into the final game, no special abilities were added to help during combat, the enemy behaviour and attack style was implemented in a fairly rudimentary way. Most of these setbacks were due to the time constraints of the project and the design being fairly optimistic. However, the game still achieved a realistic space scenario with all sounds and effects added with the intention of the game being as realistic as possible. Although outlined in the design document that Planetary Defender would aim to resemble the detailed sci-fi worlds or Mass Effect and Halo the game has predictably not reached this level but still has a wholesome, realistic feel. Little improvements were made to the players' HUD during development, this was also mainly due to the more important aspects like enemy behaviours being prioritised above aesthetic aspects less important to gameplay. The visual style of the enemies, the players' ship and the weapons all follow this quasi-realistic visual style set out in the design document to some degree. The game itself would likely benefit from an extensive testing phase where balancing weapon damage for both the player and enemies as well as enemy and player health could prove to make the game much more interesting to play. In conclusion, the game has fallen short of the expectations set in the design document, however the final game is still a fun, glitch free, semi-realistic sci-fi shooter, meaning most of the standards for the game were met but refinement is needed to improve the game to a publishable standard.

References

[1] – Unity Asset Store, STAGIT EAST, Earth & Planets skyboxes, skyboxv1_earthfar, https://assetstore.unity.com/packages/2d/textures-materials/sky/earth-planets-skyboxes-53752, 03/2019.

[2] – Unity Asset Store, CHIBOLA PRODUCTIONS, Stars, https://assetstore.unity.com/packages/audio/music/stars-17497, 03/2019.

[3] – Unity Asset Store, ZERO RARE, Sound FX – Retro Pack, laser_28, https://assetstore.unity.com/packages/audio/sound-fx/sound-fx-retro-pack-121743, 03/2019.

[4] – Unity Asset Store, RAFFAELE, Futuristic Weapons Set, shot_machinegun 1, https://assetstore.unity.com/packages/audio/sound-fx/weapons/futuristic-weapons-set-15644 - 03, 2019.

[5] - Unity Asset Store, RAFFAELE, Futuristic Weapons Set, shot_bazooka, https://assetstore.unity.com/packages/audio/sound-fx/weapons/futuristic-weapons-set-15644 - 03, 2019.

[6] – Unity Tutorials, Space Shooter Tutorial, Assets, Audio, misc bangs, bn4, 02/2019.

[7] – Unity Asset Store, NOVA SOUND, Free Fireworks – Fire FX – Nova Sound, Subfire Kick, https://assetstore.unity.com/packages/audio/free-fireworks-fire-fx-nova-sound-39475, 03/2019.

[8] – Unity Asset Store, NOVA SOUND, Free Fireworks – Fire FX – Nova Sound, Deep Fireworks, https://assetstore.unity.com/packages/audio/free-fireworks-fire-fx-nova-sound-39475, 03/2019.

[9] – Unity Tutorials, Space Shooter Tutorial, Assets, Audio, misc bangs, bn13, 02/2019.

[10] – Unity Asset Store, JMO, War FX, WFX_MF FPS RIFLE3, https://assetstore.unity.com/packages/vfx/particles/war-fx-5669, 03/2019.

[11] – Unity Asset Store, JMO, War FX, WFX_FLAMETHROWER LOOPED, https://assetstore.unity.com/packages/vfx/particles/war-fx-5669, 03/2019.

[12] – Unity Asset Store, JMO, War FX, WFX_BIMPACT METAL, https://assetstore.unity.com/packages/vfx/particles/war-fx-5669, 03/2019.

[13] - Unity Asset Store, JMO, War FX, WFX_explosiveSmoke Big, https://assetstore.unity.com/packages/vfx/particles/war-fx-5669, 03/2019.