

Unreal Game Development Documentation

SIT151 ASSESSMENT 4

Game Name: **Space Evader**

STUDENT NAME: Miles Ogilvie

STUDENT NUMBER: 220248802

WORKSHOP TIME (or Cloud): Tuesday 5:00pm

Contents

Game Component & Feature 1: Custom Player Movement (Rotation + Physics Based)	3
DESCRIPTION	3
REASONING	3
BLUEPRINTS/SCRIPTS AND IMPLEMENTATION	3
Game Component & Feature 2: Enemy Waves – Wave-based System.....	5
DESCRIPTION	5
REASONING	5
BLUEPRINTS/SCRIPTS AND IMPLEMENTATION	5
Game Component & Feature 3: Visual and Audio Feedback – Enhanced Dynamic Visual Effects.....	8
DESCRIPTION	8
REASONING	8
BLUEPRINTS/SCRIPTS AND IMPLEMENTATION	9
Game Component & Feature 4: Visual and Audio Feedback – Enhanced Dynamic Audio Features....	11
DESCRIPTION	11
REASONING	11
BLUEPRINTS/SCRIPTS AND IMPLEMENTATION	11
Extra Transformations or Features	14
Game Component & Feature 5: Screen Boundary Clamp	14
DESCRIPTION	14
REASON	14
BLUEPRINTS/SCRIPTS AND IMPLEMENTATION	14
References	15

Game Component & Feature 1: Custom Player Movement (Rotation + Physics Based)

DESCRIPTION

This feature replaces the default directional controls with a more advanced, immersive player movement system. The player ship now rotates to follow the mouse cursor and moves using a physics-based thrust system, simulating momentum and drift similar to a space shooter.

This feature supports precise aiming and skill-based navigation, enhancing player control and engagement.

REASONING

This movement system adds a layer of depth to the gameplay. The mouse-based aiming allows for quick and intuitive targeting, while the thrust-based physics creates more strategic navigation — requiring the player to think ahead and anticipate movement.

The design takes inspiration from classic space shooters and arcade-style movement systems, offering a more satisfying player experience compared to rigid grid-style movement.

BLUEPRINTS/SCRIPTS AND IMPLEMENTATION

360° Mouse-Based Rotation

The player ship now constantly rotates to face the mouse cursor. This was achieved using `Camera.main.ScreenToWorldPoint()` to convert the mouse's screen position into a world position, and `Mathf.Atan2` to compute the correct rotation angle. The result is applied using `Quaternion.Euler`.

`-90f` is used to adjust the angle because the sprite points upwards in Unity's 2D axis.

```
37 // Face mouse cursor
38 Vector3 mouseWorldPosition = Camera.main.ScreenToWorldPoint(Input.mousePosition);
39 Vector3 direction = mouseWorldPosition - transform.position;
40 direction.z = 0f;
41
42 float angle = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg;
43 transform.rotation = Quaternion.Euler(0f, 0f, angle - 90f); // Adjust if your sprite faces up/down by default
44
```

Physics-Based Thrust Movement

Instead of manually updating the position with `transform.position += ...`, movement is now handled using Unity's 2D physics engine. A `Rigidbody2D` was added to the `PlayerShip`, and movement is achieved by applying force in the direction the ship is facing (`transform.up`).

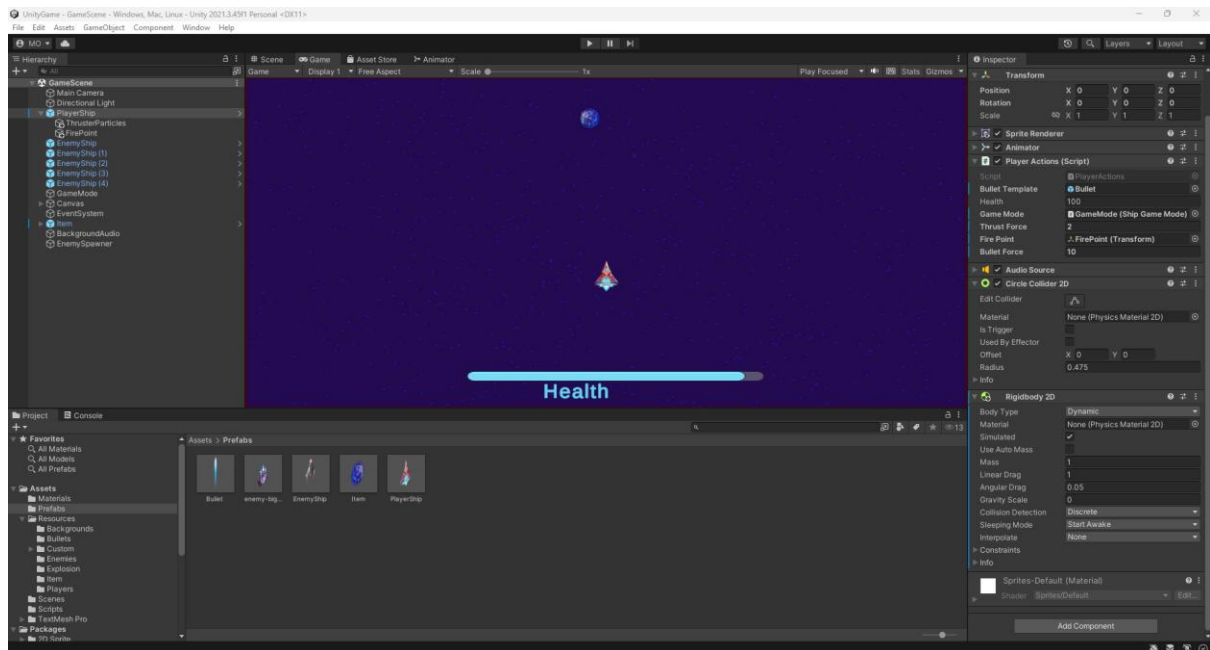
```
46 //Move (W)
47 if (Input.GetKey(KeyCode.W))
48 {
49     rb.AddForce(transform.up * thrustForce);
50 }
```

This gives the ship a more realistic sliding or drifting motion, where the ship continues to move even after the player stops pressing movement keys, unless drag is applied. This change matches the

ship's new rotation system and offers better control for navigating enemy waves coming from all directions.

To refine the feel of the physics-based movement system, the ship's RigidBody2D was configured with a Linear Drag value of 0.5–1.0. This creates a more controlled drift effect, allowing the ship to decelerate gradually when thrust is no longer applied. The result is a smoother, more satisfying feel that avoids overly slippery or hard-to-control motion, aligning well with the mouse-aimed shooting system.

Updated PlayerShip GameObject with Rigidbody2D, CircleCollider2D, and thrustForce value



Game Component & Feature 2: Enemy Waves – Wave-based System

DESCRIPTION

This feature introduces a structured, wave-based enemy spawning system. Enemies appear in waves and must be destroyed before the next wave begins. Each wave becomes progressively more challenging by increasing the number of enemies, encouraging player movement, and constant engagement.

Enemies spawn from all four screen edges — top, bottom, left, and right — and move to randomized positions within the screen bounds. This ensures the player must continually adapt their position and shooting direction.

REASONING

The wave-based system was designed to satisfy the task requirement of a structured spawning mechanic and to build tension during gameplay. Instead of spawning enemies endlessly or randomly, waves provide clear pacing, visual challenge escalation, and player feedback.

Additionally:

- Spawning from all directions prevents the player from staying in one safe spot.
- Requiring all enemies to be cleared before the next wave maintains focus and prevents system overload.
- It enables difficulty scaling and opens opportunities for scoring, combos, and boss waves later.

BLUEPRINTS/SCRIPTS AND IMPLEMENTATION

1. EnemySpawner.cs – Controls wave spawning

This script handles:

- Tracking active enemies
- Delaying between waves
- Spawning enemies at random screen edges

```

5  public class EnemySpawner : MonoBehaviour
6  {
7      public GameObject enemyPrefab;
8      public int baseEnemiesPerWave = 5;
9      public float spawnInterval = 0.3f;
10     public float waveDelay = 2f;
11
12     private int currentWave = 1;
13     private List<GameObject> activeEnemies = new List<GameObject>();
14     private bool spawningWave = false;
15
16     // Unity Message | 0 references
17     void Update()
18     {
19         // Remove any destroyed enemies from the list
20         activeEnemies.RemoveAll(enemy => enemy == null);
21
22         // If no enemies left and not already spawning a new wave, start next
23         if (activeEnemies.Count == 0 && !spawningWave)
24         {
25             StartCoroutine(SpawnWave(currentWave));
26         }
27     }
28
29     // 1 reference
30     IEnumerator SpawnWave(int wave)
31     {
32         spawningWave = true;
33         yield return new WaitForSeconds(waveDelay);
34
35         int enemiesToSpawn = baseEnemiesPerWave + (wave - 1); // Simple difficulty scaling
36
37         for (int i = 0; i < enemiesToSpawn; i++)
38         {
39             Vector3 spawnPos = GetSpawnPosition();
40             GameObject enemy = Instantiate(enemyPrefab, spawnPos, Quaternion.identity);
41             activeEnemies.Add(enemy);
42             yield return new WaitForSeconds(spawnInterval);
43         }
44
45         currentWave++;
46         spawningWave = false;
47     }
48
49     // 1 reference
50     Vector3 GetSpawnPosition()
51     {
52         Camera cam = Camera.main;
53         Vector2 screenMin = cam.ViewportToWorldPoint(Vector2.zero);
54         Vector2 screenMax = cam.ViewportToWorldPoint(Vector2.one);
55
56         int side = Random.Range(0, 4); // 0=Top, 1=Right, 2=Bottom, 3=Left
57         float x = 0, y = 0;
58
59         switch (side)
60         {
61             case 0: // Top
62                 x = Random.Range(screenMin.x, screenMax.x);
63                 y = screenMax.y + 1f;
64                 break;
65             case 1: // Right
66                 x = screenMax.x + 1f;
67                 y = Random.Range(screenMin.y, screenMax.y);
68                 break;
69             case 2: // Bottom
70                 x = Random.Range(screenMin.x, screenMax.x);
71                 y = screenMin.y - 1f;
72                 break;
73             case 3: // Left
74                 x = screenMin.x - 1f;
75                 y = Random.Range(screenMin.y, screenMax.y);
76                 break;
77         }
78
79         return new Vector3(x, y, 0f);
80     }
81 }

```

2. EnemyShip.cs – Movement and destruction logic

Each enemy:

- Picks a random target on screen
- Moves toward it using Update()
- Destroys itself on contact with bullet or player
- Triggers explosion animation if assigned

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5
6  public class EnemyShip : MonoBehaviour
7  {
8      public float moveSpeed = 2f;
9      private Vector3 targetPos;
10     public GameObject explosionPrefab;
11
12
13     @ Unity Message | 0 references
14     void Start()
15     {
16         PickNewTarget();
17     }
18
19     @ Unity Message | 0 references
20     void Update()
21     {
22         transform.position = Vector3.MoveTowards(transform.position, targetPos, moveSpeed * Time.deltaTime);
23
24         if (Vector3.Distance(transform.position, targetPos) < 0.1f)
25         {
26             PickNewTarget(); // Choose a new random point
27         }
28
29     2 references
30     void PickNewTarget()
31     {
32         Camera cam = Camera.main;
33         Vector2 screenMin = cam.ViewportToWorldPoint(Vector2.zero);
34         Vector2 screenMax = cam.ViewportToWorldPoint(Vector2.one);
35
36         float x = Random.Range(screenMin.x, screenMax.x);
37         float y = Random.Range(screenMin.y, screenMax.y);
38         targetPos = new Vector3(x, y, 0f);
39     }
40
41     @ Unity Message | 0 references
42     void OnTriggerEnter2D(Collider2D other)
43     {
44         if (other.CompareTag("Bullet") || other.CompareTag("Player"))
45         {
46             // Create explosion at enemy's position
47             if (explosionPrefab != null)
48             {
49                 Instantiate(explosionPrefab, transform.position, Quaternion.identity);
50             }
51
52             if (other.CompareTag("Bullet"))
53                 Destroy(other.gameObject); // remove bullet
54
55             Destroy(this.gameObject); // remove enemy
56         }
57     }
58 }
```

Game Component & Feature 3: Visual and Audio Feedback – Enhanced Dynamic Visual Effects

DESCRIPTION

Camera Shake

- This feature adds camera shake as an immediate visual feedback effect when the player takes damage. The shake conveys impact and urgency, enhancing the player's immersion and awareness during combat.

Player Thruster Particle Effect

- To further enhance the visual feedback and immersion, a thruster particle system was attached to the back of the player ship. The particle system emits a cone of glowing particles that simulate thrust, adding polish and visual motion cues.
- The system is configured to emit dynamically and optionally be toggled during movement only.

REASONING

Camera Shake

- This visual feedback adds a **tactile sense of being hit**, improving the game feel. It also helps players identify moments of damage through more than just UI text or numbers.
- By limiting the shake's duration and intensity, the effect is noticeable without being disorienting. This makes gameplay more engaging without compromising clarity.

Player Thruster Particle Effect

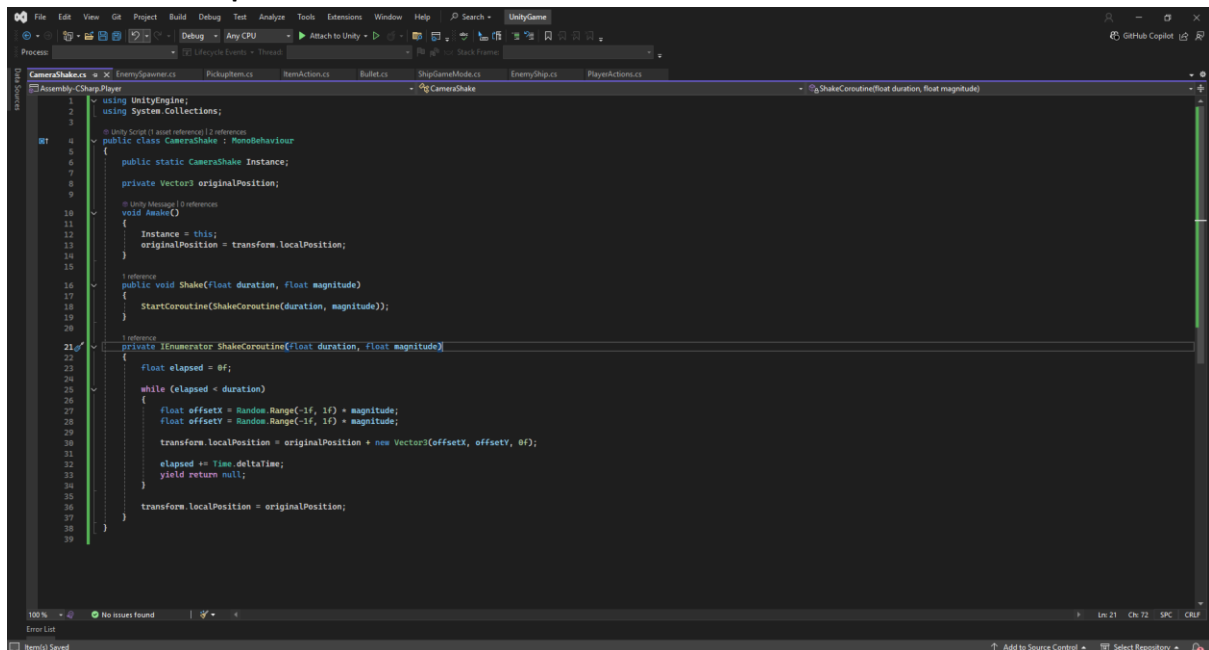
- This feature visually conveys movement and speed, which helps communicate gameplay information — especially when dodging enemies. It's a lightweight but impactful polish feature that contributes to the overall feel of the game.
- It reinforces the *“you are controlling a ship”* visual language and aligns with common game feedback techniques in arcade-style space shooters.

BLUEPRINTS/SCRIPTS AND IMPLEMENTATION

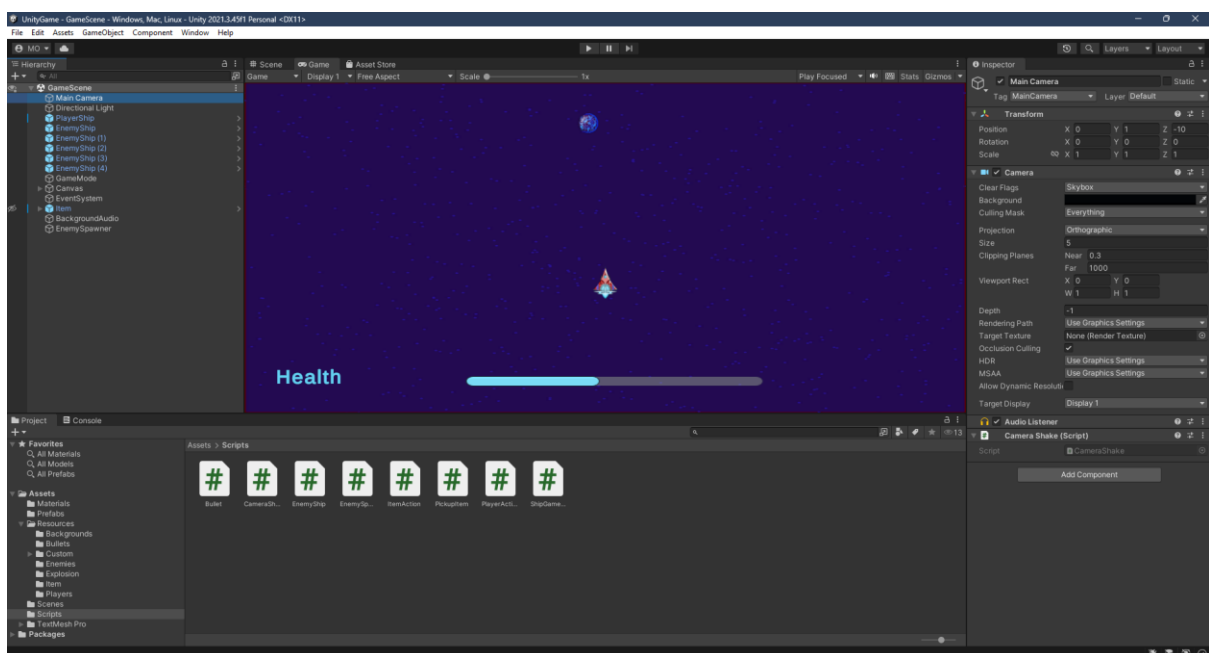
Camera Shake

- A CameraShake script was written and attached to the Main Camera. It provides a static reference so any object (like the player) can trigger a shake via `CameraShake.Instance.Shake(duration, magnitude)`.
- In the `PlayerActions.cs` script, the camera shake is triggered inside the `OnCollisionStay()` method when the player collides with an `EnemyShip`.
- This creates a short, randomized screen shake effect that helps the player feel the impact of taking damage, even without needing to look at the health bar.

Camera Shake Script:

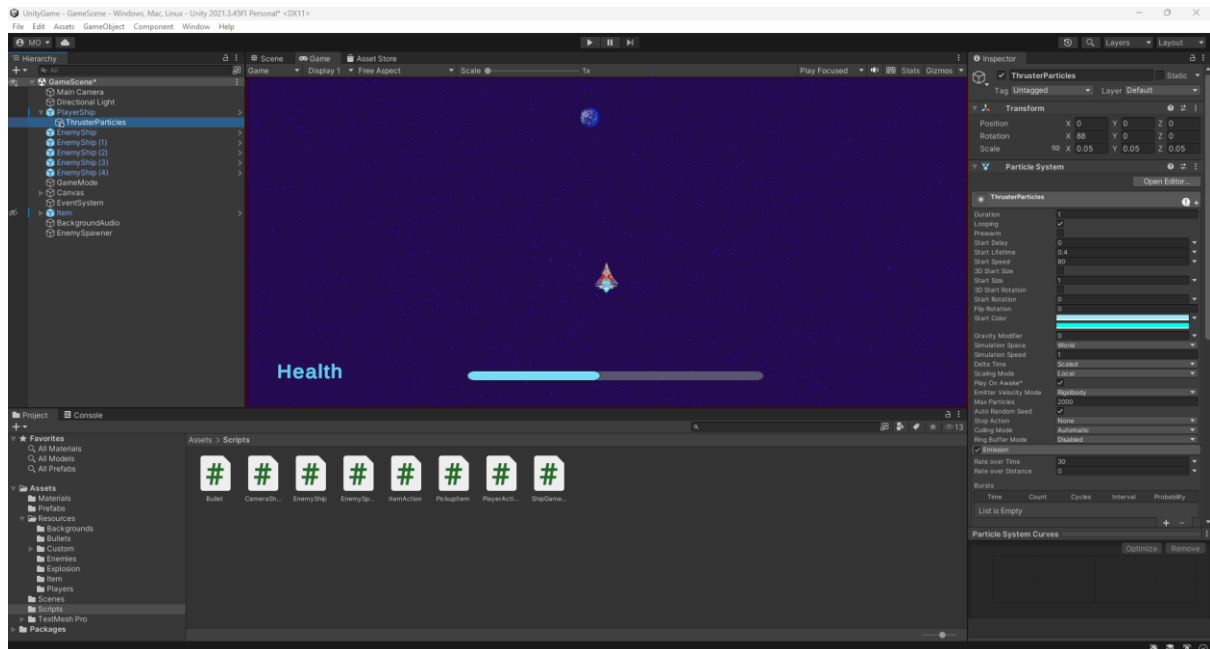


```
1 using UnityEngine;
2 using System.Collections;
3
4 // Unity Script (1 asset reference) 1 reference
5 public class CameraShake : MonoBehaviour
6 {
7     public static CameraShake Instance;
8     private Vector3 originalPosition;
9
10    // Unity Message 10 references
11    void Awake()
12    {
13        Instance = this;
14        originalPosition = transform.localPosition;
15    }
16
17    // reference
18    public void Shake(float duration, float magnitude)
19    {
20        StartCoroutine(ShakeCoroutine(duration, magnitude));
21    }
22
23    // 1 reference
24    private IEnumerator ShakeCoroutine(float duration, float magnitude)
25    {
26        float elapsed = 0f;
27
28        while (elapsed < duration)
29        {
30            float offsetX = Random.Range(-1f, 1f) * magnitude;
31            float offsetY = Random.Range(-1f, 1f) * magnitude;
32
33            transform.localPosition = originalPosition + new Vector3(offsetX, offsetY, 0f);
34
35            elapsed += Time.deltaTime;
36            yield return null;
37        }
38
39        transform.localPosition = originalPosition;
40    }
41 }
```



PlayerShip Thruster Particles:

- Simply right-clicked the PlayerShip object within the hierarchy and selected 'Effects' to add a 'Particle System'. This allowed me to modify the position, colour, trajectory, and speed of the particles to make them come from the thruster on the PlayerShip.



Game Component & Feature 4: Visual and Audio Feedback – Enhanced Dynamic Audio Features

DESCRIPTION

This feature provides enhanced player feedback when enemies are destroyed through either bullet collision or contact with the player's ship. A frame-based explosion animation and synchronized explosion sound effect are triggered at the point of destruction. This contributes to immersion, responsiveness, and audiovisual polish, enhancing the overall gameplay experience.

REASONING

Without feedback, destroying enemies feels unimpactful. By adding a brief but visually and audibly engaging effect, the player receives clear confirmation that their actions (shooting or dodging) had an effect.

This aligns with Component: Visual and Audio Feedback – Feature 1 of the task sheet and contributes to:

- A more dynamic and satisfying combat loop
- Reinforcement of player action
- Maintaining game flow without needing complex transitions or state changes

The visual and audio explosion also connects seamlessly with the wave-based spawning system: each destruction is marked by a short-lived, polished cue before the next wave begins.

BLEUPRINTS/SCRIPTS AND IMPLEMENTATION

Explosion Visual

- **Animation Clip:** Provided in unit resources (Explosion.anim)
- **Animator Controller:** explosion_0001.controller
- **Setup:**
 - Create an empty GameObject named ExplosionEffect
 - Add:
 - Sprite Renderer
 - Animator with assigned controller
 - Optional: AudioSource for synced sound

Explosion Audio

- **Audio Clip:** Provided explosion.wav or similar
- **Setup:**
 - Add AudioSource component
 - Assign the explosion clip

- Set Play on Awake On, Loop Off

Auto-Destruction Script:

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class AutoDestroy : MonoBehaviour
6  {
7      public float lifetime = 1.0f;
8
9      void Start()
10     {
11         Destroy(gameObject, lifetime);
12     }
13 }
14

```

This ensures the explosion object is removed after the animation completes.

EnemyShip Integration

In EnemyShip.cs:

```

39 void OnTriggerEnter2D(Collider2D other)
40 {
41     if (other.CompareTag("Bullet") || other.CompareTag("Player"))
42     {
43         // Create explosion at enemy's position
44         if (explosionPrefab != null)
45         {
46             Instantiate(explosionPrefab, transform.position, Quaternion.identity);
47         }
48
49         if (other.CompareTag("Bullet"))
50             Destroy(other.gameObject); // remove bullet
51
52         Destroy(this.gameObject); // remove enemy
53     }
54 }
55
56

```

The explosionPrefab is dragged into the public field in the EnemyShip prefab via the inspector.

Implementation Steps:

1. Create ExplosionEffect prefab
 - Add Sprite Renderer, Animator, and optional AudioSource
 - Assign explosion_0001.controller and sound clip
2. Attach AutoDestroy.cs to clean it up after 1s
3. Drag into Project panel to turn into a prefab

4. Assign prefab to explosionPrefab in EnemyShip.cs and PlayerActions.cs (if player also explodes)

Extra Transformations or Features

Game Component & Feature 5: Screen Boundary Clamp

DESCRIPTION

To improve control and maintain fair gameplay, the PlayerShip is now restricted from moving off-screen. This ensures the player remains within visible boundaries, even with the new 360° rotation and physics-based movement system.

This keeps the player engaged with on-screen threats and avoids issues where the ship might drift out of view due to thrust or directional momentum.

REASON

Using Viewport coordinates ensures the solution is resolution-independent and responsive across different devices or window sizes. Unlike hardcoded world space limits, this method does not break if the camera zoom or aspect ratio changes.

This complements the ship's new freeform movement and rotation system, especially when using physics-based thrust which may cause the ship to drift. It guarantees a consistent and polished player experience where the ship is always visible and under control.

BLUEPRINTS/SCRIPTS AND IMPLEMENTATION

To enforce screen boundaries dynamically across all screen sizes and aspect ratios, the ship's position is clamped using Unity's Viewport coordinate system. This method converts the world position of the ship into normalized viewport space (0 to 1 range), clamps the values, and converts them back to world coordinates:

```
52 // Keep the player within screen bounds
53 Vector3 pos = Camera.main.WorldToViewportPoint(transform.position);
54 pos.x = Mathf.Clamp(pos.x, 0.05f, 0.95f);
55 pos.y = Mathf.Clamp(pos.y, 0.05f, 0.95f);
56 transform.position = Camera.main.ViewportToWorldPoint(pos);
57
```

This logic is executed at the end of each frame in the Update() method of the PlayerActions script. It effectively prevents the ship from exceeding the screen bounds in any direction, without requiring physical barriers or colliders.

References

Pixabay (n.d) 8-bit Explosion Sound Effect, accessed 29 April 2025. <https://pixabay.com/sound-effects/explosion-8-bit-8-314694/>

Unity Documentation – Quaternion Identity (For EnemyShip), Unity, Accessed 8 May 2025. <https://docs.unity3d.com/6000.1/Documentation/ScriptReference/Quaternion-identity.html>

Unity Documentation - Vector 3 (For EnemyShip movement & PlayerActions movement), Unity, Accessed 7 May 2025. <https://docs.unity3d.com/6000.1/Documentation/ScriptReference/Vector3.html>

Unity Documentation - Collider2D (For EnemyShip & PlayerActions), Unity, Accessed 6 May 2025. <https://docs.unity3d.com/6000.1/Documentation/ScriptReference/Collider2D.html>

Unity Documentation – Rigidbody2D (For EnemyShip, Bullet, PlayerActions), Unity, Accessed 6 May 2025. <https://docs.unity3d.com/6000.1/Documentation/ScriptReference/Rigidbody2D.html>

Unity Documentation - Mathf (For PlayerActions movement and Screen Clamping), Unity, Accessed 7 May 2025. <https://docs.unity3d.com/6000.1/Documentation/ScriptReference/Mathf.html>

Unity Documentation – MouseButton (For PlayerActions shooting and movement), Unity, Accessed 6 May 2025. <https://docs.unity3d.com/6000.1/Documentation/ScriptReference/UIElements.MouseButton.html>

CodeFriend (26 June 2024), **How to make a Camera Shake in Unity (FAST and EASY !)**, YouTube, Accessed 3 May 2025. <https://www.youtube.com/watch?v=7BVAIYrM2FU>

Muddy Wolf (27 April 2023), **Enemy Wave Spawner - Build a 2D Tower Defence Game in Unity #3**, Youtube, Accessed 6 May 2025. <https://www.youtube.com/watch?v=5j8A79-YUo0>

Press Start (19 August 2018), **Unity – Keeping The Player Within Screen Boundaries**, Youtube, Accessed 7 May 2025. https://www.youtube.com/watch?app=desktop&v=ailbszpt_AI&t=35s

Stackoverflow – endless wave spawner question by “notBatman” Feb 21, 2022 at 3:47 <https://stackoverflow.com/questions/71201006/making-an-endless-wave-spawner-that-spawns-harder-enemies-over-time-in-unity>