

PDF Portifolio

Name: Navod Avishka Hewawanithunga

This portfolio showcases code snippets, screenshots, and video links from my Unity projects.

Table of Contents

Project 1: Royal Run.....	1
Project 2: Rocket Booster	6
Project 3: Space Shooter NXT	8
Project 4: Galaxy Striker	11

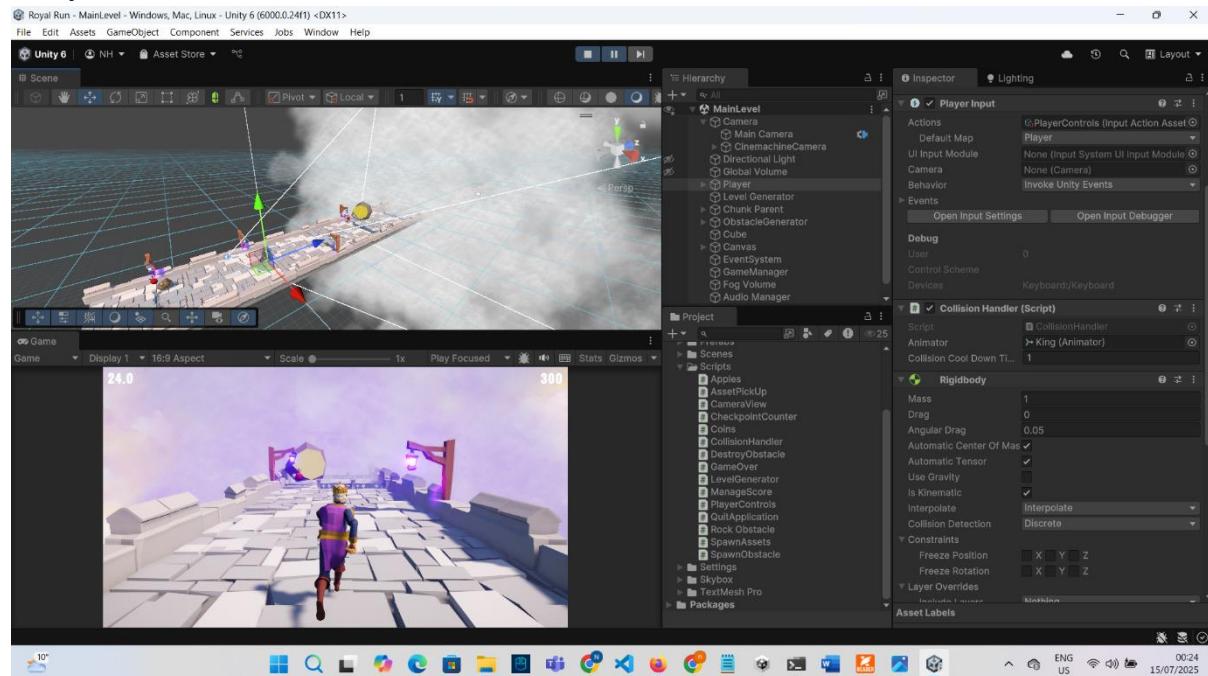
Project 1: Royal Run

Overview

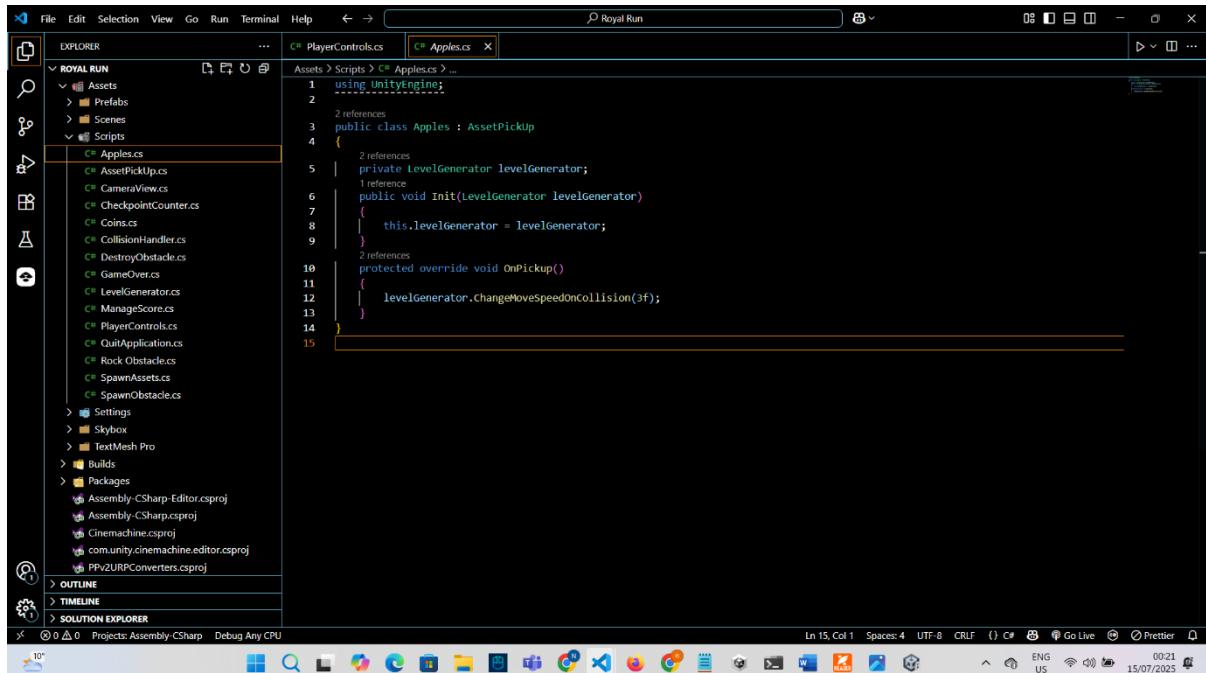
Endless runner with obstacle avoidance, pickup systems, fog lighting, Unity's new Input System, and animation events, Cinemachine and Cinemachine Impulse source. Used object-oriented concepts like Inheritance

Here is the Short Youtube clip of the game: https://youtu.be/qGxxW2_Id3A

Unity Editor



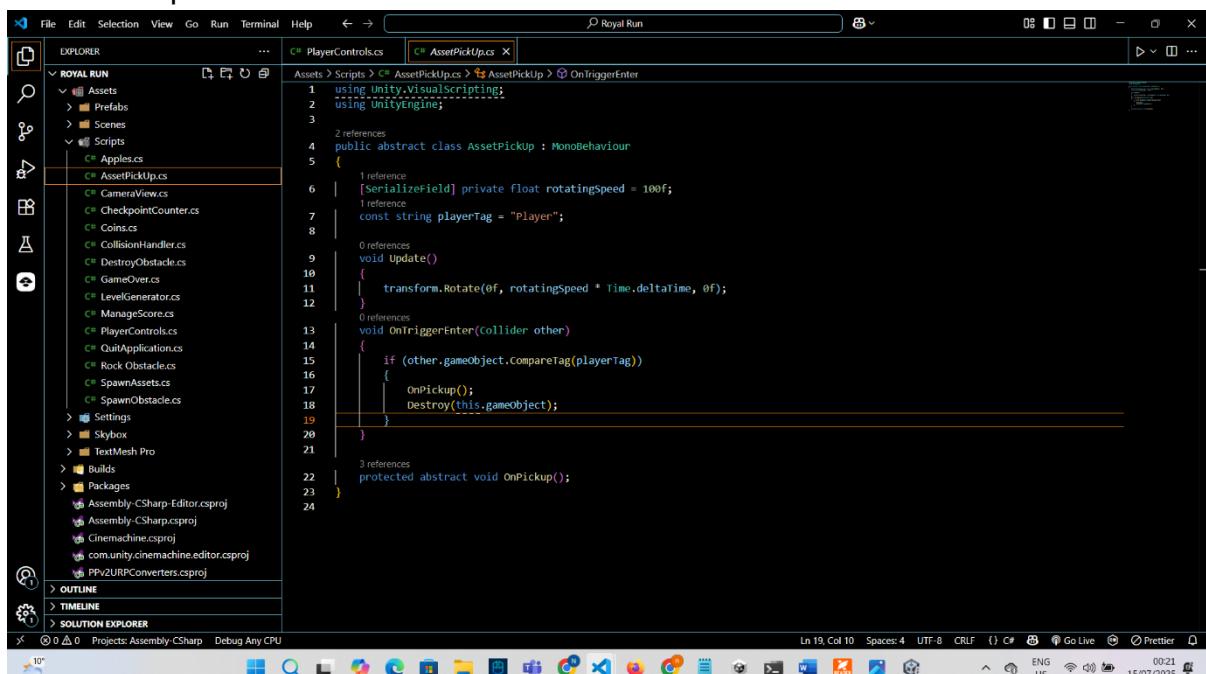
Apples.cs



```
File Edit Selection View Go Run Terminal Help ⌘ ⌘ Royal Run
EXPLORER ... Assets > Scripts > Apples.cs ...
Assets > Scripts > Apples.cs ...
1 using UnityEngine;
2 -----
3 public class Apples : AssetPickUp
4 {
5     private LevelGenerator levelGenerator;
6     public void Init(LevelGenerator levelGenerator)
7     {
8         this.levelGenerator = levelGenerator;
9     }
10    protected override void OnPickup()
11    {
12        levelGenerator.ChangeMoveSpeedOnCollision(3f);
13    }
14 }
15 
```

The screenshot shows the Unity Editor's script editor window. The file path is Assets > Scripts > Apples.cs. The code defines a class Apples that inherits from AssetPickUp. It has a private field levelGenerator and a public method Init that takes a LevelGenerator as a parameter. The class also overrides the protected OnPickup method to change move speed on collision.

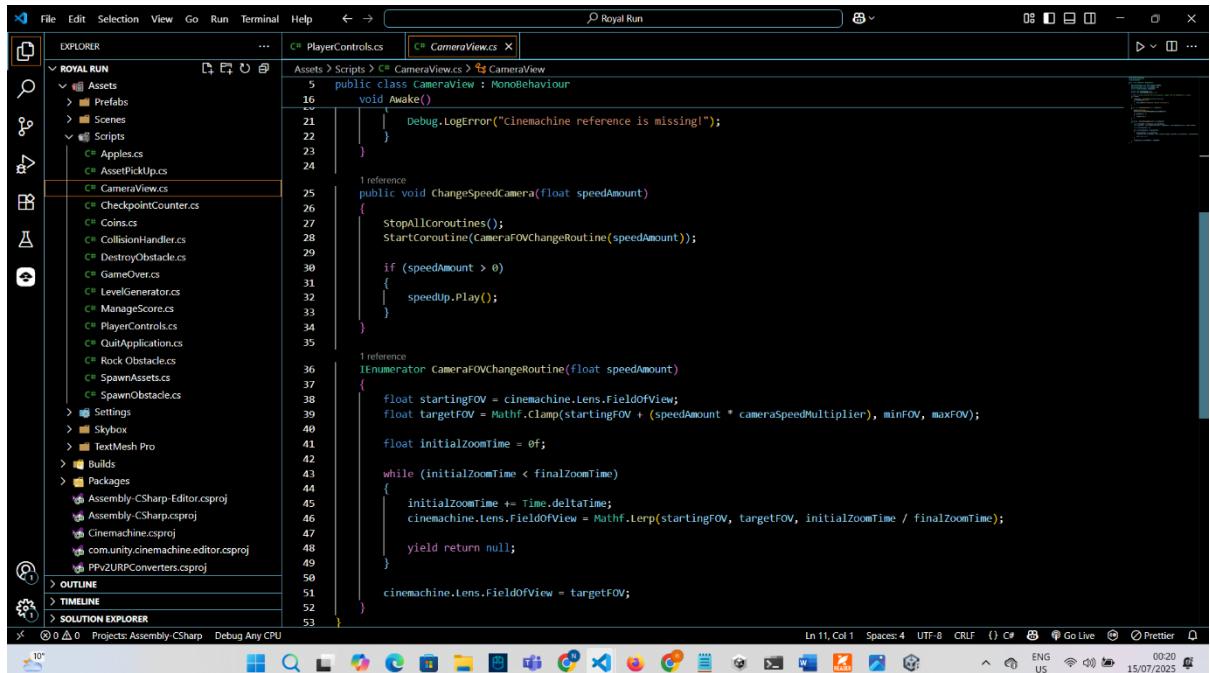
AssetPickUp.cs



```
File Edit Selection View Go Run Terminal Help ⌘ ⌘ Royal Run
EXPLORER ... Assets > Scripts > AssetPickUp.cs ...
Assets > Scripts > AssetPickUp.cs ...
1 using Unity.VisusalScripting;
2 using UnityEngine;
3 -----
4 public abstract class AssetPickUp : MonoBehaviour
5 {
6     [SerializeField] private float rotatingSpeed = 100f;
7     const string playerTag = "Player";
8     void Update()
9     {
10        transform.Rotate(0f, rotatingSpeed * Time.deltaTime, 0f);
11    }
12    void OnTriggerEnter(Collider other)
13    {
14        if (other.gameObject.CompareTag(playerTag))
15        {
16            OnPickup();
17            Destroy(this.gameObject);
18        }
19    }
20    protected abstract void OnPickup();
21 }
22 
```

The screenshot shows the Unity Editor's script editor window. The file path is Assets > Scripts > AssetPickUp.cs. The code defines an abstract class AssetPickUp that inherits from MonoBehaviour. It has a public field rotatingSpeed and a constant playerTag. The class overrides the protected abstract OnPickup method and implements the OnTriggerEnter event to destroy itself when it collides with a player.

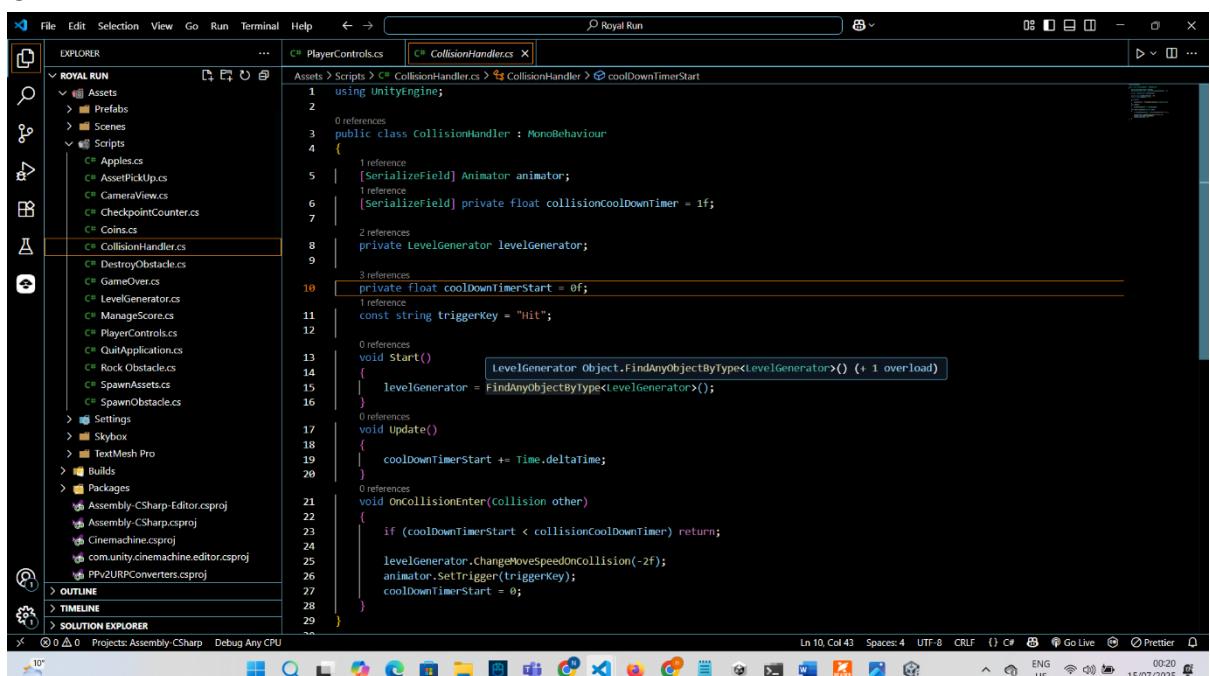
CameraView.cs



The screenshot shows the Visual Studio IDE with the CameraView.cs file open in the editor. The code implements a camera movement system using Cinemachine. It includes methods for changing camera speed and zooming in/out over time.

```
Assets > Scripts > CameraView.cs
5 public class CameraView : MonoBehaviour
6 {
7     void Awake()
8     {
9         Debug.LogError("Cinemachine reference is missing!");
10    }
11
12    void ChangeSpeedCamera(float speedAmount)
13    {
14        StopAllCoroutines();
15        StartCoroutine(CameraFOVChangeRoutine(speedAmount));
16    }
17
18    if (speedAmount > 0)
19    {
20        speedUp.Play();
21    }
22
23    IEnumerator CameraFOVChangeRoutine(float speedAmount)
24    {
25        float startingFOV = cinemachine.Lens.FieldOfView;
26        float targetFOV = Mathf.Clamp(startingFOV + (speedAmount * cameraSpeedMultiplier), minFOV, maxFOV);
27
28        float initialZoomTime = 0f;
29
30        while (initialZoomTime < finalZoomTime)
31        {
32            initialZoomTime += Time.deltaTime;
33            cinemachine.Lens.FieldOfView = Mathf.Lerp(startingFOV, targetFOV, initialZoomTime / finalZoomTime);
34        }
35
36        yield return null;
37    }
38
39    cinemachine.Lens.FieldOfView = targetFOV;
40
41 }
```

CollisionHandler.cs



The screenshot shows the Visual Studio IDE with the CollisionHandler.cs file open in the editor. This script handles collisions between objects and a level generator, applying a temporary speed reduction.

```
Assets > Scripts > CollisionHandler.cs
1 using UnityEngine;
2
3 public class CollisionHandler : MonoBehaviour
4 {
5     [SerializeField] Animator animator;
6     [SerializeField] private float collisionCoolDownTimer = 1f;
7
8     private LevelGenerator levelGenerator;
9
10    private float coolDownTimerStart = 0f;
11    const string triggerKey = "Hit";
12
13    void Start()
14    {
15        levelGenerator = FindAnyObjectByType<LevelGenerator>(); (+ 1 overload)
16    }
17
18    void Update()
19    {
20        coolDownTimerStart += Time.deltaTime;
21    }
22
23    void OnCollisionEnter(Collision other)
24    {
25        if (coolDownTimerStart < collisionCoolDownTimer) return;
26
27        levelGenerator.ChangeMoveSpeedOnCollision(-2f);
28        animator.SetTrigger(triggerKey);
29        coolDownTimerStart = 0f;
30    }
31 }
```

LevelGenerator.cs

The screenshot shows the Unity Editor interface with the code editor open. The file path is Assets > Scripts > LevelGenerator.cs. The code defines a MonoBehaviour class named LevelGenerator. It includes methods for spawning chunks and managing scores.

```

5 public class LevelGenerator : MonoBehaviour
63 }
64
65 private void SpawnChunk()
66 {
67     float nextChunkPos = CalculateNewSpawnChunks();
68     Vector3 nextChunkPos = new Vector3(0, 0, 1) * nextChunkPos;
69     GameObject chunkPrefab = ProcessSpawnChunkVariant();
70     GameObject newchunk = Instantiate(chunkPrefab, transform.position + nextChunkPos, quaternion.identity, chunkParent.transform);
71     spawnAssets.chunk = newchunk.GetComponent<spawnAssets>();
72     chunksList.Add(newchunk);
73     chunk.Init(this, manageScore);
74
75     noOfChunksSpawned++;
76
77 }
78
79 private GameObject ProcessSpawnChunkVariant()
80 {
81     GameObject chunkPrefab;
82
83     if (noOfChunksSpawned % chunkInterval == 0 && noOfChunksSpawned != 0)
84     {
85         chunkPrefab = checkpointPrefab;
86     }
87     else
88     {
89         chunkPrefab = chunkPrefabs[UnityEngine.Random.Range(0, chunkPrefabs.Length)];
90     }
91
92     return chunkPrefab;
93 }
94
95 private void movechunks()
96 {
97 }

```

Rock Obstacle.cs

The screenshot shows the Unity Editor interface with the code editor open. The file path is Assets > Scripts > RockObstacle.cs. The code defines a MonoBehaviour class named RockObstacle. It includes methods for updating the timer and generating collision impulses.

```

5 public class RockObstacle : MonoBehaviour
18
19 void Update()
20 {
21     startTimer += Time.deltaTime;
22 }
23
24 void OnCollisionEnter(Collision other)
25 {
26     if (startTimer < coolDownTimer) return;
27
28     GenerateCCImpulse();
29     ProcessRocksParticleFx(other);
30     startTimer = 0f;
31 }
32
33 private void GenerateCCImpulse()
34 {
35     float distance = Vector3.Distance(transform.position, Camera.main.transform.position);
36     float shockConstant = (1f / distance) * shockModifier;
37     float shockValue = Mathf.Min(shockConstant, distance);
38     cinemachineImpulseSource.GenerateImpulse(shockValue);
39 }
40
41 private void ProcessRocksParticleFx(Collision other)
42 {
43     ContactPoint contactPoint = other.contacts[0];
44     rocksEffects.transform.position = contactPoint.point;
45     rocksEffects.Play();
46     boulderAudio.Play();
47 }
48
49

```

SpawnObstacle.cs

```
File Edit Selection View Go Run Terminal Help < > Royal Run Assets > Scripts > PlayerControls.cs > SpawnObstacle.cs > ChangeObstacleSpawnTimer
```

```
6 public class SpawnObstacle : MonoBehaviour
7 {
8     [SerializeField] private GameObject[] obstaclePrefabs;
9     [SerializeField] private Transform obstacleSpawn;
10    [SerializeField] private float obstacleSpawnTimer = 3f;
11    [SerializeField] private float minObstacleSpawnTimer = .15f;
12
13    void Start()
14    {
15        StartCoroutine(ObstacleGeneratorRoutine());
16    }
17
18    public void ChangeObstacleSpawnTimer(float time)
19    {
20        obstacleSpawnTimer -= time;
21
22        if (obstacleSpawnTimer <= minObstacleSpawnTimer) obstacleSpawnTimer = minObstacleSpawnTimer;
23    }
24
25    IEnumerator ObstacleGeneratorRoutine()
26    {
27        while (true)
28        {
29            GameObject obstaclePrefab = obstaclePrefabs[UnityEngine.Random.Range(0, obstaclePrefabs.Length)];
30            Vector3 obstacleSpawnPos = new Vector3(UnityEngine.Random.Range(-4.5f, 4.5f), transform.position.y, transform.position.z);
31            Instantiate(obstaclePrefab, obstacleSpawnPos, UnityEngine.Random.rotation, ___gameObject.transform);
32            yield return new WaitForSeconds(obstacleSpawnTimer);
33        }
34    }
35}
```

File Edit Selection View Go Run Terminal Help < > Royal Run Assets > Scripts > PlayerControls.cs > SpawnObstacle.cs > ChangeObstacleSpawnTimer

6 public class SpawnObstacle : MonoBehaviour

7 {

8 [SerializeField] private GameObject[] obstaclePrefabs;

9 [SerializeField] private Transform obstacleSpawn;

10 [SerializeField] private float obstacleSpawnTimer = 3f;

11 [SerializeField] private float minObstacleSpawnTimer = .15f;

12

13 void Start()

14 {

15 StartCoroutine(ObstacleGeneratorRoutine());

16 }

17

18 public void ChangeObstacleSpawnTimer(float time)

19 {

20 obstacleSpawnTimer -= time;

21

22 if (obstacleSpawnTimer <= minObstacleSpawnTimer) obstacleSpawnTimer = minObstacleSpawnTimer;

23 }

24

25 IEnumerator ObstacleGeneratorRoutine()

26 {

27 while (true)

28 {

29 GameObject obstaclePrefab = obstaclePrefabs[UnityEngine.Random.Range(0, obstaclePrefabs.Length)];

30 Vector3 obstacleSpawnPos = new Vector3(UnityEngine.Random.Range(-4.5f, 4.5f), transform.position.y, transform.position.z);

31 Instantiate(obstaclePrefab, obstacleSpawnPos, UnityEngine.Random.rotation, ___gameObject.transform);

32 yield return new WaitForSeconds(obstacleSpawnTimer);

33 }

34 }

35}

SpawnAssets.cs

The screenshot shows the Unity Editor interface with the following details:

- File Explorer:** Shows the project structure under "ROYAL RUN". The "SpawnAssets.cs" script is selected.
- Code Editor:** Displays the C# code for "SpawnAssets.cs".
- Code Navigation:** Shows references and definitions for various methods and variables.
- Bottom Status Bar:** Shows file paths, line numbers, and other editor settings.

```
Assets > Scripts > SpawnAssets.cs > SpawnAssets > SpawnCoins
5 public class SpawnAssets : MonoBehaviour
{
    // Reference
    public void Init(LevelGenerator levelGenerator, ManageScore manageScore)
    {
        this.levelGenerator = levelGenerator;
        this.manageScore = manageScore;
    }

    // Reference
    private void SpawnFences()
    {
        int spawnFenceRandom = UnityEngine.Random.Range(0, 3);

        for (int i = 0; i < spawnFenceRandom; i++)
        {
            if (availableLane.Count == 0) break;

            int selectedLane = ProcessSelectLane();
            Vector3 lanePos = new Vector3(laneXVal[selectedLane], transform.position.y, transform.position.z);
            Instantiate(fencePrefab, lanePos, quaternion.identity, this.transform);
        }
    }

    // References
    private int ProcessSelectLane()
    {
        int randomLaneIndex = UnityEngine.Random.Range(0, availableLane.Count);
        int selectedLane = availableLane[randomLaneIndex];
        availableLane.RemoveAt(randomLaneIndex);
        return selectedLane;
    }

    // Reference
    private void SpawnApple()
    {
        if (UnityEngine.Random.value > spawnAppleChance || availableLane.Count <= 0) return;
    }
}
```

PlayerControls.cs

The screenshot shows the Visual Studio Code interface with the file 'PlayerControls.cs' open in the editor. The code implements a MonoBehavior for player movement using Rigidbody physics. It includes methods for Awake, FixedUpdate, and PlayerMovement, which calculates new position based on movement input and current position.

```
public class PlayerControls : MonoBehaviour
{
    [SerializeField] private float moveSpeed = 10f;
    private Vector2 movement;
    private Rigidbody rb;

    void Awake()
    {
        rb = GetComponent<Rigidbody>();
    }

    private void FixedUpdate()
    {
        PlayerMovement();
    }

    private void PlayerMovement()
    {
        Vector3 currentPos = rb.position;
        Vector3 moveToNewPos = currentPos + new Vector3(movement.x, 0, movement.y) * moveSpeed * Time.fixedDeltaTime;
        float xVal = Mathf.Clamp(moveToNewPos.x, -4f, 4f);
        float zVal = Mathf.Clamp(moveToNewPos.z, -2.8f, 4f);
        Vector3 moveToClampPos = new Vector3(xVal, 0, zVal);
        rb.MovePosition(moveToClampPos);
    }

    public void MoveControls(InputAction.CallbackContext value)
    {
        movement = value.ReadValue<Vector2>();
    }
}
```

Project 2: Rocket Booster

Overview: A 3D rocket navigation game using Rigidbody physics, Cinemachine, particle systems, and level progression logic.

Here is the youtube video clip of the game: <https://youtu.be/1MA5DufkdFA>

Movement.cs

The screenshot shows the Visual Studio Code interface with the file 'Movement.cs' open in the editor. The code implements a MonoBehavior for rocket movement, handling thrust and rotation. It uses a Thrust component and an AudioSource for sound effects.

```
public class Movement : MonoBehaviour
{
    void Start()
    {
        rb = GetComponent<Rigidbody>();
    }

    void FixedUpdate()
    {
        processThrust();
        processRotation();
    }

    private void processThrust()
    {
        if (thrust.IsPressed())
        {
            rb.AddRelativeForce(Vector3.up * thrustStrength * Time.fixedDeltaTime);
            if (!audioSource.isPlaying)
            {
                audioSource.PlayOneShot(audioClip);
            }
            if (!_mainThrustParticles.isPlaying)
            {
                _mainThrustParticles.Play();
            }
        }
        else
        {
            _mainThrustParticles.Stop();
            audioSource.Stop();
        }
    }
}
```

Oscillator.cs

The screenshot shows the Visual Studio Code interface with the file 'Oscillator.cs' open in the editor. The code defines a MonoBehavior named 'Oscillator' with methods for Start and Update.

```
1 using UnityEngine;
2
3 public class Oscillator : MonoBehaviour
4 {
5     [SerializeField] private Vector3 movementVector;
6     [SerializeField] private float speed;
7     private Vector3 startPosition;
8     private Vector3 endPosition;
9     private float movementFactor;
10    // Start is called once before the first execution of Update after the MonoBehaviour is created
11    void Start()
12    {
13        startPosition = transform.position;
14        endPosition = startPosition + movementVector;
15    }
16
17    // Update is called once per frame
18    void Update()
19    {
20        movementFactor = Mathf.PingPong(Time.time * speed, 1f);
21        transform.position = Vector3.Lerp(startPosition, endPosition, movementFactor);
22    }
23}
```

The Explorer sidebar shows the project structure, including files like CollisionHandler.cs, Movement.cs, and Oscillator.cs. The bottom status bar shows the file is 14 lines long, column 27, and the date is 14/07/2025.

CollisionHandler.cs

The screenshot shows the Visual Studio Code interface with the file 'CollisionHandler.cs' open in the editor. The code defines a MonoBehavior named 'CollisionHandler' with a method for OnCollisionEnter.

```
1 using UnityEngine;
2
3 public class CollisionHandler : MonoBehaviour
4 {
5     void OnCollisionEnter(Collision other)
6     {
7         if(!isControllable || !isCollidable) { return; }
8
9         switch(other.gameObject.tag)
10        {
11            case "start":
12                Debug.Log("At the starting pad: " + other.gameObject.tag);
13                break;
14            case "Finish":
15                StartDelaySequence(0);
16                break;
17            default:
18                StartDelaySequence(1);
19                break;
20        }
21
22        private void StartDelaySequence(int seq)
23        {
24            isControllable = false;
25            audioSource.Stop();
26            audioSource.PlayOneShot(audioClips[seq]);
27            GetComponent<Movement>().enabled = false;
28            if(seq == 1)
29            {
30                explosionParticles.Play();
31                Invoke("LoadCurrentScene", delaySeconds);
32            }
33            else if(seq == 0)
34            {
35                successParticles.Play();
36                Invoke("LoadNextScene", delaySeconds);
37            }
38        }
39    }
40}
```

The Explorer sidebar shows the project structure, including files like CollisionHandler.cs, Movement.cs, and Oscillator.cs. The bottom status bar shows the file is 61 lines long, column 50, and the date is 14/07/2025.

QuitApplication.cs

The screenshot shows the Visual Studio Code interface with the file 'QuitApplication.cs' open in the center editor pane. The code implements a MonoBehaviour with Start and Update methods. The Start method checks if the escape key is pressed and calls Application.Quit(). The Update method logs a message to the console.

```
1 using UnityEngine;
2 using UnityEngine.InputSystem;
3
4 public class QuitApplication : MonoBehaviour
5 {
6     // Start is called once before the first execution of Update after the MonoBehaviour is created
7     void Start()
8     {
9     }
10
11     // Update is called once per frame
12     void Update()
13     {
14         if (Keyboard.current.escapeKey.isPressed)
15         {
16             Application.Quit();
17             Debug.Log("Quit the application");
18         }
19     }
20 }
21
22 }
```

Project 3: Space Shooter NXT

Overview

A 2d Game developed using Unity (2019.1.5f1). A spaceship moving right to left while killing enemies using lasers, while having clamped movement up and down.

Here is the youtube clip of the game: <https://youtu.be/q08RrHWISYQ>

Player.cs

The screenshot shows the Visual Studio Code interface with the file 'Player.cs' open in the center editor pane. The code defines a MonoBehaviour with Start and Update methods. The Update method handles player input for firing lasers and calculating movement. It uses CrossPlatformInputManager to detect key presses and mouse button down events. The calculateMovement method uses Input.GetAxis to get horizontal and vertical input and applies it to the transform's position with clamping.

```
1 public class Player : MonoBehaviour
2 {
3     void Start()
4     {
5     }
6
7     // Update is called once per frame
8     void Update()
9     {
10        calculateMovement();
11
12        #if UNITY_ANDROID
13            if (Input.GetKeyDown(KeyCode.Space) || CrossPlatformInputManager.GetButtonDown("Fire") && Time.time > _cantFire)
14            {
15                FireLaser();
16            }
17        #else
18            if (Input.GetKeyDown(KeyCode.Space) || Input.GetMouseButton(0) && Time.time > _cantFire)
19            {
20                FireLaser();
21            }
22        #endif
23    }
24
25    void calculateMovement()
26    {
27        float horizontalInput = CrossPlatformInputManager.GetAxis("Horizontal"); //Input.GetAxis("Horizontal");
28        float verticalInput = CrossPlatformInputManager.GetAxis("Vertical"); //Input.GetAxis("Vertical");
29
30        transform.Translate(new Vector3(horizontalInput, verticalInput, 0) * _speed * Time.deltaTime);
31
32        transform.position = new Vector3(transform.position.x, Mathf.Clamp(transform.position.y, -3.92f, 0), 0);
33    }
34 }
```

Enemy.cs

The screenshot shows the VS Code interface with the file `Enemy.cs` open in the editor. The code defines a MonoBehavior script for an enemy. It includes methods for calculating movement and handling collisions with the player.

```
public class Enemy : MonoBehaviour
{
    void Update()
    {
        CalculateMovement();
    }

    void CalculateMovement()
    {
        transform.Translate(Vector3.down * _speed * Time.deltaTime);

        if (transform.position.y < -5f)
        {
            float randomX = Random.Range(-9.45f, 9.45f);
            transform.position = new Vector3(randomX, 8.0f, 0);
        }
    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.name == "Player")
        {
            Player player = other.GetComponent<Player>();
            if (player != null)
            {
                player.PlayerDestroy();
            }

            _EnemyAnim.SetTrigger("OnEnemyDeath");
            _speed = 0f;
            AudioSource.Play();
            GetComponent<BoxCollider2D>().enabled = false;
            //Destroy(GetComponent<BoxCollider2D>());
            Destroy(this.gameObject, 2.4f);
        }
    }
}
```

Laser.cs

The screenshot shows the VS Code interface with the file `Laser.cs` open in the editor. The code defines a MonoBehavior script for a laser. It includes methods for moving up and down, and a public method to set it as an enemy laser.

```
public class Laser : MonoBehaviour
{
    void MoveUp()
    {
        transform.Translate(Vector3.up * _speed * Time.deltaTime);

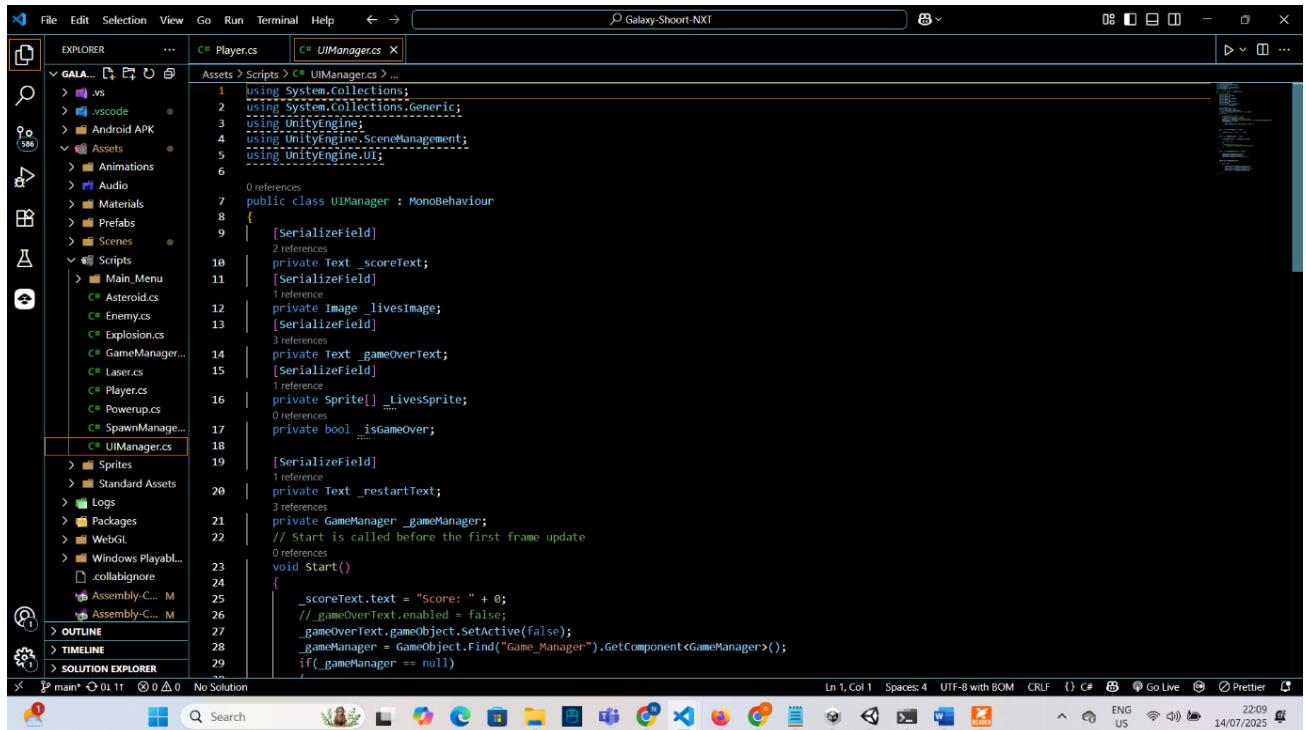
        if (transform.position.y > 8f)
        {
            if (this.transform.parent != null)
            {
                Destroy(this.transform.parent.gameObject);
            }
            Destroy(this.gameObject);
        }
    }

    void MoveDown()
    {
        transform.Translate(Vector3.down * _speed * Time.deltaTime);

        if (transform.position.y < -7f)
        {
            if (this.transform.parent != null)
            {
                Destroy(this.transform.parent.gameObject);
            }
            Destroy(this.gameObject);
        }
    }

    public void EnemyLaser()
    {
        _isEnemyLaser = true;
    }
}
```

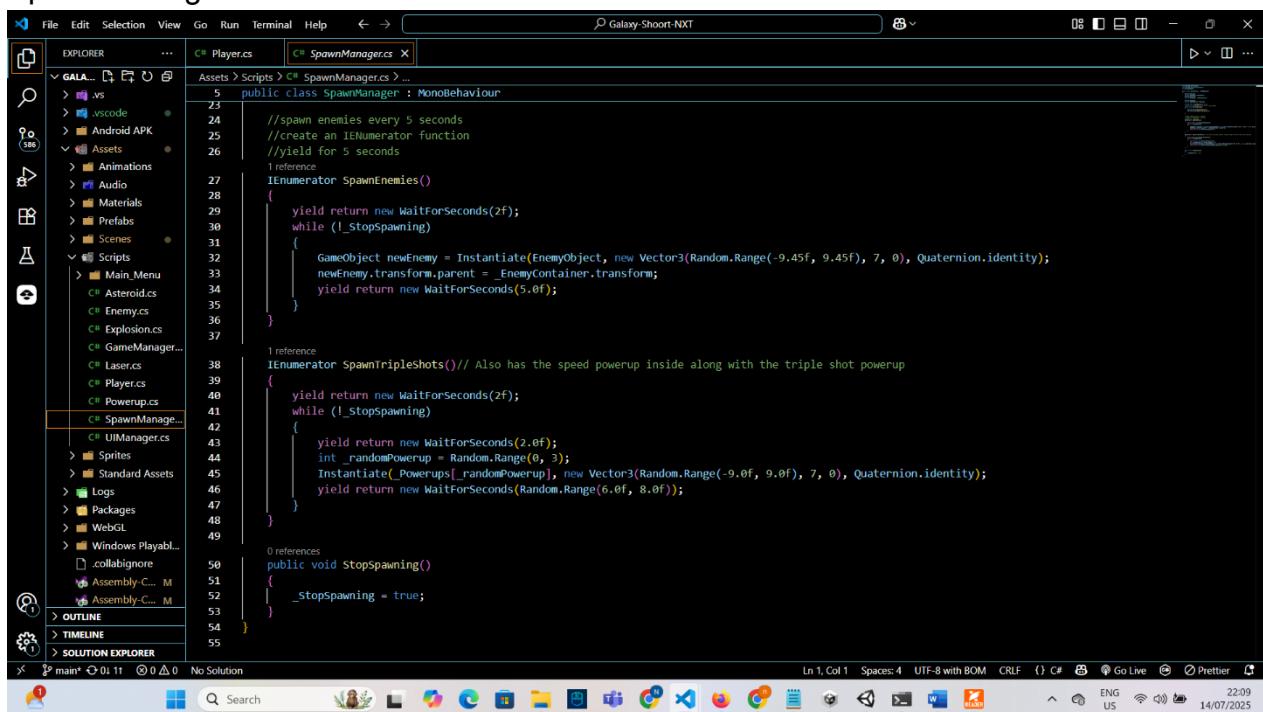
UIManager.cs



The screenshot shows the Visual Studio Code interface with the file `UIManager.cs` open in the editor. The code defines a `UIManager` class that inherits from `MonoBehaviour`. It contains fields for score text, lives image, game over text, and a list of lives sprites. The `Start` method initializes these fields and starts a coroutine to spawn enemies every 5 seconds. The `StopSpawning` method stops the spawning process.

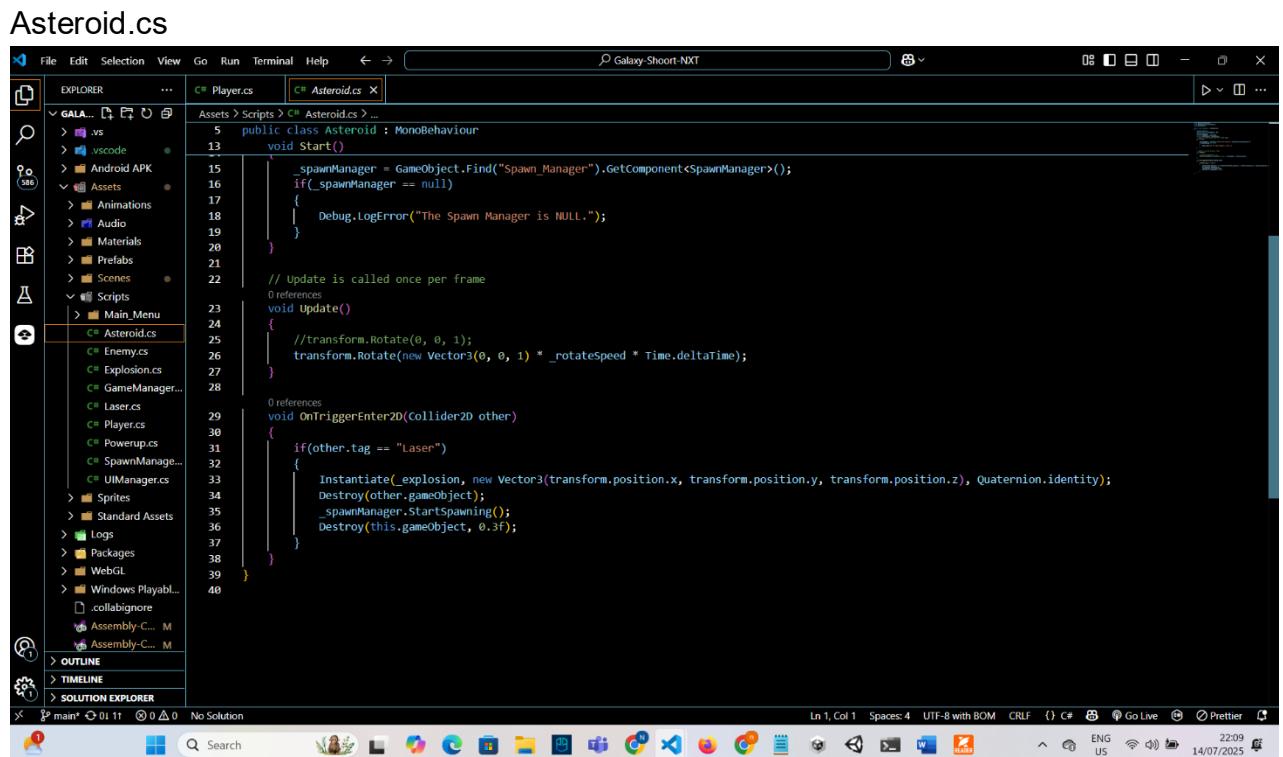
```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4 using UnityEngine.SceneManagement;
5 using UnityEngine.UI;
6
7 public class UIManager : MonoBehaviour
8 {
9     [SerializeField]
10    private Text _scoreText;
11    [SerializeField]
12    private Image _livesImage;
13    [SerializeField]
14    private Text _gameOverText;
15    [SerializeField]
16    private Sprite[] _livesSprite;
17
18    [SerializeField]
19    private Text _restartText;
20
21    private GameManager _gameManager;
22
23    // Start is called before the first frame update
24    void Start()
25    {
26        _scoreText.text = "Score: " + 0;
27        //_gameOverText.enabled = false;
28        _gameOverText.gameObject.SetActive(false);
29        _gameManager = GameObject.Find("Game_Manager").GetComponent<GameManager>();
30        if (_gameManager == null)
```

SpawnManager.cs



The screenshot shows the Visual Studio Code interface with the file `SpawnManager.cs` open in the editor. The code defines a `SpawnManager` class that inherits from `MonoBehaviour`. It contains two enumerators: `SpawnEnemies()` and `SpawnTripleShots()`. The `SpawnEnemies()` enumerator spawns enemies every 5 seconds. The `SpawnTripleShots()` enumerator also spawns enemies, but includes a speed powerup and a triple shot powerup. Both enumerators yield returns after each wait or instantiation. The `StopSpawning` method stops the spawning process.

```
5 public class SpawnManager : MonoBehaviour
6
7     //spawn enemies every 5 seconds
8     //create an IEnumerator function
9     //yield for 5 seconds
10
11     IEnumerator SpawnEnemies()
12     {
13         yield return new WaitForSeconds(2f);
14         while (!_stopSpawning)
15         {
16             GameObject newEnemy = Instantiate(EnemyObject, new Vector3(Random.Range(-9.45f, 9.45f), 7, 0), Quaternion.identity);
17             newEnemy.transform.parent = _EnemyContainer.transform;
18             yield return new WaitForSeconds(5.0f);
19         }
20     }
21
22     IEnumerator SpawnTripleShots()// Also has the speed powerup inside along with the triple shot powerup
23     {
24         yield return new WaitForSeconds(2f);
25         while (!_stopSpawning)
26         {
27             yield return new WaitForSeconds(2.0f);
28             int _randomPowerup = Random.Range(0, 3);
29             Instantiate(_Powerups[_randomPowerup], new Vector3(Random.Range(-9.0f, 9.0f), 7, 0), Quaternion.identity);
30             yield return new WaitForSeconds(Random.Range(6.0f, 8.0f));
31         }
32     }
33
34     public void StopSpawning()
35     {
36         _stopSpawning = true;
37     }
38
39
40
41
42
43
44
45
46
47
48
49
```



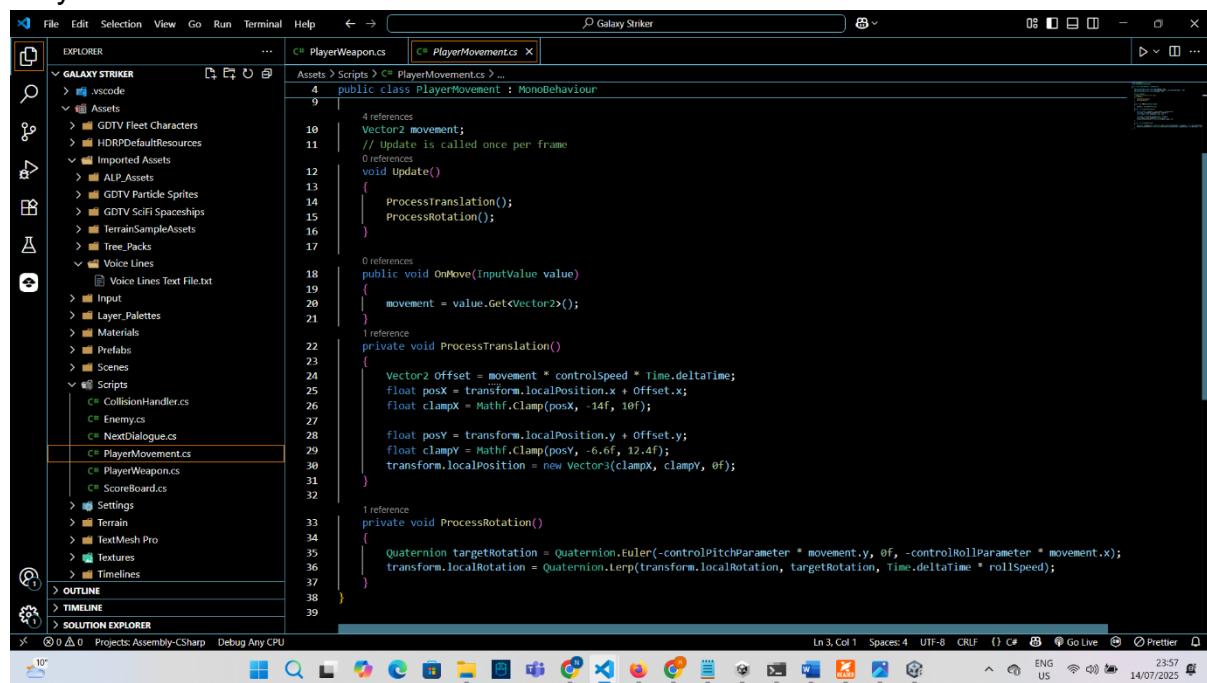
Project 4: Galaxy Striker

Overview

Space combat game featuring timeline-based movement, fog effects, prefab variants, and terrain tool usage.

Here is the youtube clip of the game: <https://youtu.be/09QRfdK5YKA>

PlayerMovement.cs



PlayerWeapon.cs

The screenshot shows the Visual Studio Code interface with the file 'PlayerWeapon.cs' open in the editor. The code defines a class 'PlayerWeapon' that inherits from 'MonoBehaviour'. It includes methods for handling input, processing fire, moving the target, and laser aiming. The code uses Unity's Input system and Particle Systems.

```
public class PlayerWeapon : MonoBehaviour
{
    public void OnFire(InputValue value)
    {
        isFiring = value.isPressed;
    }

    private void ProcessFiring()
    {
        crosshair.position = Input.mousePosition;
        foreach (GameObject laser in lasers)
        {
            ParticleSystem.EmissionModule laserEmission = laser.GetComponent<ParticleSystem>().emission;
            laserEmission.enabled = isFiring;
        }
    }

    private void MoveTarget()
    {
        Vector3 targetPos = new Vector3(Input.mousePosition.x, Input.mousePosition.y, targetPosZ);
        target.position = Camera.main.ScreenToWorldPoint(targetPos);
    }

    private void LaserAiming()
    {
        foreach (GameObject laser in lasers)
        {
            Vector3 distanceDiff = target.position - this.transform.position;
            laser.transform.rotation = Quaternion.LookRotation(distanceDiff);
        }
    }
}
```

Enemy.cs

The screenshot shows the Visual Studio Code interface with the file 'Enemy.cs' open in the editor. The code defines a class 'Enemy' that inherits from 'MonoBehaviour'. It includes methods for starting, handling particle collisions, and processing collision effects. The script uses Unity's Physics and Scoreboard components.

```
public class Enemy : MonoBehaviour
{
    [SerializeField] private float hitPoints = 6f;
    [SerializeField] private int scorePoints = 10;

    private ScoreBoard scoreBoard;

    void Start()
    {
        scoreBoard = FindFirstObjectByType<ScoreBoard>();
    }

    private void OnParticleCollision(GameObject other)
    {
        ProcessCollisionEffects();
    }

    private void OnTriggerEnter(Collider other)
    {
        ProcessCollisionEffects();
    }

    private void ProcessCollisionEffects()
    {
        hitPoints--;
        if (hitPoints <= 0)
        {
            Instantiate(Des_Enemy_VFX, transform.position, Quaternion.identity);
            scoreBoard.ProcessScore(scorePoints);
            Destroy(this.gameObject);
        }
    }
}
```

Unity Editor

