Rocket Launch

Game Design Document

Github

Story

- humanity has explored the far reaches of the galaxy and established a network of space stations. Amidst these adventures, You are the astronaut, You discover a powerful energy crystal on an alien planet, Zephyria. This crystal has the potential to revolutionize space travel and bring prosperity to humanity. So you took the Crystal (Zephyria) from the alien plant and launched the rocket to get away from the Alien Spaceship. And the Lord (Zarkon) of that planet wants to destroy you.

Objective

- You have to Launch the Rocket (by Slingshot) and Avoid obstacles that come in space(by Tap left and right) and there will be fuel getting low so you also have to collect Fuel and some powerups to Reach your Space station anyhow.

Difficulty and progress

- There will be multiple levels, In the Starting there will be Less difficulty with less number of obstacles(asteroids, space particles, Black Hole) and speed.
- By the level upgrading there will be more obstacles (asteroids, space particles, Black Hole) and speed, you have to collect the stars in between also there will be fuel draining during the level so in between the level there will be fuels you have to collect to complete the level.
- There will be BlackHoles in between which can take you inside it and if you lose your objective then you have to restart the level.
- You have to make each decision perfectly else you either lose fuel or get crashed with asteroids or the black holes take you inside.
- By the time you will also get a wormhole which can help you to teleport from one place to another. It will be your decision whether you pass through it or not.

- Boss level- There will be Other UFO’s who are trying to kill you. Avoid them and win the level.

Genre: Hyper Casual

Subgenre: Arcade

Theme: Space

Mechanics:

- Slingshotting,Tap left and Right(Navigation), Double tap (powerups)

GamePlay Loop

The rocket will launch in space and you have to navigate it using tap left for left and tap right for right. You have to avoid obstacles and Get into your space station. You have to take things in mind such as obstacles, Fuel capacity, and also you will have to use power ups for better strategy.

Perspective/Orientation : 2D/Portrait

Artstyle: 2D Cartoonish

Target Audience: 8+

MindMap - Click here

UserFlow - Click here

WireFrame - Click here

Gameplay

Launch the rocket and avoid obstacles such as Asteroids, space particles, and Black holes. Use power ups for your benefit. Also make sure to collect Fuel else you would end up Fuel over.

Launch - Drag back and release it to launch the rocket.

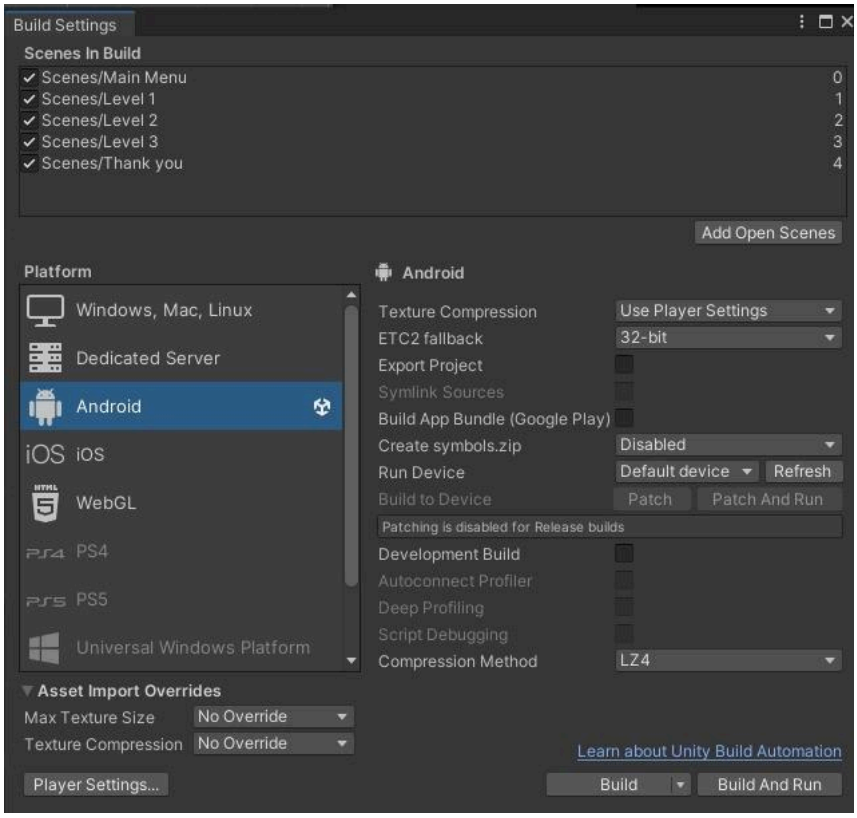
Navigation - Tap Left or Right to navigate.

Powerup - Select the powerup and double tap.

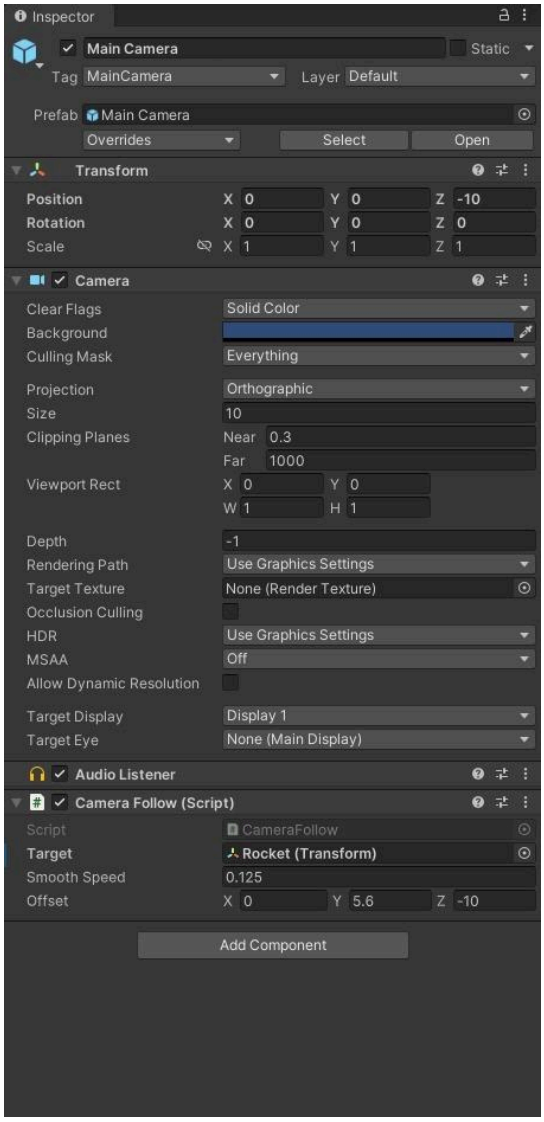
Monetization:

- In-App purchase: combo pack(eg.,100 stars , rocket skin), star packs(eg.,50 stars) , Noads, Universe, Theme based Packs.
- Ads : Banner ad at the bottom throughout gameplay, interstitial ad at the end of every 2 levels, rewarded ad to multiply the stars earned in the level.

Unity Build Settings:

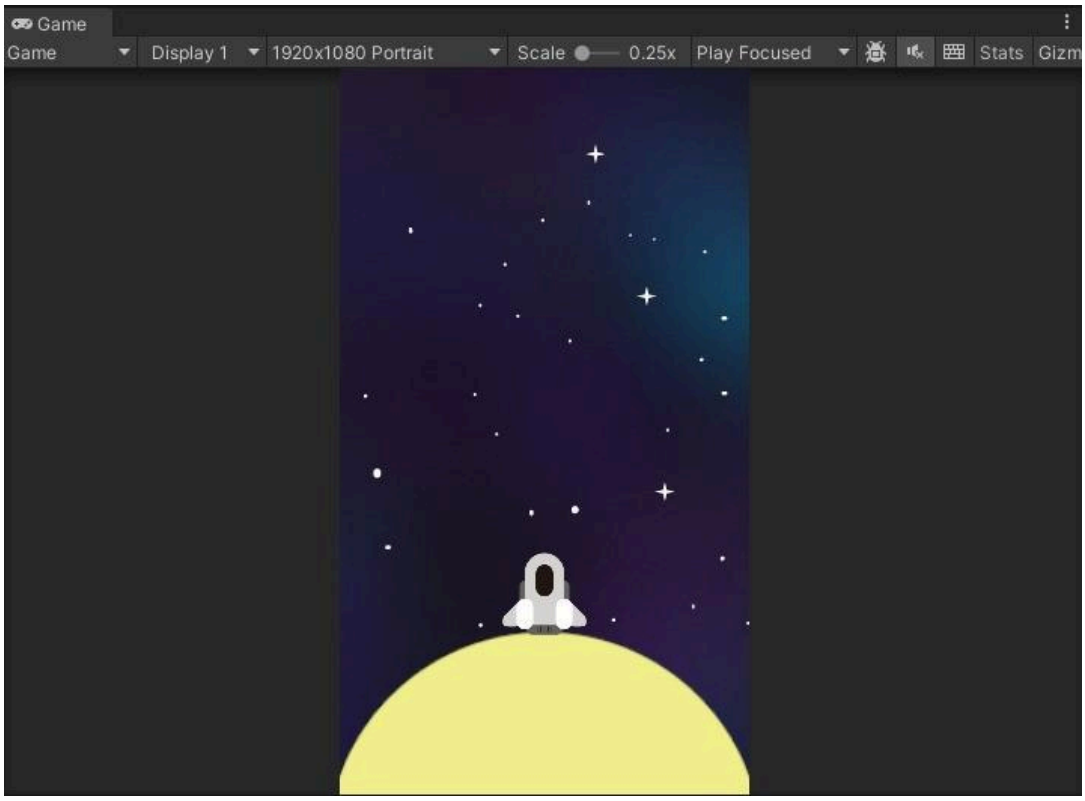


Camera Settings :



Game Orientation :

This is how our game orientation will look like.



GIT Repository: https://github.com/jayesh3801/Rocket-Launch

Logical Breakdown:

Input Handling Breakdown:

- Start Dragging
  - When: Can drag and mouse button is pressed.
  - Action: Capture start position and set dragging to true.
- End Dragging
  - When: Dragging and mouse button is released.
  - Action: Capture end position, launch rocket, reset size, show speed lines, set dragging and canDrag to false, set launched to true.
- Update Dragging
  - When: Dragging.
  - Action: Update drag position and rocket size.
- Tap to Rotate Rocket
  - When: Rocket launched, and mouse button is pressed.
  - Action: Capture tap position, rotate rocket left or right.
- Limit Rocket Speed
  - When: Rocket speed exceeds max.
  - Action: Cap speed to maxVelocity.

```
private void Update()
{
    if (canDrag && Input.GetMouseButtonDown(0))
    {
        dragStartPos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
        isDragging = true;
    }

    if (isDragging && Input.GetMouseButtonUp(0))
    {
        dragEndPos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
        LaunchRocket();
        ResetScale();
        StartCoroutine(ShowSpeedLines());
        isDragging = false;
        canDrag = false;
        isLaunched = true;
    }

    if (isDragging)
    {
        Vector2 currentDragPos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
        UpdateScale(currentDragPos);
    }

    // Handle tap input for rotation after launch
    if (isLaunched && Input.GetMouseButtonDown(0))
    {
        Vector3 tapPosition = Camera.main.ScreenToWorldPoint(Input.mousePosition);
        tapPosition.z = 0; // Ensure z position is zero for 2D

        if (tapPosition.x < transform.position.x)
        {
            RotateAndMoveRocket(45f);
        }
        else
        {
            RotateAndMoveRocket(-45f);
        }
    }

    // Cap the rocket's velocity to maxVelocity
    if (rb.velocity.magnitude > maxVelocity)
    {
        rb.velocity = rb.velocity.normalized * maxVelocity;
    }
}
```

Rocket Launch Slingshot Mechanic:

- Calculate Drag Vector
  - Action: Find the difference between drag end position and drag start position.

- Limit Drag Distance
  - When: Drag distance is greater than maxDragDistance.
  - Action: Limit the drag vector to maxDragDistance.

```
private void LaunchRocket()
{
    Vector2 dragVector = dragEndPos - dragStartPos;

    // Limit the drag distance to maxDragDistance
    if (dragVector.magnitude > maxDragDistance)
    {
        dragVector = dragVector.normalized * maxDragDistance;
    }

    // Calculate the launch force based on the drag distance
    float launchForce = dragVector.magnitude * launchForceMultiplier;

    // Apply the force to the rocket's Rigidbody2D
    rb.AddForce(Vector2.up * launchForce, ForceMode2D.Impulse);
}
```

- Calculate Launch Force
  - Action: Multiply drag distance by launchForceMultiplier to get launch force.
- Apply Launch Force
  - Action: Apply the calculated force upwards to the rocket's Rigidbody2D.

Rotation and Movement Mechanic:

- Get Current Rotation
  - Action: Retrieve and adjust the current Z-axis rotation to ensure it's within -180 to 180 degrees.
- Calculate Target Rotation
  - Action: Add the given angle to the current rotation and clamp it between -45 and 45 degrees.
- Check Rotation Change
  - When: Target rotation is the same as the current rotation.
  - Action: Exit method without rotating.

- Rotate and Move Rocket
  - Action: Rotate the rocket to the target angle using DOTween. After rotation, calculate the new forward direction and apply force in that direction. Schedule a rotation back to zero after a delay using Invoke.

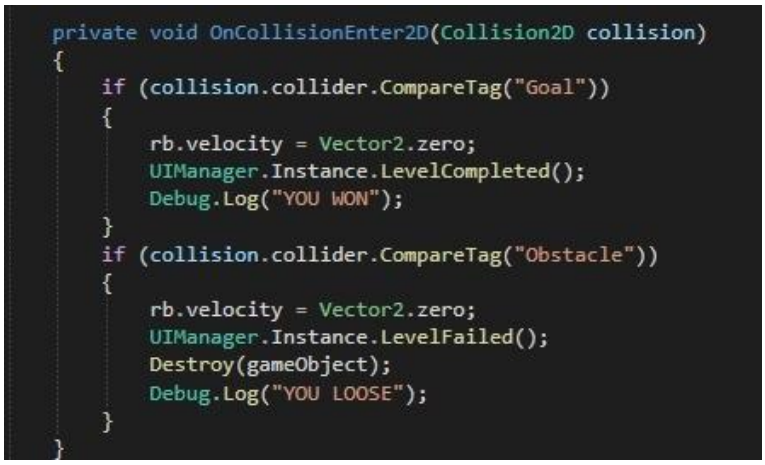
RotateBackToZero Method :

- Store Current Velocity
  - Action: Save the rocket's current velocity.
- Rotate Back to Zero
  - Action: Rotate the rocket back to zero degrees using DO Tween.
  - On Completion: Restore the stored velocity to maintain the rocket's movement direction.

```
private void RotateAndMoveRocket(float angle)
{
    float currentZRotation = transform.eulerAngles.z;
    if (currentZRotation > 180)
    {
        currentZRotation -= 360;
    }

    float targetZRotation = currentZRotation + angle;

    // Clamp the target rotation to -45 and 45 degrees
    targetZRotation = Mathf.Clamp(targetZRotation, -45f, 45f);

    // If target rotation is same as current rotation, return without rotating
    if (Mathf.Approximately(targetZRotation, currentZRotation))
    {
        return;
    }

    // Rotate the rocket
    transform.DORotate(Vector3.forward * targetZRotation, 0.5f, RotateMode.Fast).OnComplete(() =>
    {
        // Calculate the direction to move based on the new rotation
        Vector2 moveDirection = transform.up; // The up vector after rotation is the new forward direction

        // Apply force in the direction of the new rotation
        rb.AddForce(moveDirection * moveForce *2, ForceMode2D.Impulse);

        // Rotate back to 0 after the delay
        Invoke(nameof(RotateBackToZero), returnDelay);
    });
}

private void RotateBackToZero()
{
    // Store the current velocity
    Vector2 currentVelocity = rb.velocity;

    // Rotate back to 0 degrees
    transform.DORotate(Vector3.zero, rotationDuration, RotateMode.Fast).OnComplete(() =>
    {
        // Apply the stored velocity to ensure the rocket continues in the same direction
        rb.velocity = currentVelocity;
    });
}
```

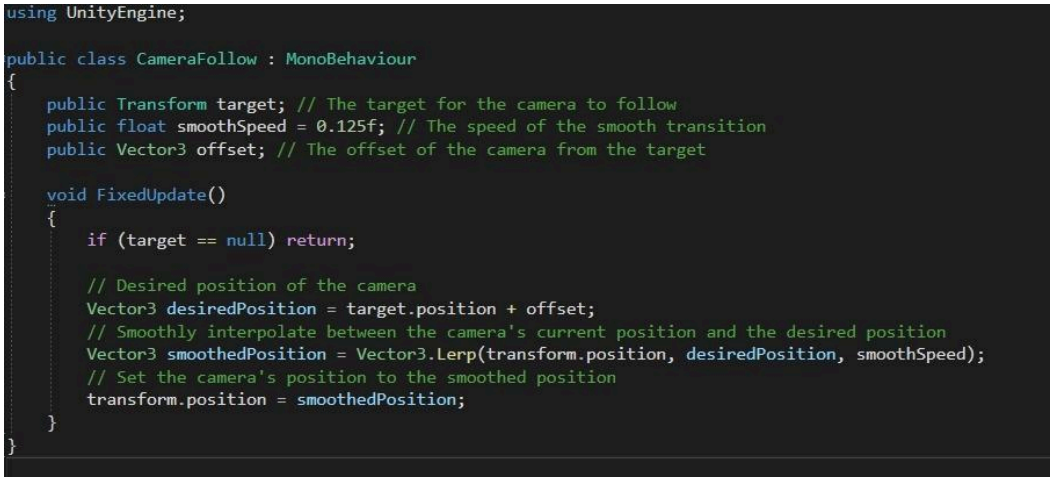
Collision Handling:

- Goal Collision
  - Action: Stop rocket, mark level as completed, log "YOU WON."
- Obstacle Collision
  - Action: Stop rocket, mark level as failed, destroy rocket, log "YOU LOOSE."



```
private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.collider.CompareTag("Goal"))
    {
        rb.velocity = Vector2.zero;
        UIManager.Instance.LevelCompleted();
        Debug.Log("YOU WON");
    }
    if (collision.collider.CompareTag("Obstacle"))
    {
        rb.velocity = Vector2.zero;
        UIManager.Instance.LevelFailed();
        Destroy(gameObject);
        Debug.Log("YOU LOOSE");
    }
}
```

Camera Follow:



```
using UnityEngine;

public class CameraFollow : MonoBehaviour
{
    public Transform target; // The target for the camera to follow
    public float smoothSpeed = 0.125f; // The speed of the smooth transition
    public Vector3 offset; // The offset of the camera from the target

    void FixedUpdate()
    {
        if (target == null) return;

        // Desired position of the camera
        Vector3 desiredPosition = target.position + offset;
        // Smoothly interpolate between the camera's current position and the desired position
        Vector3 smoothedPosition = Vector3.Lerp(transform.position, desiredPosition, smoothSpeed);
        // Set the camera's position to the smoothed position
        transform.position = smoothedPosition;
    }
}
```

- Check Target
  - When: Target is null.
  - Action: Exit method.
- Calculate Desired Position
  - Action: Compute the target position plus the offset.
- Smooth Positioning
  - Action: Interpolate between the current camera position and the desired position using smoothSpeed.
- Update Camera Position
  - Action: Set the camera’s position to a smooth position.

Infinite Background System:

- Assign Camera
  - When: cameraTransform is null.
  - Action: Use the main camera's transform.
- Calculate Background Height
  - Action: Get the height of the background sprite from its SpriteRenderer.

- Reposition Background
  - If Background Moves Out of View (Top):
    - Action: Move background to the top by 2 times its height.
  - If Background Moves Out of View (Bottom):
    - Action: Move background to the bottom by 2 times its height.

```
using UnityEngine;

public class InfiniteBackground : MonoBehaviour
{
    public Transform cameraTransform; // The camera's transform
    private float backgroundHeight; // Height of the background sprite

    private void Start()
    {
        // If no camera is assigned, use the main camera
        if (cameraTransform == null)
        {
            cameraTransform = Camera.main.transform;
        }

        // Calculate the height of the background sprite
        SpriteRenderer spriteRenderer = GetComponent<SpriteRenderer>();
        backgroundHeight = spriteRenderer.bounds.size.y;
    }

    private void Update()
    {
        // Check if the background has moved out of the camera's view on the Y-axis
        if (transform.position.y + backgroundHeight < cameraTransform.position.y)
        {
            // Reposition the background to the top
            Vector3 newPos = transform.position;
            newPos.y += 2 * backgroundHeight;
            transform.position = newPos;
        }
        else if (transform.position.y - backgroundHeight > cameraTransform.position.y)
        {
            // Reposition the background to the bottom
            Vector3 newPos = transform.position;
            newPos.y -= 2 * backgroundHeight;
            transform.position = newPos;
        }
    }
}
```

Obstacle Logic:

- Initialize Rigidbody
  - Action: Get the Rigidbody2D component.
- Add Force in Direction
  - Calculate Force Direction: Determine the force direction based on forceDirection.
  - Apply Force: Multiply the force direction by forceMagnitude and apply it to the Rigidbody2D.

```
public class Obstacle : MonoBehaviour
{
    public enum Direction
    {
        Up,
        Down,
        Left,
        Right,
        UpLeft,
        UpRight,
        DownLeft,
        DownRight
    }

    public Direction forceDirection;
    public float forceMagnitude = 10f;
    private Rigidbody2D rb;

    private void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        AddForceInDirection();
    }

    private void AddForceInDirection()
    {
        Vector2 force = Vector2.zero;

        switch (forceDirection)
        {
            case Direction.Up:
                force = Vector2.up;
                break;
            case Direction.Down:
                force = Vector2.down;
                break;
            case Direction.Left:
                force = Vector2.left;
                break;
            case Direction.Right:
                force = Vector2.right;
                break;
            case Direction.UpLeft:
                force = new Vector2(-1, 1).normalized;
                break;
            case Direction.UpRight:
                force = new Vector2(1, 1).normalized;
                break;
            case Direction.DownLeft:
                force = new Vector2(-1, -1).normalized;
                break;
            case Direction.DownRight:
                force = new Vector2(1, -1).normalized;
                break;
        }

        rb.AddForce(force * forceMagnitude, ForceMode2D.Impulse);
    }
}
```

GameManager:

```
public class GameManager : MonoBehaviour
{
    public static GameManager Instance;

    private void Awake()
    {
        // Check if an instance already exists
        if (Instance != null && Instance != this)
        {
            Destroy(gameObject);
        }
        else
        {
            Instance = this;
            DontDestroyOnLoad(gameObject);
        }
    }

    private void Start()
    {
        LoadSceneByIndex(1);
    }

    public void LoadSceneByIndex(int sceneIndex)
    {
        SceneManager.LoadScene(sceneIndex);
    }
}
```

- Singleton Initialization
  - When: Awake is called.
  - Action: Check if an instance already exists and is not the current one. If another instance exists, destroy the current gameobject. Otherwise, set Instance to the current object and mark it to not be destroyed on scene load.

- Load Initial Scene
    - When: Start is called.
    - Action: Load the scene with index 1.
- Load Scene
    - When: LoadSceneByIndex is called.
    - Action: Load the scene specified by sceneIndex.

UI Manager:

```
using System.Collections;
using TMPro;
using UnityEngine;
using UnityEngine.SceneManagement;

public class UIManager : MonoBehaviour
{
    public static UIManager Instance;

    public TextMeshProUGUI headingText;
    public GameObject nextLevelbutton;
    public GameObject retryLevelbutton;

    private void Awake()
    {
        Instance = this;
    }

    public void LevelCompleted()
    {
        StartCoroutine(EnableUIwithDelay(0));
    }

    public void LevelFailed()
    {
        StartCoroutine(EnableUIwithDelay(1));
    }

    public void OnNextLevelCliked()
    {
        GameManager.Instance.LoadSceneByIndex(SceneManager.GetActiveScene().buildIndex + 1);
    }

    public void OnRetryLevelCliked()
    {
        GameManager.Instance.LoadSceneByIndex(SceneManager.GetActiveScene().buildIndex);
    }

    private IEnumerator EnableUIwithDelay(int state)
    {
        yield return new WaitForSeconds(2);
        if (state == 0)
        {
            LevelCompletedLogic();
        }
        else if (state == 1)
        {
            LevelFailedLogic();
        }
    }

    private void LevelCompletedLogic()
    {
        headingText.gameObject.SetActive(true);
        headingText.text = "Level Completed!";
        nextLevelbutton.SetActive(true);
    }

    private void LevelFailedLogic()
    {
        headingText.gameObject.SetActive(true);
        headingText.text = "Level Failed";
        retryLevelbutton.SetActive(true);
    }
}
```

- Singleton Initialization
  - When: Awake is called.
  - Action: Set Instance to the current object.
- Level Completed
  - When: LevelCompleted is called.
  - Action: Start coroutine to enable UI with a delay (state = 0)(WIN).
- Level Failed
  - When: LevelFailed is called.
  - Action: Start coroutine to enable UI with a delay (state = 1)(LOOSE).
- Next Level Button Click
  - When: OnNextLevelCliked is called.
  - Action: Load the next scene using the build index.
- Retry Level Button Click
  - When: OnRetryLevelCliked is called.
  - Action: Reload the current scene.
- Enable UI with Delay
  - When: Coroutine EnableUIwithDelay is running.
  - Action: Wait for 2 seconds. Based on state, call LevelCompletedLogic or LevelFailedLogic.
- Level Completed Logic
  - Action: Show "Level Completed!" message and enable the next level button.
- Level Failed Logic
  - Action: Show "Level Failed" message and enable the retry level button.

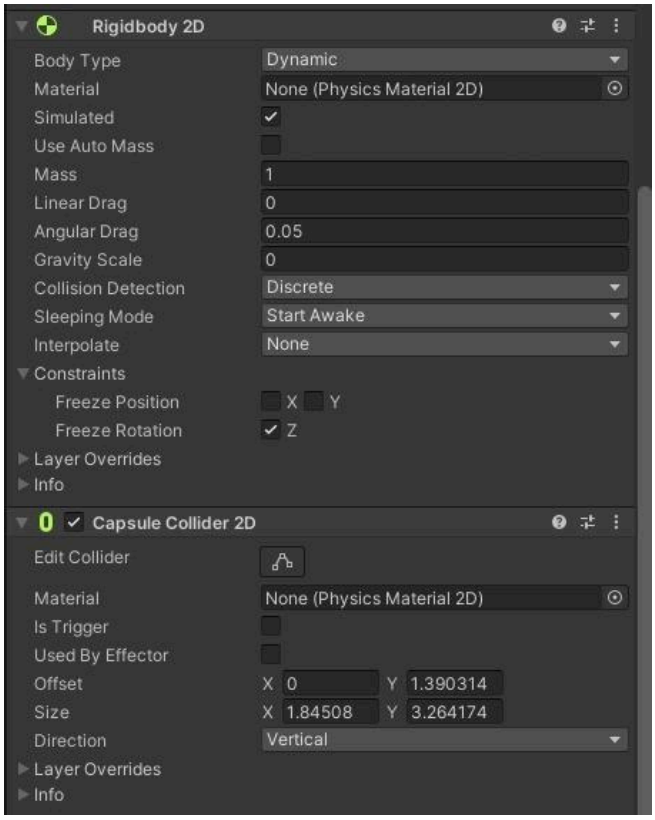
Level Manager:

- Instantiate Goal Object
  - When: Awake is called.
  - Action: Create an instance of goalObject at position (0, goalPos) with no rotation.

```
using UnityEngine;

public class LevelManager : MonoBehaviour
{
    [SerializeField] private GameObject goalObject;
    [SerializeField] private float goalPos;

    private void Awake()
    {
        Instantiate(goalObject,new Vector2(0,goalPos),Quaternion.identity);
    }
}
```

Physics:

- Rocket



- Obstacle


Rigidbody 2D
Body Type
Dynamic
Material
None (Physics Material 2D)
Simulated
Use Auto Mass
Mass
10
Linear Drag
0
Angular Drag
0.05
Gravity Scale
0
Collision Detection
Discrete
Sleeping Mode
Start Awake
Interpolate
None
Constraints
Freeze Position
X
Y
Freeze Rotation
Z
Layer Overrides
Info
Circle Collider 2D
Edit Collider
Material
None (Physics Material 2D)
Is Trigger
Used By Effector
Offset
X 0
Y 0
Radius
0.5
Layer Overrides
Info
Obstacle (Script)
Script
Obstacle
Force Direction
Down Right
Force Magnitude
0.3
Sprites-Default (Material)