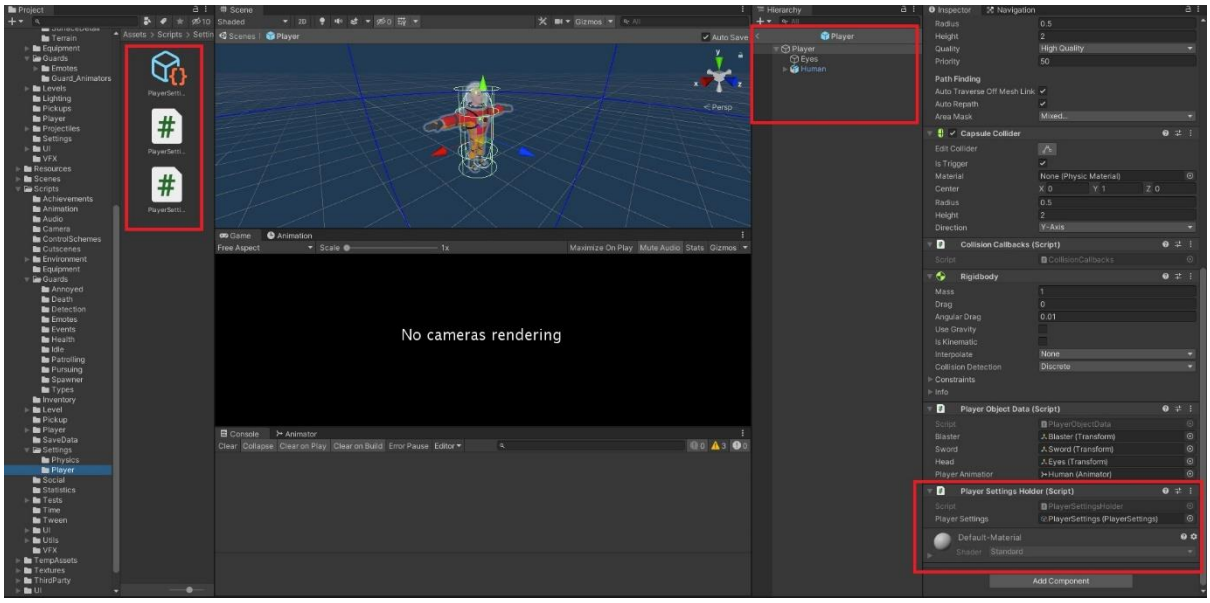
Sprint 02

Assignment 04 : Balancing Gameplay

Step 01: Setup

- We simply do everything using scripts
- First, we create a script called "*PlayerSettingsHolder*" and attach it to the "*Player*" Prefab.
- In that script we add functionality to create a "*ScriptableObject*" by inheriting from it.
- Then we add that ScriptableObject in the Scripts > Settings > Player folder.
- Then we attach this Settings Object into the "*PlayerSettingsHolder*" script attached to the Player.



Step 02: Script & Workflow

- The Scripts workflow is like this:
    - PlayerSettingsHolder > PlayerSettings.
    - LevelController > GunTargetLocator > PlayerSettings.

❖ GunTargetLocator.

- We move the "*shotCheckRaduis*" and the "*ShotAngleRadius*" from this script to the "*PlayerSettings*" script.
- Then we access it by implementing the "*PlayerSettings*" in the "*GunTargetLocator*" constructor.
- And then track it all the down to "*LevelController*" script.

```
GunTargetLocator.cs M ×   LevelController.cs M   PlayerSettings.cs U   PlayerSettingsHolder.cs U   Guards.cs M   Sword.cs M   InventoryMenuController.cs   Playe
Assets > Scripts > Player > Pickups > Gun > GunTargetLocator.cs > GunTargetLocator > GunTargetLocator(GuardManager guardManager, PlayerObjectData playerObjectData, PlayerSettings playerSettings)
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

7 references
public class GunTargetLocator   //S2 - Assignment 02
{
    2 references
    public GuardManager guardManager;
    8 references
    public PlayerObjectData playerObjectData;
    3 references
    private PlayerSettings playerSettings;  //S2 - Assignment 04
    //private float shotCheckRadius = 15f;
    //private float shotAngleRadius = 60f;

    1 reference
    public GunTargetLocator(GuardManager guardManager, PlayerObjectData playerObjectData, PlayerSettings playerSettings)   //S2 - Assignment 04
    {
        this.guardManager = guardManager;
        this.playerObjectData = playerObjectData;
        this.playerSettings = playerSettings;
    }
```

❖ LevelController.

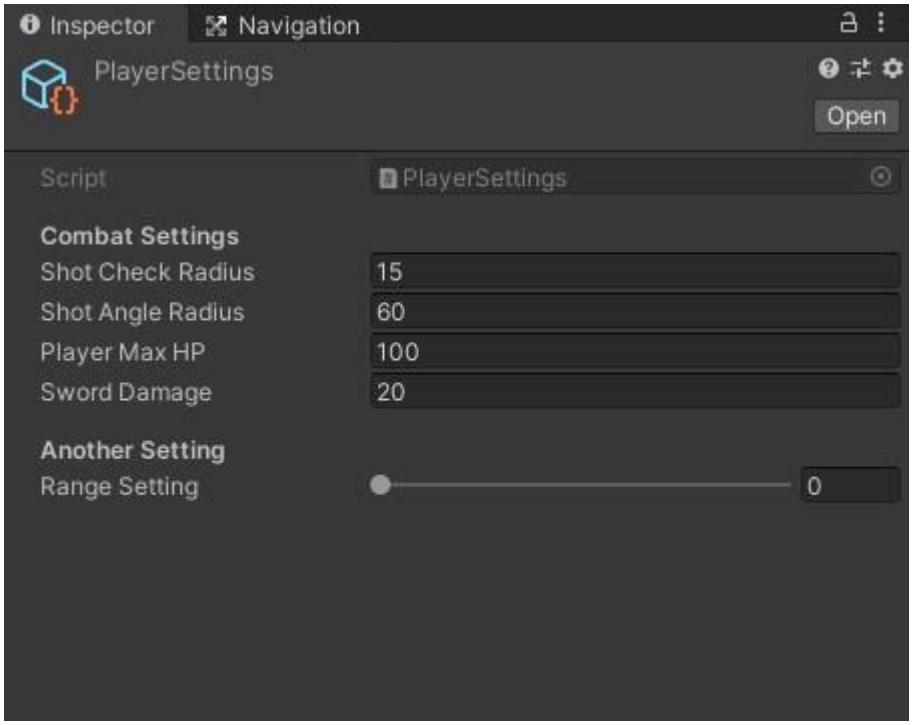
- We first instantiate the "*PlayerSettings*" and get it from the "*PlayerObjectData*" class.
- Then we pass it into the "*CreatePlayer*" function and finally pass it to "*GunTargetLocator*".

```
PlayerSettings playerSettings = playerObj.GetComponent<PlayerSettingsHolder>().playerSettings;  //S2 - Assignment 04

//Part II - Added projectilePool
player = CreatePlayer(playerObj, projectilePool, guardManager, playerSettings);     //Assignment - 04 Part I & S2 - Assignment 04
```

```
GunTargetLocator.cs M   LevelController.cs M ×   PlayerSettings.cs U   PlayerSettingsHolder.cs U   Guards.cs M   Sword.cs M   InventoryMenuController.cs   PlayerEquipmentController.cs
Assets > Scripts > Level > LevelController.cs > LevelController > CreatePlayer(GameObject playerObject, ProjectilePool projectilePool, GuardManager guardManager, PlayerSettings playerSettings)
139     {
140         GameObject playerObject = GameObject.Instantiate(playerObjectPrefab, transform);
141         playerObject.name = "Player";
142
143         return playerObject;
144     }
145
        1 reference
146     private Player CreatePlayer(GameObject playerObject, ProjectilePool projectilePool, GuardManager guardManager, PlayerSettings playerSettings)  //S2 - Assignment 04
147     {
148         PlayerEvents playerEvents = new PlayerEvents(); //S2 - Assignment 02
149
150         NavMeshAgent navMeshAgent = playerObject.GetComponent<NavMeshAgent>();
151
152         PlayerObjectData objectData = playerObject.GetComponent<PlayerObjectData>();
153
154         //Bcoz "Human" is One of the Child Object to "Player".
155         Animator animator = playerObject.transform.Find("Human").GetComponent<Animator>();  //S2 - Assignment 01
156
157         PlayerCollision collision = new PlayerCollision(playerObject.transform);
158
159         PlayerInteractionController interaction = new PlayerInteractionController(transform, playerObject.transform, collision, objectData);
160
161         PlayerInputBroadcaster broadcaster = new PlayerInputBroadcaster();
162
163
164         GunTargetLocator gunTargetLocator = new GunTargetLocator(guardManager, objectData, playerSettings); //S2 - Assignment 02 & 04
165
166         inventory = new InventoryController();  //Assignment 04 - Part I
167
168         //We aslo get ProjectilePool & ProjectileFactory here.
169         PlayerEquipmentController equipmentController = new PlayerEquipmentController(playerObject.transform, inventory, objectData, projectilePool, broadcaster, gunTargetLocator,
```

❖ PlayerSettings.

```
PlayerSettings.cs U ×
H: > Unity > Projects > 1_1_Project_Files (GauravNikalje) > Assets > Scripts > Settings > Player > PlayerSettings.cs
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   [CreateAssetMenu(fileName = "PlayerSettings", menuName = "Player/Settings", order = 1)]
6   public class PlayerSettings : ScriptableObject
7   {
8       [Header("Combat Settings")]
9       public float shotCheckRadius = 15f;     //Default: 15
10      public float shotAngleRadius = 60f;     //Default: 60
11      public float playerMaxHP = 100f;
12
13       public float swordDamage = 10f;
14
15      [Header("Another Setting")]
16      [Range(0f, 15f)]
17      public float rangeSetting;
18
19  }
```



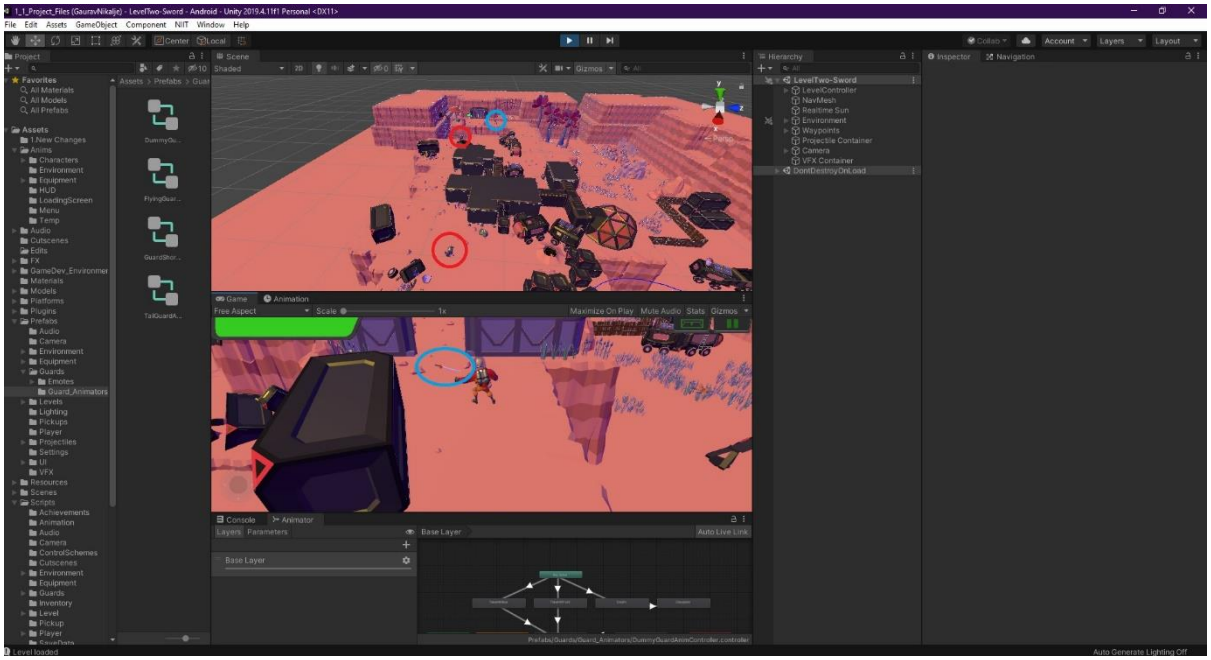
❖ **PlayerSettingsHolder.**

PlayerSettings.cs U | PlayerSettingsHolder.cs U | GunTargetLocator.cs M | LevelController.cs M

Assets > Scripts > Settings > Player > PlayerSettingsHolder.cs > PlayerSettingsHolder > OnDrawGizmos()

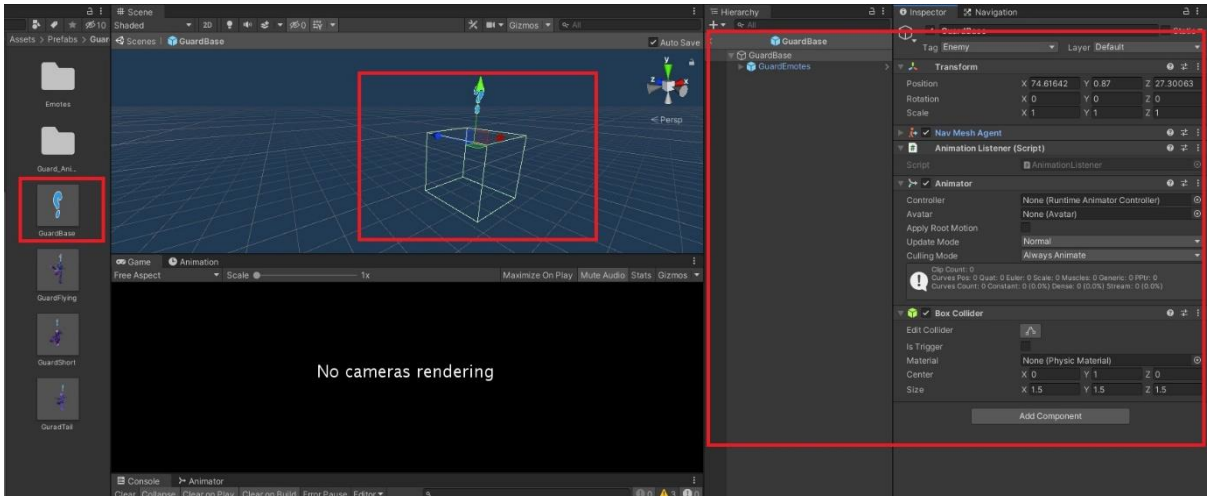
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

1 reference
public class PlayerSettingsHolder : MonoBehaviour
{
    2 references
    public PlayerSettings playerSettings;

    /// <summary>
    /// Callback to draw gizmos that are pickable and always drawn.
    /// </summary>
    0 references
    private void OnDrawGizmos()
    {
        Gizmos.color = Color.blue;
        Gizmos.DrawWireSphere(transform.position, playerSettings.shotCheckRadius);
    }
}
```

❖ Add some additional guards and adjust guard pathing



- Added some "*Guards*" and a "*Sword*" Pickup GameObject in "*LevelTwo*".
- Red – Guards & Blue – Pickup GameObject.

❖ GuardBase Prefab & Script.

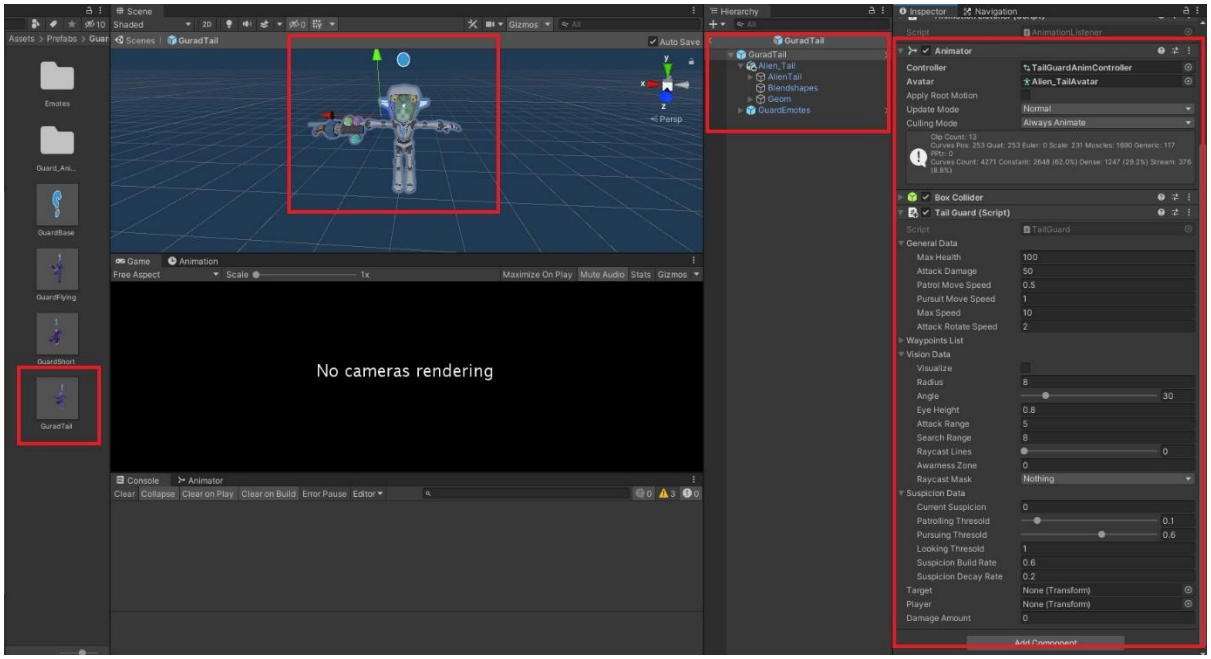


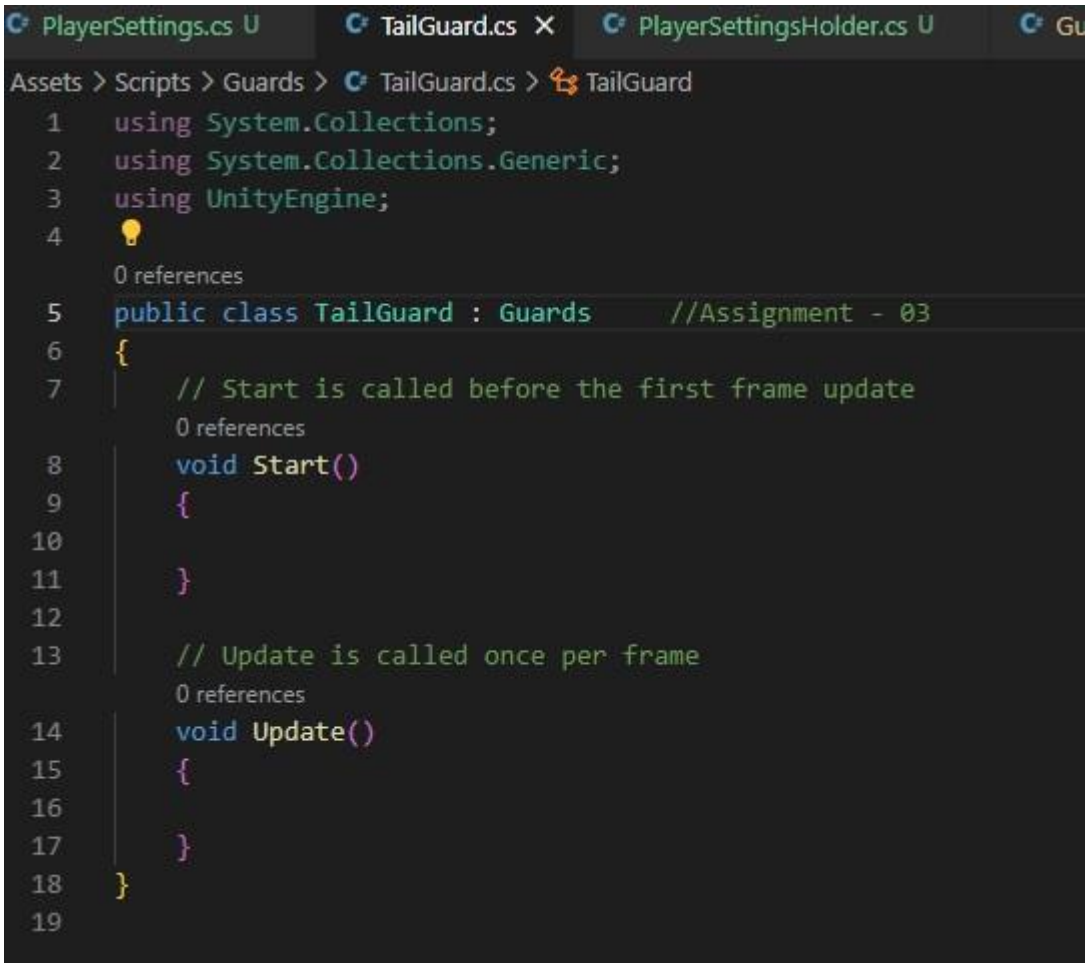
Guards.cs M × | TailGuard.cs | PlayerSettings.cs U | PlayerSettingsHolder.cs U

Assets > Scripts > Guards > Guards.cs > Guards

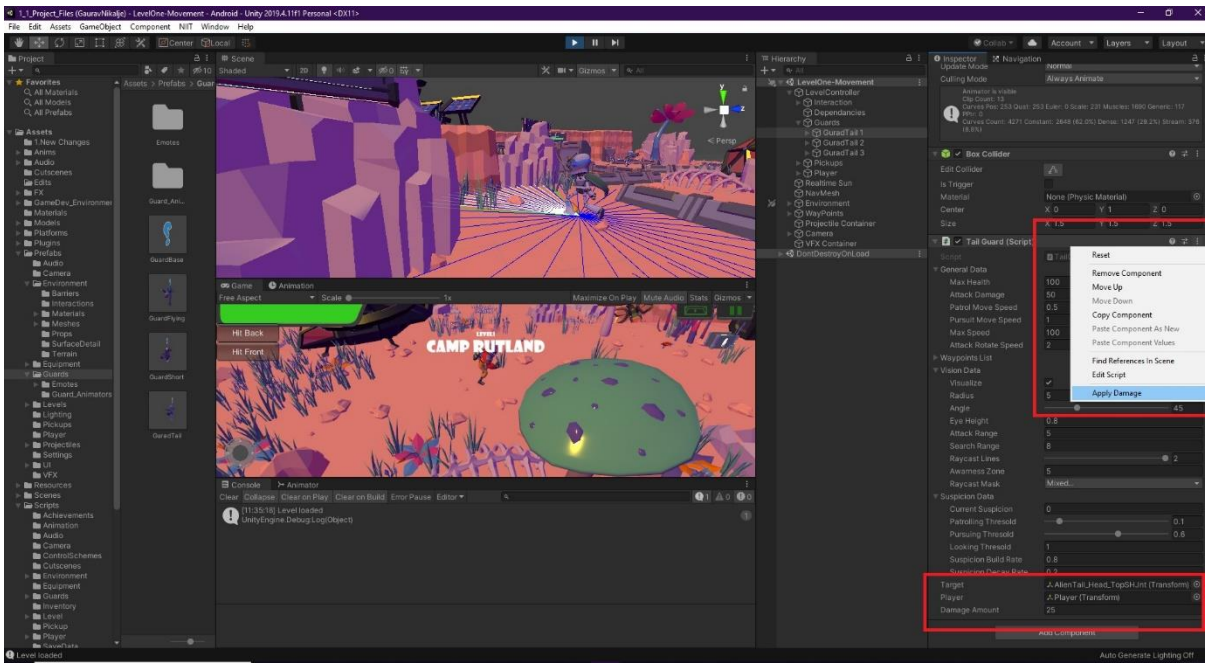
```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

48 references
public class Guards : MonoBehaviour     //Assignment - 03
{
    7 references
    public GeneralData generalData;
    2 references
    public WayPoints waypointsList;
    4 references
    public VisionData visionData;
    1 reference
    public SuspicionData suspicionData;
    0 references
    public Transform target;

    3 references
    public Action<float, Transform> OnDamageTaken = delegate { };

    4 references
    public void TakeDamage(float damageTaken, Transform instigator)
    {
        OnDamageTaken.Invoke(damageTaken, instigator);
    }

```

❖ GuardTail Prefab – from GuardBase.





❖ **Debug Function – Guard Taking 25 Damage.**



Step 03: What I Learnt

- **Explain a mechanic teaching method of introducing and reinforcing mechanics gradually.**

  - In Video Games there are various Game or Gameplay Mechanics, which are considered as the architecture of the Game.
  - These mechanics help to function the Game as per required.

- For that we use various Programming or Game Design Patterns, such as Observer Pattern, Command, Singleton and more.
- For instance, lets just consider we want to implement the Observer Pattern.
- So, we would have to create One Main Class which would hold all the main functionalities of the game.
- Say we have created a Player Collision Class.
- So, we set up all the Player Collision Functionalities in this script and add a Collision Event in it.
- Then we create various other scripts and subscribe to the function or method of the Collision Scripts.
- So, like that we implement various mechanics in a Game, so as to make it easy to understand, modify and use.
- We have also implemented “Command Pattern” in the Guard AI assignment.
- So, in short, we should at least develop a basic structure of Game Mechanics before we start with development process.

- **How can the use of scriptable objects improve your development process?**

  - Scriptable Objects are just like the Public Fields that we implement in a script, where we can modify it in the scene view.

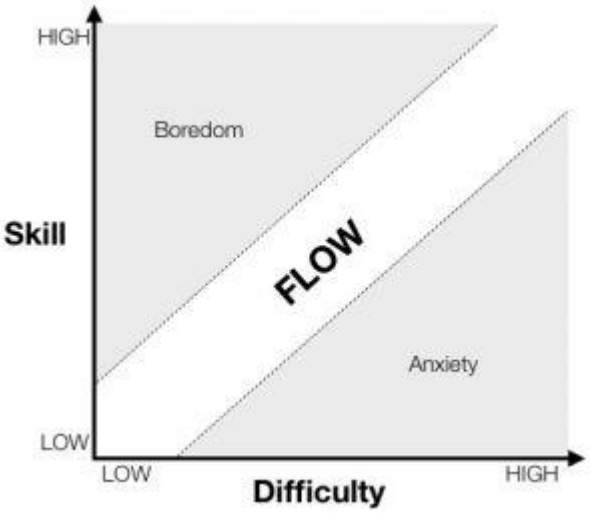
- Also, scriptable objects are useful in creating multiple instances of a same asset.
- For example, we could create the GuardBase Prefab as a Scriptable Object, so that we can create multiple instances of Guards, with different functionalities.
- Another Example, while creating a Card Game, we can create the Basic Card Info Class as a Scriptable Object.
- Then we can create multiple cards with this scriptable and create a complete set of 52 Cards, by just creating a simple Basic Card Class as Scriptable Object.
- And hence, scriptable objects help to save time and efforts during the development process.

- **Explain the fundamentals of flow theory. Include info on:**
  - **4 Key concepts**
  - **Difficulty curve**

- In positive psychology, a flow state, also known colloquially as being in the zone, is the mental state in which a person performing some activity is fully immersed in a feeling of energized focus, full

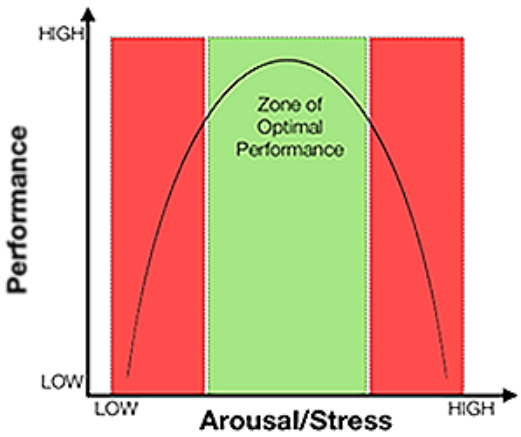
involvement, and enjoyment in the process of the activity.

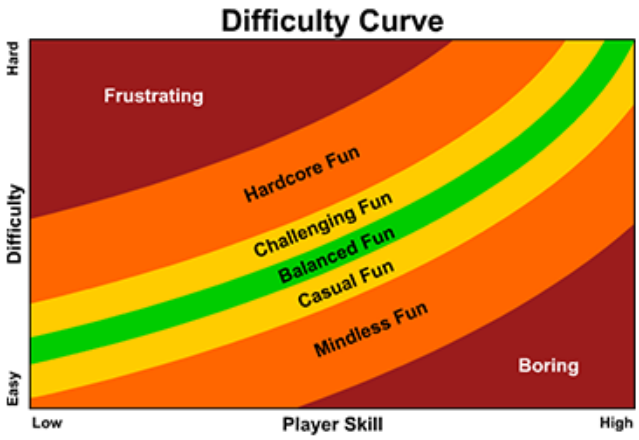
- In Short it simply means Completely being focused on the work you are performing, and forgetting about the world.
- The 4 Key Concepts are,
  - Skill
  - Difficulty
  - Boredom
  - Anxiety



- Our focus tends to fluctuate between these 4 Zones.
- The Graph can go Up & Down based upon the Mood, Mentality, Like & Dislike of a Player.
- While practicing the Flow Zone, we should tend to maintain our Focus in a moderate Graph.

- The Difficulty Curve, is thus defined same as the Above.
- If we make the Level More Difficult to complete, the Player will get Frustrated and Bored.
- And if we make it too Easy, then also the Player will get Bored at some point, due to lack of Obstacles or Events.
- So, we have to design a Level or a Game such as a Pro Player/Gamer as well as a Newbie Gamer can enjoy the Game and Complete it.
- For such instances many Games have the implementation of the "*Game Difficulty*" at the beginning of the Game.





----------THE END----------