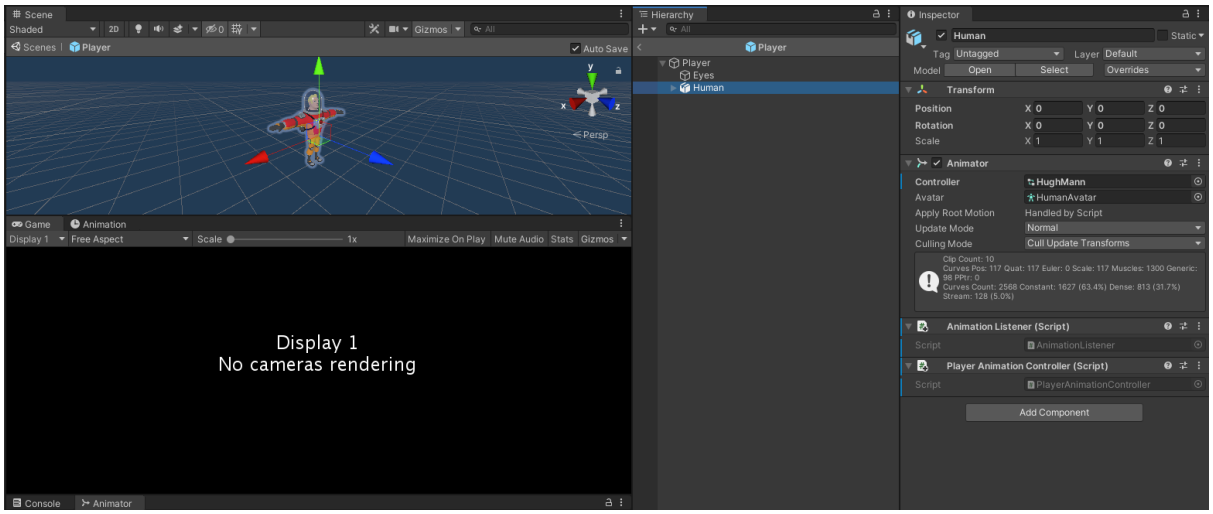
Sprint 02

Assignment 01 : POV

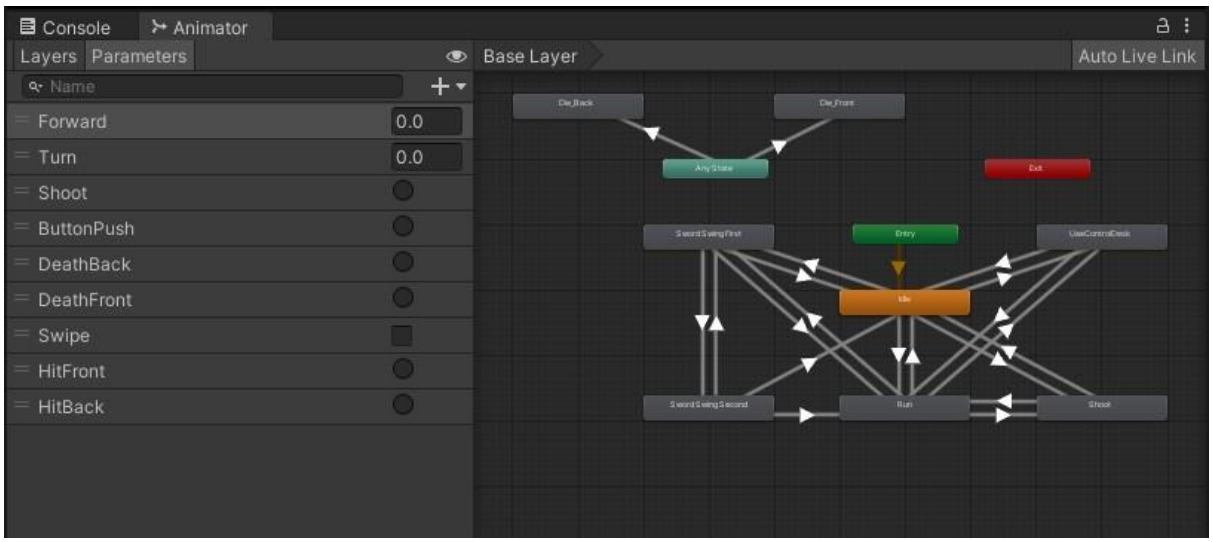
Part II

Step 01: Setup

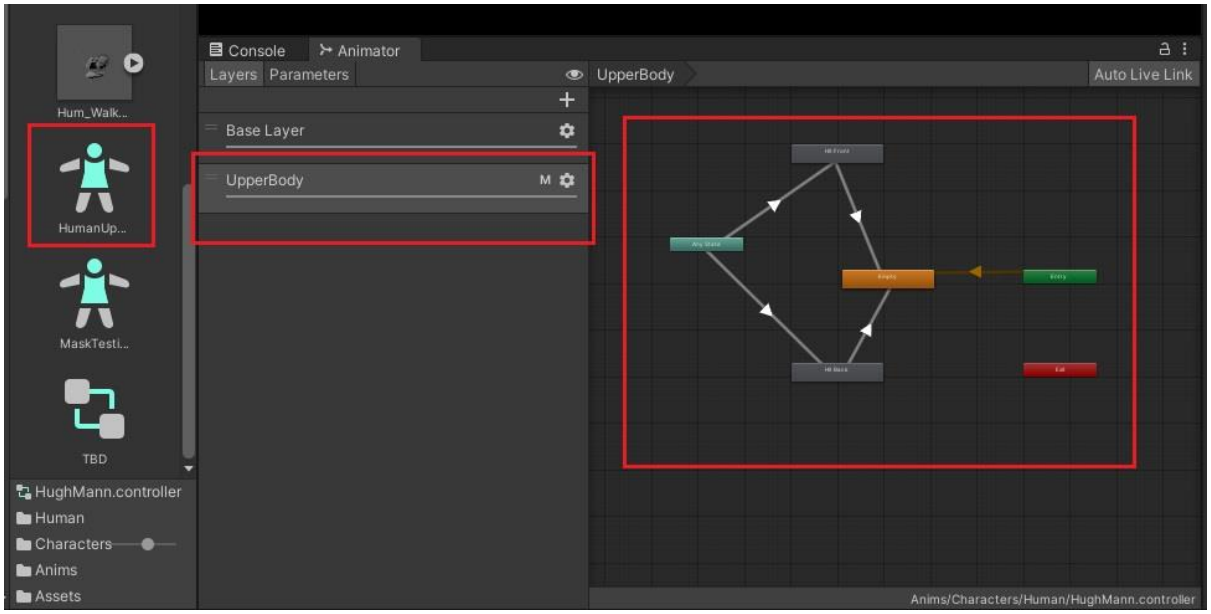
- We attach the "*PlayerCollisions*" and the "*PlayerAnimationController*" scripts to the "Human" GameObject under the "*Player*" Prefab.



- Then, we create the Animator of the Player with different States and Conditions.



- Then, we also create another Layer on the same Animator to Mask the UpperBodyPortion of the Player and attach in the Mask Area of the "*UpperBody*" Layer.



Step 02: Script & Workflow

- The Scripts workflow is like this:
  - LevelController > GunTargetLocator & PlayerController.
  - Gun > GunTargetLocator.
  - Sword.
  - Guards > DebugGuardDamager.

❖ **GunTargetLocator.**

- This script performs the action of locating Enemy Targets and Shooting Projectiles towards them.

- It uses a Radius attached to the Player Weapon, and if the Enemy is within the range of the radius, the Gun GameObject automatically detects the Enemy and Fires a Projectiles towards that Enemy.
- We have created this script because it is hard to shoot at enemy from a Third Person View.
- The "*Locate*()" function is the main function which performs all the Action.
- It takes an Action Delegate called "*OnSuccess*" which passes the Closest Guard towards the Player.
- The function first loops for each Guard present within the Range of the Radius, and then tries to find the Guard that has the Smallest Distance from the Player; i.e., the Closest Guard.
- Then once the Guard is found, it passes that Guard into the "*OnSuccess*" delegate which is called or accessed in other scripts.
- In our case in the "*Gun*" script.

DebugGuardDamager.cs M | GunTargetLocator.cs | Gun.cs M

Assets > Scripts > Player > Pickups > Gun > GunTargetLocator.cs > GunTargetLocator > Locate(Action<Transform> OnSuccess)

```
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;
    1 reference
    private float shotCheckRadius = 15f;
    1 reference
    private float shotAngleRadius = 60f;

    1 reference
    public GunTargetLocator(GuardManager guardManager, PlayerObjectData playerObjectData)
    {
        this.guardManager = guardManager;
        this.playerObjectData = playerObjectData;
    }

    1 reference
    public void Locate(Action<Transform> OnSuccess)
    {
        //It locates the Nearest Enemy among all the other Enemies.
        Transform closestEnemy = null;

        //We run a Check for all/each the Guard(s) present inside the Player's Aim Vision.
        //Note: While using KeyValuePair, you should use Dictionary variables and not any other than that.
        foreach(KeyValuePair<Guards, GuardController> guard in guardManager.guardDict)
        {
            if(guard.Value.CanBeTargeted == false)
            {
                continue;
            }

            //Position of the Guard.
            Vector3 guardPosition = new Vector3(guard.Key.transform.position.x, guard.Key.visionData.eyeHeight, guard.Key.transform.position.z);

            //Calculate the Distance from the Guard's Head or Eye Position to the Player's Head Position.
            Vector3 playerToEnemy = guardPosition - playerObjectData.Head.position;

            //Ray from Player's Head towards Guard's Head.
            Ray rayToEnemy = new Ray(playerObjectData.Head.position, playerToEnemy);
            Debug.DrawRay(playerObjectData.Head.position, playerToEnemy, Color.yellow);

```

```
43              //Hit: Stores & Returns the info. of the Objects that it has Hit.
44              RaycastHit hit;
45              if(Physics.Raycast(rayToEnemy, out hit, shotCheckRadius))
46              {
47                  //We check if the Hit has Hit a Target with Tag "Enemy", then go into the below Function.
48                  if(hit.transform.tag == "Enemy")
49                  {
50                      Vector3 distance = guard.Key.transform.position - playerObjectData.Head.position;
51                      float angle = Vector3.Angle(distance, playerObjectData.Head.forward);
52                      Debug.DrawRay(distance, playerObjectData.Head.position);
53
54                      //Absolute (Abs) = Returns Positive Integer. Even if the Integer Value is -ve, its Absolute Value will be in +ve
55                      //Like if Angle is "-75", its Abs will be "75".
56                      if(Mathf.Abs(angle) <= shotAngleRadius)
57                      {
58                          //This is the First Enemy that our Player Sees or Detects.
59                          //And then it assigns that Guard's transfrom to the "closestEmeny" transform varialbe.
60                          //And from the 2nd Guard on, the code will directly go into the "esle" part & execute its logic.
61                          if(closestEnemy == null)
62                          {
63                              closestEnemy = guard.Key.transform;
64                          }
65
```

```
65
66                          //Here we check over each Guard in the Player's Vision Range (shotAngleRadius) and calculate its ClosestDistance.
67                          //We Iterate this Process for Each Guard.
68                          //Then we compare Each Guard's Distance, and see if it is Smaller than the Previous Guard.
69                          //If it is, then we assign that Guard as "closestEnemy" and pass in that value to other scripts.
70                          else
71                          {
72                              //We make the Positions of the Player & the Guard FLAT.
73                              //It helps in calculating the Distance. As only the X & Z components are to be calulated. The Y component is Zero.
74                              Vector3 playerPositionFloor = playerObjectData.Head.position;
75                              playerPositionFloor.y = 0;
76
77                              Vector3 guardPositionFloor = guard.Key.transform.position;
78                              guardPositionFloor.y = 0;
79
80                              //We calculate the Distance of each Guard found after the First Guard. (i.e., Guard2, Guard3, so on...)
81                              float checkingAllGuardDistance = Vector3.Distance(playerPositionFloor, guardPositionFloor);
82
83
84                              //Then we compare whether the Distance of Each Next or New Gaurd found, is Smaller than the Distance of the First Guard. (i.e., Guard1.)
85                              //That is, the one from the "if(closestEnemy == null)" condition.
86                              //And if it satisfies as True, then we stores that Guard, (Guard2, Guard3, so on...Whichever has Smallest Distance from Player) as Closest Enemy.
87                              //If not, then it considers the above guard from Line 61, the First Guard as the ClosestGuard.
88                              //This Process Iterates every time for each Guard, as it is inside a "Foreach-Loop".
89                              //And once we find the Guard that has the Smallest Distance from the Player, we pass that Guard as the "Closest Guard" and run a check along with the
90                              //Previous Closest Guards. Then we iterate with each Guard and try to find the Guard that has the Smallest Distance from Player.
91                              //And once we find that, we then pass it along other scripts on Line 106.
92                              Vector3 closestGuardPositionFloor = closestEnemy.position;
93                              closestGuardPositionFloor.y = 0;
94                              float currentClosestGuardDistance = Vector3.Distance(playerPositionFloor, closestGuardPositionFloor);
95
96                              //This checks whether Current Guard's Distance is Smaller than the Previous Guard's Distance.
97                              if(checkingAllGuardDistance < currentClosestGuardDistance)
98                              {
99                                  closestEnemy = guard.Key.transform;
100                             }
101
102                         }
103                     }
104                 }
105             }
106         }
107
108         OnSuccess?.Invoke(closestEnemy);
109         //We could have used a Return Function type, where we would return this Transform value,
110         //But it does not provide More Functionalities.
111
112     }
113 }
```

❖ Gun.

- This script is responsible for the management of ammos & its use.
- We access the "*Locate*()" function form the "*GunTargetLocator*" script and pass in the parameters & logic.
- But before that we create 3 classes within this script, namely "*Target*" the main Abstract Class, and "*VectorTarger*" & "*TransformTarget*" as its override classes.
- **VectorTarger**: is used to make the Player towards the other Direction of the Enemy, if the Player sees no Enemy within the Range or when the Guard is Dead and there a No Guards left within the Range, i.e., when it returns Null for Guards.
- **Transform Target**: is used to make the Player Look towards the Guard and perform the "*Shoot*" Projectile animations.

DebugGuardDamager.cs M | Gun.cs M | GunTargetLocator.cs M

Assets > Scripts > Player > Pickups > Gun > Gun.cs > Gun > FindTarget()

```
248
249
250     //S2 - Assignment 02
        5 references
251     private abstract class Target
252     {
            2 references
253         public abstract Vector3 position { get;}
254     }
255
        1 reference
256     private class VectorTarget : Target
257     {
            2 references
258         private Vector3 value;
259
            1 reference
260         public VectorTarget(Vector3 value)
261         {
262             this.value = value;
263         }
264
            2 references
265         public override Vector3 position { get { return value; } } //=> value; //Short FOrm for: "{ get{ return value; }}".
266
267     }
268
        1 reference
269     private class TransformTarget : Target
270     {
            2 references
271         private Transform value;
272
            1 reference
273         public TransformTarget(Transform value)
274         {
275             this.value = value;
276         }
            2 references
277         public override Vector3 position => value.position; //Short FOrm for: "{ get{ return value.position; }}".
278     }
279 }
280
```

- Then, we create some functionalities to execute the VectorTarget & TransformTarget classes in the "*StartUse*()" function.
- But before that, we shift the code that's in there into another Function called "*FireBullets*".
- Then we create a function called "*OnAnimationEvent*" with a String parameter, and pass in a Switch statement called "Fire" and "*AttackEnd*" string parameters.
- These are the Animation Events on the Animation Clip of the Projectile Shoot animation on the Gun GameObject.
- Then we Subscribe to this function into the "*Equip*()" function by using the "*OnAnimationEvent*" function from the "*AnimationListener*" script, that we attach to the "*Player*" Prefab and access it.
- Similar way, we UnSubscibe to the Function in the "*UnEquip*()" method.

```
2 references
private void OnAnimationEvent(string param)
{
    switch(param)
    {
        case "Fire":
            //AttackStart();
            FireBullet();
            break;

        case "AttackEnd":
            AttackEnd();
            break;
    }
}
```

```
2 references
public void Equip(IEquipable currentMainHandEquipment, Transform equipmentHolder)
{
    animationListener.OnAnimationEvent += OnAnimationEvent;
    _gameObject.SetActive(true);

}

2 references
public void UnEquip(IEquipable currentMainHandEquipment)
{
    animationListener.OnAnimationEvent -= OnAnimationEvent;
    _gameObject.SetActive(false);

}
```

```
3 references
private AnimationListener animationListener;

//S2 - Assignment 02
3 references
private PlayerInputBroadcaster inputBroadcaster;
2 references
private GunTargetLocator gunTargetLocator;
2 references
private PlayerEvents playerEvents;
5 references
private Transform transform;
2 references
private Target currentTarget;
2 references
private Collider[] colliders;   //Extra

1 reference
public Gun(GameObject gameObject, PlayerObjectData playerObjectData, GunTargetLocator gunTargetLocator, PlayerEvents playerEvents,  //S2 - Assignment 02
           ProjectilePool projectilePool, PlayerInputBroadcaster inputBroadcaster)
{
    this._gameObject = gameObject;
    this.playerObjectData = playerObjectData;
    this.projectilePool = projectilePool;

    //S2 - Assignment 02
    this.inputBroadcaster = inputBroadcaster;
    animator = playerObjectData.PlayerAnimatior;
    this.gunTargetLocator = gunTargetLocator;
    this.transform = gameObject.transform;
    this.playerEvents = playerEvents;

    bulletStart = playerObjectData.Blaster.Find("BulletStart");

    colliders = gameObject.GetComponentsInChildren<Collider>(); //Extra

    animationListener = animator.gameObject.GetComponent<AnimationListener>();
}
```

- These Then we create a Bool called "*AttackActive*" in the script.
- Then we create two following functions namely, "*AttackStart*()" & "*AttackEnd*()" and turn "*True*" & "*False*" the "AttackActive" Bool and the "*PlayerInputBroadcaster*" states.
- Then finally in the "*CanUse*()" function, we pass the "*AttackActive*" as false.

```
1 reference
private void AttackStart()
{
    attackActive = true;
    //FireBullet();
    inputBroadcaster.EnableActions(ControlType.None);   //Bcoz Player should Not Move while playing the Shoot Animation.
    //Debug.Log("Player inputs = None!");
}

1 reference
private void AttackEnd()
{
    attackActive = false;
    inputBroadcaster.EnableActions(ControlType.Gameplay);   //Player can Move after the Shoot Animation has ended.
    //Debug.Log("Player inputs = GamePlay!");
}
```

```
1 reference
public bool CanUse()
{
    //Debug.Log("Shoot!");
    //Have we left enough time left between Shots?
    //Do we have ammo?
    //return (ammo != null && coolDown <= 0f);    // "="(Equals to) Sign is VERY IMP //

    return ammo != null && attackActive == false;   //S2 - Assignment 02
}
```

- Then we return to the "*StartUse*()" function, where we pass in the "AttackStart" and the "*FindTarget*()" functions.
- Also, we pass the Shoot Trigger in the "*OnShotTargetSet*" Action Delegate from the "*PlayerEvents*" class.
- In the "*TargetFind*()" target function, we pass an If-Else Condition where we Return the "*TransformTarget*" if the Enemy or Guard is Not Null, and do the opposite of that if the Enemy is Null by passing the "*VectorTarget*" where we add Player's Position with its Forward.

```
1 reference
public void StartUse()
{
    //S2 - Assignment 02
    AttackStart();

    Target target = FindTarget();

    playerEvents.OnShotTargetSet.Invoke(target.position, () =>
    {
        animator.SetTrigger("Shoot");
    });

    currentTarget = target;
}

1 reference
private Target FindTarget() //S2 - Assignment 01 - Part II
{
    Transform closestEnemy = null;
    gunTargetLocator.Locate( (closest) => closestEnemy = closest );
    //Explanation of the Above Line.
    /*private LocateAction(Transform closest)
    {
        Transform closestEnemy = null;
        closestEnemy = closest;
    }*/

    if(closestEnemy != null)
    {
        return new TransformTarget(closestEnemy);
    }
    else
    {
        //IMP: We cannot use "transform.position" as this is not a MonoBehaviour Script.
        //So we assign this GameObject's Transform to the Transform.
        //See Lines 38 & 55.
        return new VectorTarget(transform.position + transform.forward);    //Bcoz we want to make our Player Look at its Forward Direction.
    }
}
```

❖ Sword.

- Firstly, we discard all the references & functionalities of the "*SwordAttackAnimation*" script.
- This function is similar to "*Gun*" script, but the only difference is this script plays the "*SwordSwing*" animations.
- We implement the same "*OnAnimationEvent*" function in this script, and pass in the String Parameter as "*SwipeOneStart*", "*SwipeTwoStart*" and "*SwipeOneEnd*", "*SwipeTwoEnd*" animation states.
- Then we Subscribe and UnSubscribe to this function in the "*Equip*()" and the "*UnEquip*()" functions respectively.
- Then we create a Boolean called "*ChainAttack*".
- We set this Boolean as "*False*" in the "*CanUse*()" function.
- Then we create "*AttackStart*" and "*AttackEnd*" function where we pass in the SwordSwing logic as well as the "*SwordTrail*" visual effect and set the collisions to True & False respectively.
- Then finally, in the "*StartUse*()" function we set the "*Swipe*" bool as True.

DebugGuardDamager.cs M | Sword.cs | Gun.cs M | PlayerEvents.cs | GunTargetLocator.cs M

Assets > Scripts > Player > Pickups > Sword > Sword.cs > Sword > AttackStart()

```
4 references
private GameObject _gameObject; //Sword GameoBject attached from "EquipmentFactory"
2 references
private PlayerObjectData playerObjectData;
6 references
private CollisionCallbacks collisionCallbacks;
//private SwordAttackAnimation attackAnimation;
6 references
private Animator animator;  //S2 - Assignment 02
3 references
private AnimationListener animationListener;    //S2 - Assignment 02
3 references
private TrailRenderer swordTrail;   //S2 - Assignment 02
4 references
private bool chainAttack;

1 reference
public Sword(GameObject gameObject, PlayerObjectData playerObjectData)
{
    this._gameObject = gameObject;
    this.playerObjectData = playerObjectData;
    animator = playerObjectData.PlayerAnimatior; //.GetComponent<Animator>();   //S2 - Assignment 02
    animationListener = animator.gameObject.GetComponent<AnimationListener>();  //S2 - Assignment 02
    swordTrail = gameObject.GetComponentInChildren<TrailRenderer>();    //S2 - Assignment 02

    //We add <>(true) becoz it checks hidden GameObjects too. If we do not write the (true),
    //it will not look through the hidden GameObjects
    collisionCallbacks = gameObject.GetComponentInChildren<CollisionCallbacks>(true);
    //collisionCallbacks.gameObject.SetActive(false);   //DO NOT ENABLE - Turns the Sword Collison Off in the Scene.
}

1 reference
public void StartUse()  //S2 - Assignment 02
{
    //If the "Swipe" is True, then go inside & Return.
    if(animator.GetBool("Swipe"))   //Default: True. Also can be written as: animator.GetBool("Swipe") == true.
    {
        chainAttack = true;
        return;
    }

    //If not, then set it True here in the below line.
    animator.SetBool("Swipe", true);

    //attackAnimation = new SwordAttackAnimation(_gameObject.transform);
    //attackAnimation.OnAttackStarted += () => EnableHitBox();    //Turn On Collison
    //attackAnimation.OnAttackStarted += () => DisableHitBox();   //Turn Off Collison
    //attackAnimation.Start();
}

2 references
public void Equip(IEquipable currentMainHandEquipment, Transform equipmentHolder)
{
    _gameObject.SetActive(true);
    collisionCallbacks.OnTriggerEntered += SwrodCollision;
    animationListener.OnAnimationEvent += OnAnimationEvent; //S2 - Assignment 02
}

2 references
public void UnEquip(IEquipable currentMainHandEquipment)
{
    _gameObject.SetActive(false);
    collisionCallbacks.OnTriggerEntered -= SwrodCollision;
    animationListener.OnAnimationEvent -= OnAnimationEvent; //S2 - Assignment 02
}
```

```
2 references
private void OnAnimationEvent(string param) //S2 - Assignment 02
{
    //The String "param" checks every String on an ANimation Clip or State.
    //And if the String matches any of the Below Names, it executes the respective Functions.
    switch(param)
    {
        case "SwipeOneStart":
        case "SwipeTwoStart":
            AttackStart();
            break;

        case "SwipeOneEnd":
        case "SwipeTwoEnd":
            AttackEnd();
            break;
    }
}
```

```
1 reference
public void AttackStart()   //S2 - Assignment 02
{
    //Debug.Log("OnAttack - from Sword");
    swordTrail.emitting = true; //Start Trail.
    collisionCallbacks.gameObject.SetActive(true);
    OnUsed();
    //EnableHitBox();
}

2 references
private void AttackEnd()    //S2 - Assignment 02
{
    swordTrail.emitting = false;    //End Trail.
    collisionCallbacks.gameObject.SetActive(false);

    //If the ChainAttack is True, that means we have 1 Animation Pending, and after finishing that we set it to False.
    if(chainAttack) //Default: True.
    {
        chainAttack = false;
    }
    //If the ChainAttack is False, that means we do not have any Animations Pending, so we set the Swipe also to False.
    else
    {
        animator.SetBool("Swipe", false);
    }

}
```

❖ LevelController.

- We just simply pass in and instantiate the "*GunTargetLocator*" script within the "*CreatePlayer*" function.

```
DebugGuardDamager.cs M   LevelController.cs M ×   Sword.cs   EquipmentFactory.cs M   Gun.cs M   GunTargetLocator.cs M
Assets > Scripts > Level > LevelController.cs > LevelController > CreatePlayer(GameObject playerObject, ProjectilePool projectilePool, GuardManager guardManager)

1 reference
private Player CreatePlayer(GameObject playerObject, ProjectilePool projectilePool, GuardManager guardManager)
{
    PlayerEvents playerEvents = new PlayerEvents(); //S2 - Assignment 02

    NavMeshAgent navMeshAgent = playerObject.GetComponent<NavMeshAgent>();

    PlayerObjectData objectData = playerObject.GetComponent<PlayerObjectData>();

    //Bcoz "Human" is One of the Child Object to "Player".
    Animator animator = playerObject.transform.Find("Human").GetComponent<Animator>();  //S2 - Assignment 01

    PlayerCollision collision = new PlayerCollision(playerObject.transform);

    PlayerInteractionController interaction = new PlayerInteractionController(transform, playerObject.transform, collision, objectData);

    PlayerInputBroadcaster broadcaster = new PlayerInputBroadcaster();


    GunTargetLocator gunTargetLocator = new GunTargetLocator(guardManager, objectData); //S2 - Assignment 02

    inventory = new InventoryController();  //Assignment 04 - Part I

    //We aslo get ProjectilePool & ProjectileFactory here.
    PlayerEquipmentController equipmentController = new PlayerEquipmentController(inventory, objectData, projectilePool, projectileFactory, broadcaster, gunTargetLocator, playerEvents);

    equipmentController.OnItemConsumed += (item) => OnItemConsumed(item);

    PlayerController controller = new PlayerController(playerObject.transform, navMeshAgent,
        interaction, collision, broadcaster, animator, playerEvents, inventory, equipmentController, projectilePool);   //Assignment 04 - Part I & S2 - Assugnment 02

    broadcaster.Callbacks.OnPlayerStartUseFired += () => controller.StartUse();

    return new Player(controller, broadcaster,
        objectData, interaction, equipmentController);  //Assignment 04 - Part II
}
```

❖ PlayerController.

- We just simply pass in a function called "*TakeHit*" in the "*OnDamageTaken*" delegate.
- This function plays the "*HitFront*" animation when the Player's Forward or the Player is Facing towards the Enemy/Guard and plays the "*HitBack*" if the Player's Forward is facing to the opposite side of the Guard.

```
DebugGuardDamager.cs M   PlayerController.cs M ×   LevelController.cs M   Sword.cs   Equipment
Assets > Scripts > Player > PlayerController.cs > PlayerController

        this.collision = collision;
        this.inputBroadcaster = inputBroadcaster;
        this.animator = animator;
        this.equipmentController = equipmentController;     //Assignment 04 - Part I
        this.inventory = inventory; //Assignment 04 - Part I
        this.projectilePool = projectilePool;   //Assignment 04 - Part II

        playerHealth = new PlayerHealth(playerMaxHP);

        playerHealth.OnDamageTaken += (currentHealth) =>
        {
            TakeHit(lastDamageLocation);    //S2 - Assignment 02
            OnPlayerDamageTaken(currentHealth);
        };
```

```
1 reference
public void TakeHit(Vector3 location)   //S2 - Assignment 02
{
    if(IsPlayerFacing(location))
    {
        animator.SetTrigger("HitFront");
    }
    else
    {
        animator.SetTrigger("HitBack");
    }
}
```

❖ **DebugGuardDamager.**

- I do not have the Player taking Hit from Guard functionality, so I created 2 Buttons in the Game which make the Player take Hit from Front & Back.
- For this I simply created 2 rects in the "*DebugGuardDamager*" script and added them lines of code by accessing the Player's "*Animator*" and the "*PlayerHealth*" delegate function.
- I created and accessed all the required files in the "*Start*()" functions and implemented them respectively.

DebugGuardDamager.cs M | LevelController.cs M | Sword.cs | EquipmentFactory.cs M | Gun.cs M | GunTargetLocator.cs M

Assets > Scripts > Guards > Health > DebugGuardDamager.cs > ...

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

0 references
public class DebugGuardDamager : MonoBehaviour
{
    7 references
    Guards[] guards;
    2 references
    bool damageButton;

    //S2 - Assignment 02
    1 reference
    public Transform levelObj;
    2 references
    Animator anim;
    0 references
    GameObject obj;
    2 references
    LevelController levelController;

    0 references
    private void Start()    //S2 - Assignment 02
    {
        levelObj = GetComponent<Transform>();
        //obj = GameObject.Find("Player").GetComponent<GameObject>();
        levelController = GameObject.Find("LevelController").GetComponent<LevelController>();
        anim = GameObject.Find("Player").GetComponentInChildren<Animator>();    //IMP: Use "GameObject" whenever not working with other components.
        //anim = player.GetComponent<Animator>();
    }

    0 references
    private void OnGUI()
    {
        if(guards == null)
        {
            return;
        }

        if(GUI.Button(new Rect(0, 30, 100, 30), "Hit Back"))   //Button1 - Back    //S2 - Assignment 02
            //anim?.SetTrigger("HitFront");
            levelController.player.Controller.playerHealth.OnDamageTaken(50f);
            //Debug.Log(levelObj);

        if(GUI.Button(new Rect(0, 60, 100, 30), "Hit Front"))   //Button2 - Front    //S2 - Assignment 02
            //levelController.player.Controller.playerHealth.OnDamageTaken(50f);
            anim?.SetTrigger("HitFront");

        for(int i = 0; i < guards.Length; i++)
        {
            if(guards[i] == null)
            {
                continue;   //Increments 'i' value. Does not execute below code
            }

            float yPos = Screen.height - (30 * (i+1));
            Rect rect = new Rect(0, yPos, 200, 30);

            damageButton = GUI.Button(rect, "Damage" + guards[i].gameObject.name);

            if(damageButton)
            {
                guards[i].TakeDamage(guards[i].generalData.attackDamage, transform);    //transform = instigator
            }

        }
    }
}
```

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