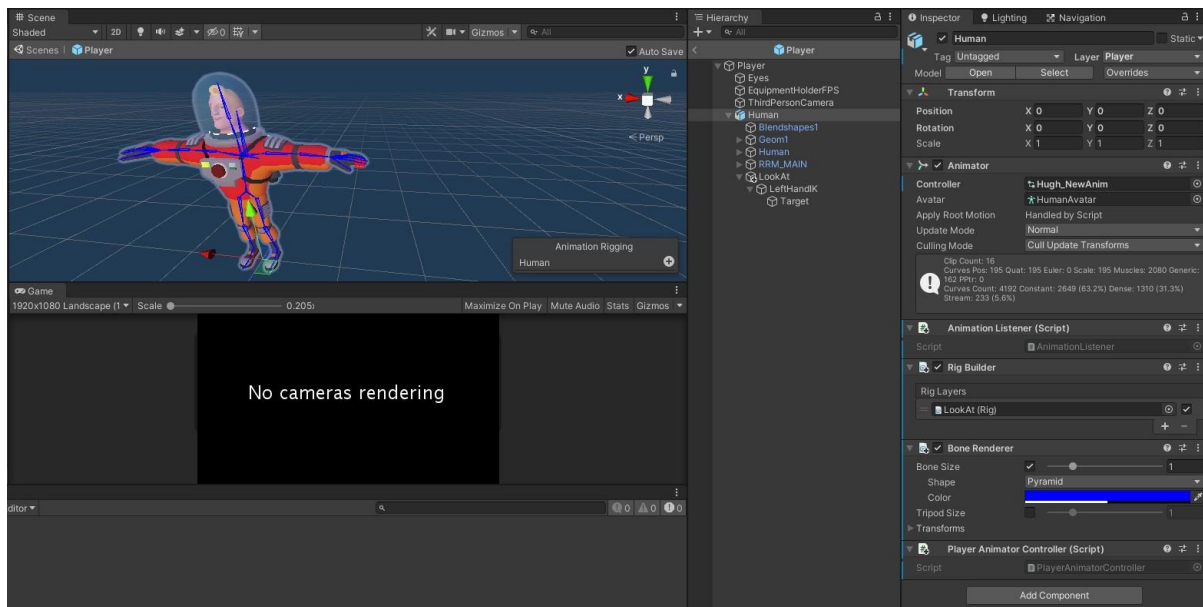


Sprint 03

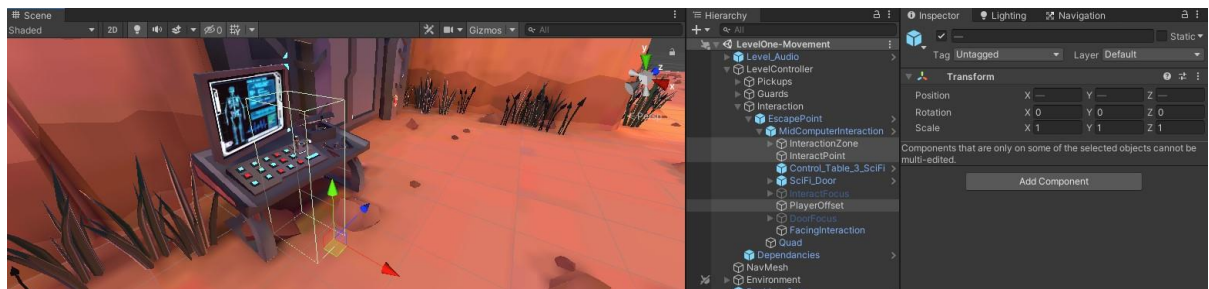
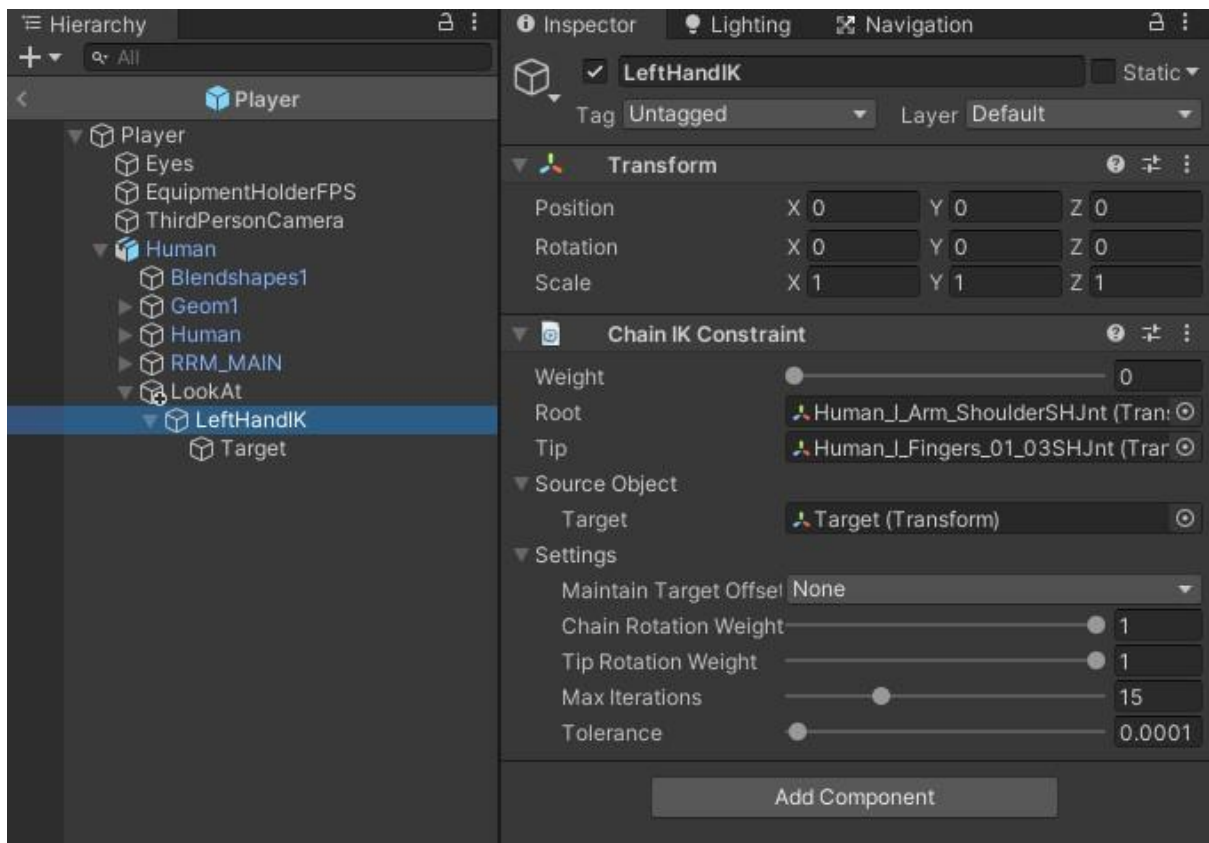
Assignment 05 : IK & Animation Rigging

Step 01: Setup

- First, we import the “*Animation Rigging*” package from the “*Package Manager*” or grab it from the “*Asset Store*”.
- Then we open the Player Prefab and add the “*Bone Render Setup*” from the “*Animation Rigging*” tab.



- This tool displays the Bone Rig in the scene view and grabs all the bones attached to the Player or Object.
- Then next we create “*Rig Setup*” from the “*Animation Rigging*” tab and rename it “*LookAt*”.
- And create an Empty GameObject and attach to it “*Chain IK Constraint*” and add the Root and Tip bones respectively.
- Then we create another child empty gameobject called “*Target*” and attach it the “*Ball Effector*” object and place it to Player’s Left-Hand Finger.
- Then we attach this Target object to the “*Target*” source object of the “*Chain IK Constraint*”.

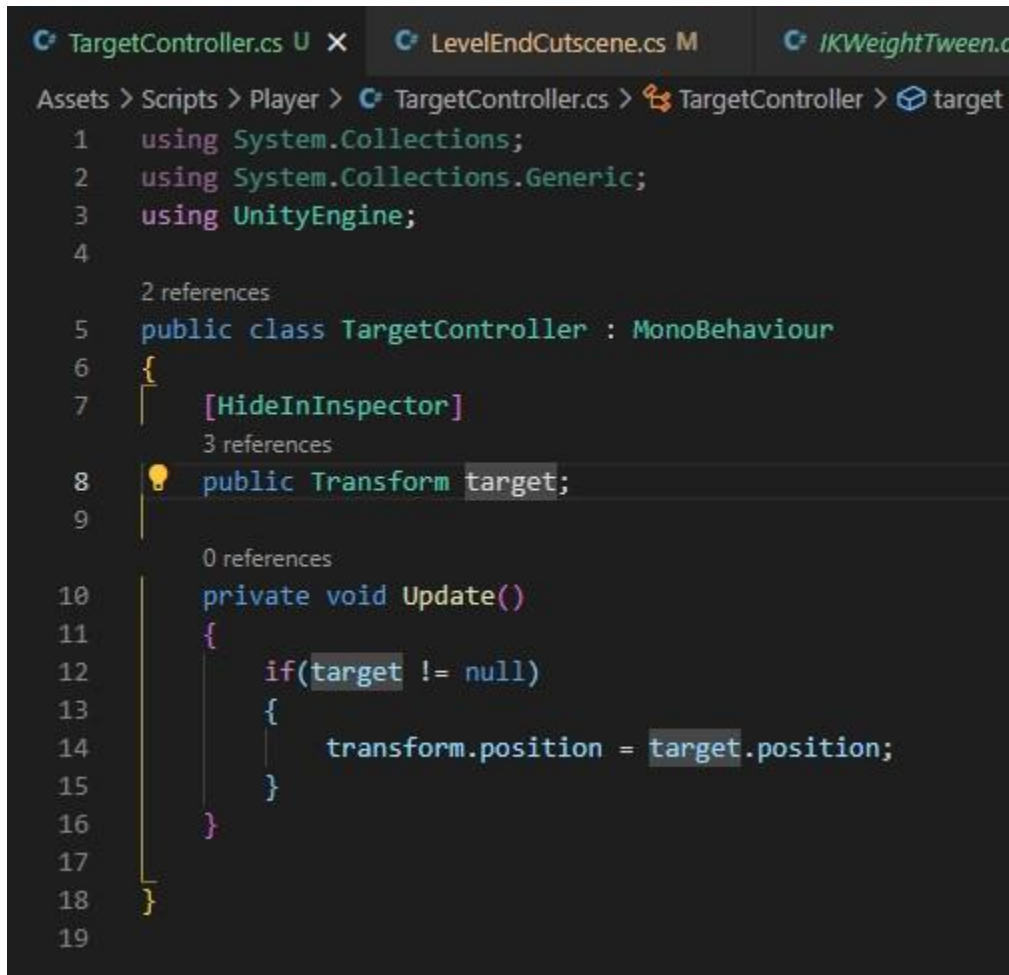


Step 02: Script & Workflow

- The Scripts workflow is like this:
 - LevelController > LevelEndCutscene > TargetController & IKWeightTween.

❖ TargetController.

- This script simply passes the Target, i.e., the one attached to Player's Finger and to the "Chain IK Constraint".
- This script is attached to the "Target" GameObject of the Player.



The screenshot shows the Unity Inspector window with the 'TargetController' script selected. The script is attached to the 'target' GameObject. The code is as follows:

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class TargetController : MonoBehaviour
6 {
7     [HideInInspector]
8     public Transform target;
9
10    private void Update()
11    {
12        if(target != null)
13        {
14            transform.position = target.position;
15        }
16    }
17 }
18
19
```

❖ IKWeightTween.

- This script is actually responsible to make the IK work.
- This script simply increases the "IK Weight" value as the Player reaches the "Offset" area, and thus we get the effect of Player's Hand reaching towards the Button, in real-time.

```
Assets > Scripts > Tween > IKWeightTween.cs > IKWeightTween > IKWeightTween(ChainIKConstraint IKChain, EndLevelInteraction endLevel)

1  using System.Collections;
2  using System;
3  using System.Collections.Generic;
4  using UnityEngine;
5  using UnityEngine.Animations.Rigging;
6
7  2 references
8  public class IKWeightTween
9  {
10     6 references
11     private ChainIKConstraint IKChain;
12     2 references
13     private EndLevelInteraction endLevel;
14     2 references
15     public Action OnIKComplete;
16     5 references
17     private float distFromButton;
18
19     1 reference
20     public IKWeightTween(ChainIKConstraint IKChain, EndLevelInteraction endLevel)
21     {
22         this.IKChain = IKChain;
23         this.endLevel = endLevel;
24     }
25
26     0 references
27     private void Start()
28     {
29         if(IKChain.weight == 1.0f)
30         {
31             IKChain.weight = 0.0f;
32         }
33     }
34
35     1 reference
36     public void Update()
37     {
38         distFromButton = Vector3.Distance(endLevel.interactionTarget.position, IKChain.data.target.position);
39         distFromButton = Mathf.Clamp01(distFromButton);
40
41         IKChain.weight = distFromButton;
42         OnIKComplete();
43         Debug.Log("IKWeight: " + IKChain.weight);
44         Debug.Log("DistanceFromButton: " + distFromButton);
45
46         // if(distFromButton == 0f)
47         // {
48         //     IKChain.weight = 1.0f;
49         //     OnIKComplete();
50         //     //Debug.Log("This is called!");
51         // }
```

```
28     public void Update()
29     {
30         distFromButton = Vector3.Distance(endLevel.interactionTarget.position, IKChain.data.target.position);
31         distFromButton = Mathf.Clamp01(distFromButton);
32
33         IKChain.weight = distFromButton;
34         OnIKComplete();
35         Debug.Log("IKWeight: " + IKChain.weight);
36         Debug.Log("DistanceFromButton: " + distFromButton);
37
38         // if(distFromButton == 0f)
39         // {
40         //     IKChain.weight = 1.0f;
41         //     OnIKComplete();
42         //     //Debug.Log("This is called!");
43         // }
44     }
45 }
46
47 }
48
```

❖ LevelEndCutscene.

- This script is responsible to Play the LevelEnd Cutscene Timeline Animation, as well as perform the IK Rig Setup.
- The “*PlayComplete()*” function is the one which defines the parameters to Play the Timeline and perform IK Chain Weight animation.
- It simply runs some checks and if the conditions are met, then increases the IK Chain Weight accordingly to the position or distance from the Button Point.
- It first checks if the Player is within the Offset range.
- If it is, then checks if the Player is facing towards the “*FacingInteraction*” point.
- And then finally assigns the Target to the Target attached to the Player’s Hand.
- Then the “*IKWeightTween*” script is called and executes its functionality.
- Then ends the Tween Animation by calling the “*OnComplete*” delegate function.

```
LevelEndCutscene.cs M X
Assets > Scripts > Level > LevelEndCutscene.cs > LevelEndCutscene
1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using UnityEngine;
5
6 public class LevelEndCutscene
7 {
8     private Player player;
9     private GuardManager guardManager;
10    private EndLevelInteraction endLevelInteraction;
11    private IKWeightTween ikWeightTween;
12
13    public LevelEndCutscene(Player player, GuardManager guardManager) //, EndLevelInteraction endLevelInteraction)
14    {
15        this.player = player;
16        this.guardManager = guardManager;
17        //this.endLevelInteraction = endLevelInteraction;
18    }
19 }
```



```

1 reference
20 public void PlayComplete(EndLevelInteraction endLevel, Action CompleteLevel)
21 {
22     player.Broadcaster.EnableActions(ControlType.None);
23     guardManager.ForceIdle();
24
25     //Step - 01
26     float distFromOffset = Vector3.Distance(player.ObjectData.transform.position, endLevel.playerInteractionOffset.position);
27     //Debug.Log(distFromOffset);
28     if(distFromOffset <= 0.5f)
29     {
30         //Step - 02
31         player.Controller.Face(endLevel.faceTarget.position, () =>
32         {
33             //Step - 03
34             player.ObjectData.setTargetController.target = endLevel.interactionTarget;
35             Debug.Log(player.ObjectData.setTargetController.target);
36
37             //Step - 04: Increase the Weight of the IKChain.
38             ikWeightTween = new IKWeightTween(player.ObjectData.leftHandIK, endLevel);
39             //Debug.Log(ikWeightTween);
40             ikWeightTween.OnIKComplete += () =>
41             {
42                 ikWeightTween = null;
43                 player.Animator.SetTrigger("ButtonPush");
44
45                 CutsceneSignallistener cutsceneListener = endLevel.GetComponentInChildren<CutsceneSignallistener>();
46                 cutsceneListener.OnCutsceneComplete += CompleteLevel;
47             };
48             //endLevel.playableDirector.playableAsset = endLevel.playableAsset;
49             endLevel.playableDirector.Play();
50         });
51     }
52 }
53
54

```

❖ LevelController.

- This script controls every level if the game.
- The “*LevelEndCutscene*” script is instantiated here, and its respective function “*PlayComplete()*” is called in the if interaction statement.

```

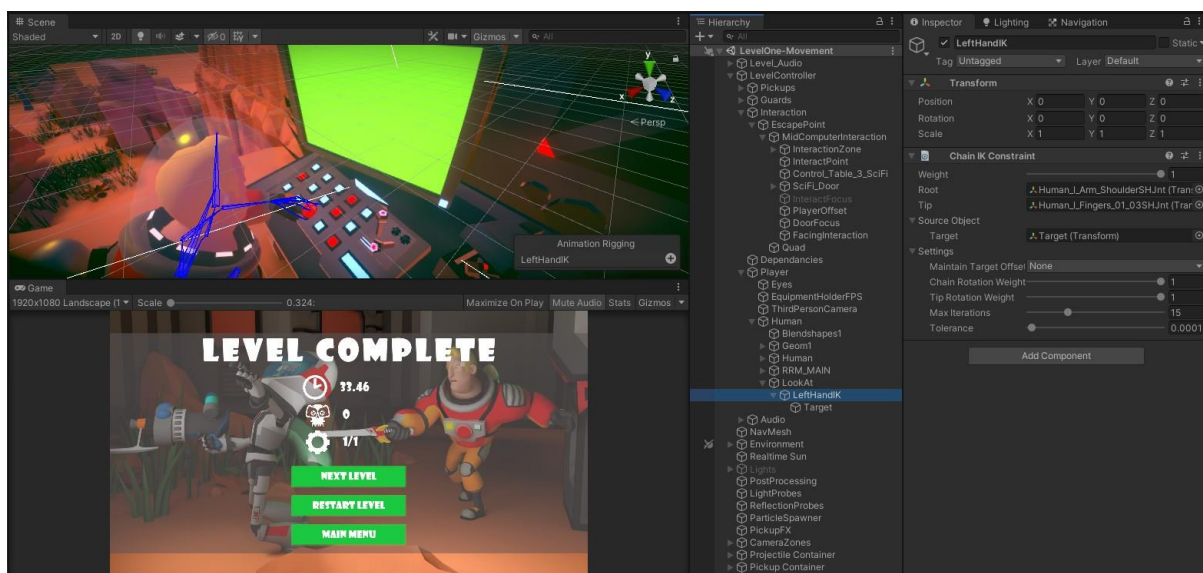
TargetController.cs U X  LevelEndCutscene.cs M  LevelController.cs M X
Assets > Scripts > Level > LevelController.cs > LevelController > Start()
87     levelStatsController.OnEnemyKilled();
88     OnGuardKilled();
89 });
90 guardEvents.AddSpawnedListener((guard, controller) => OnGuardSpawned());
91
92 cameraController = new CameraController(dependencies.cameraContainer,
93     transform.parent, player.Controller.Transform);
94
95 levelEndCutscene = new LevelEndCutscene(player, guardManager); //S3 - Assignment 04
96
97 player.Interaction.OnInteractionStarted += (interaction) =>
98 {
99     if(interaction is EndLevelInteraction endLevel)
100     {
101         //EndLevel(endLevel);
102         //endLevel.playableDirector.Play();
103         levelEndCutscene.PlayComplete(endLevel, CompleteLevel); //S3 - Assignment 04
104         //CompleteLevel();
105     }
106 };
107

```

```
LevelEndCutscene.cs M  LevelController.cs M X
Assets > Scripts > Level > LevelController.cs > LevelController > Update()

151
152     CompleteLevel();
153 }
154
0 references
155 public void Update()
156 {
157     levelStatsController.UpdateTime(Time.deltaTime);
158     guardManager.Update();
159     pickupController.Update();
160     player.Controller.Update(cameraController.MainCameraTransform.forward);
161     uiController.Update();
162     audioManager.Update();
163     player.Interaction.Update();
164     cameraController.Update();
165     projectileManager.Update();
166     levelEndCutscene.Update(); //S3 - Assignment 05
167 }
168
```

❖ Output.



Step 03: What I Learnt

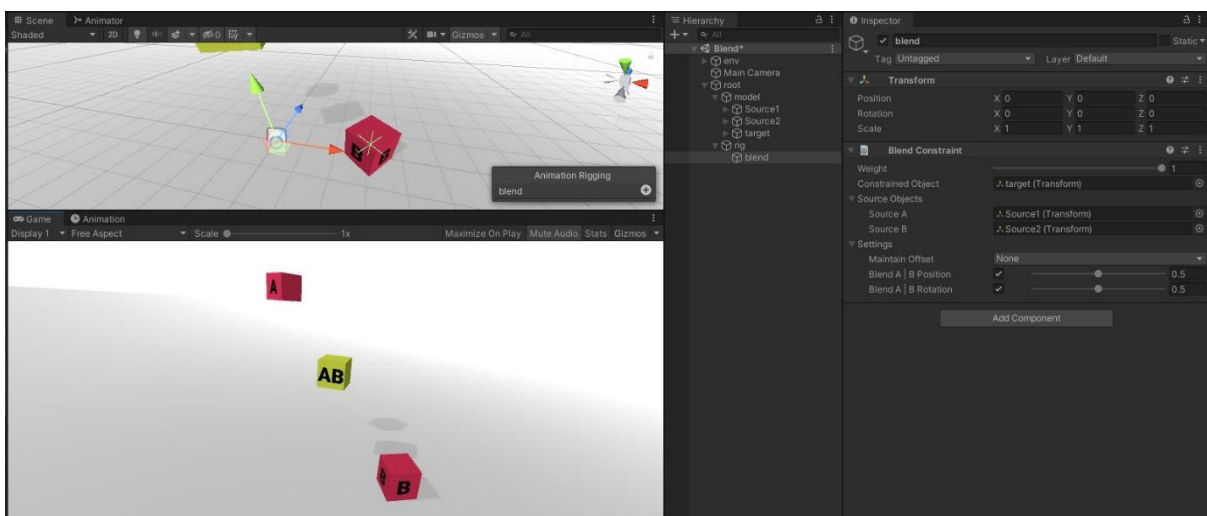
- **Explain the difference between forward kinematics and inverse kinematics.**

- Forward Kinematics uses a single joint to animate- Translate or Rotate, while Inverse Kinematics uses two or more joints to determine the Translate and Rotate values.
- While using FK, we have to individually move each joint to the target, and make adjustments respectively.
- While using IK, we just have to grab the end joint and place it to the Target, and the rest of the joints move along respectively.
- When using FK, when we move Bone 1, all the other bones associated with it, move along. (Bone 2, Bone3, and so on).
- While using IK, when we move the End Effector Bone, all the respective bones adjust their orientation with respect to the End Bone & Root Bone.
- In FK, we have more control over each bone than we have in IK.
- FK produces curve along animation, while IK produces linear animation.
- While animating, FK is more time consuming than IK.

- **List 3 ways in which IK can help you as a game developer.**

- In Video Games, IK allows a user to simulate the motion of a chain (of bones) without having to manually control each of them.
- One use case is when we want to grab objects or Pickup objects in a game, at that time IKs are very helpful, as we can place the object wherever we want in the environment, and yet still the Player's hand will be able to locate it when the Player reaches close to that object.
- IKs are also very important in keeping the player grounded, and to adjust the Player's legs accordingly to the type of terrain Player is standing.
- If standing or walking on a slope, Player's legs will bend at an angle with respect to the slope elevation.
- IK are also useful while using weapons, like sword, gun or knives.
- As different weapons will have different Hold Positions and when we attach these objects to the player's hand, they move along the Player's Hand position.
- IK is also useful while shooting, and in moving objects like Blocks, in a game where you have to move blocks in order to unlock or open the door.

- We have only used IKChainConstraint in this example of IK – do a little digging and find another type of constraint you can use with the AnimationRigging package in Unity, name it and explain how/why it can be used.
- **Blend Constraint:** This constraint simple blends the Target object between two Rigs or Animations.
 - You need to define two Source Objects for this. Object A & B.
 - And the “Constraint Object” is the one we want to affect, that is the Target Object.
 - We create a Rig Component and add to it “Blend Constraint”.
 - Further we have a “Settings” panel, which simply has two sliders for the two objects to blend in.
 - And we blend towards these objects with respect to Position and/or Rotation.



- If we keep the Slider value to 0, it will blend towards the Source A Object, and if we give value 1, then it blends towards Source B Object.
- 0.5 sets it in between both the Objects.
- This type of blend is useful in a game scene or level where we want the Player to walk in the middle path, in a straight line without leaning towards Left or Right, or the vice versa case.
- Suppose our Player is walking on a Rope at a certain height, we want the Player to maintain balance.
- And if the Player fails to maintain balance, and lean towards Left or Right Side, then he gets attracted to that side and eventually falls down.
- Example: Prince of Persia Game Series.

-----**THE END**-----