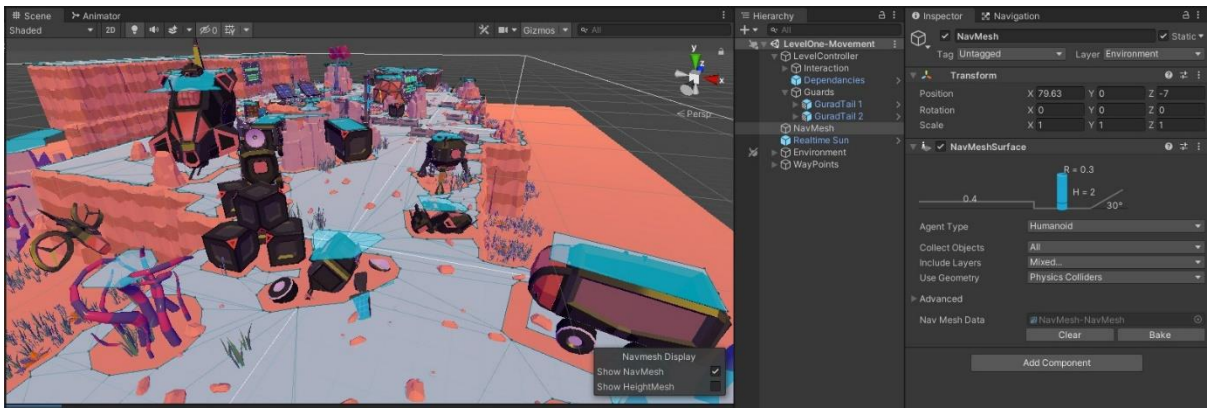
Sprint 01

Assignment 03 : AI Part I

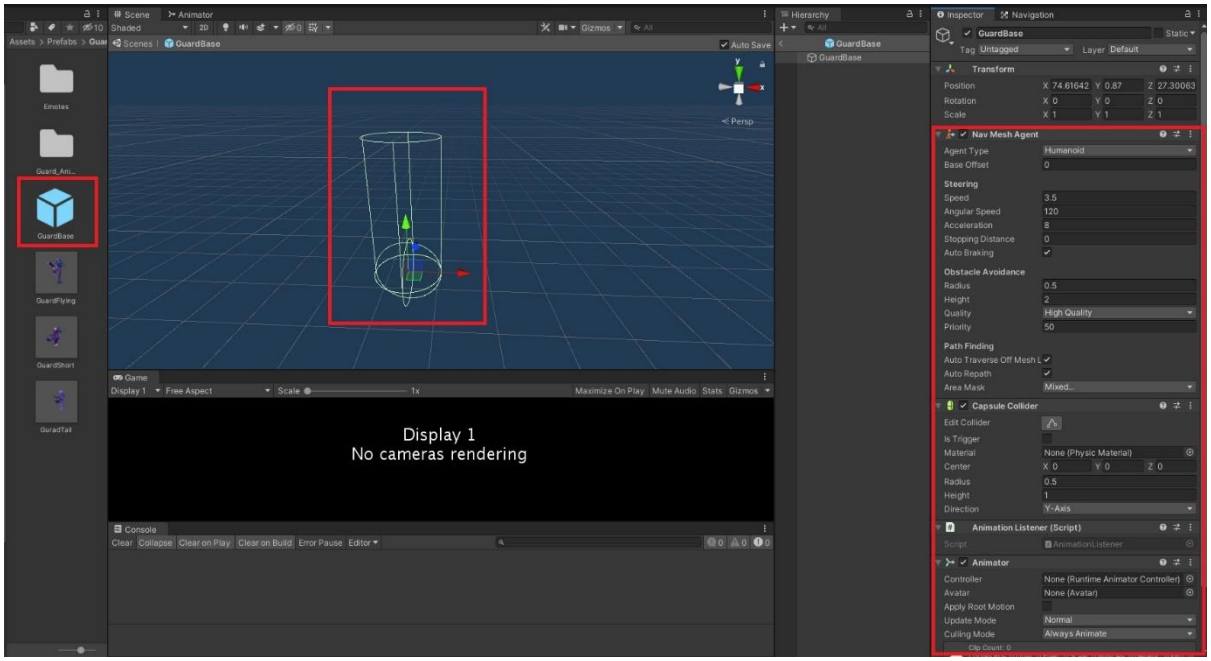
P.S: I really appreciate your feedback and your mention to cut down things to half. And I will be doing so for my every next assignment. But I would also want you to know that I create it for my reference so that I could look back at it anytime in future. And also sorry for I have exceeded this one too, but the AI Part is huge and I have many doubts in it. Hope I won't take much of your time for this documentation.

Step 01: Setup

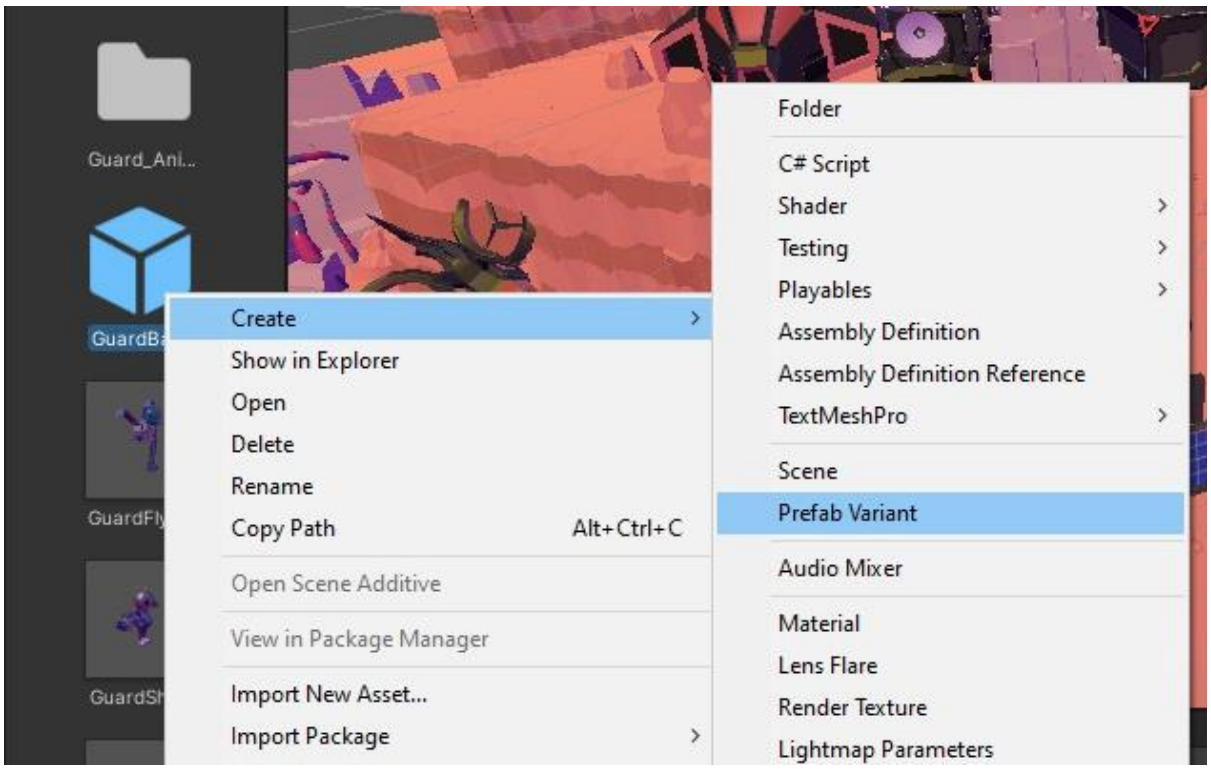
- First, create an Empty GameObject and add "*NavMeshSurface*" component to it.
- Select the terrain/Model and click "Bake" to create a walkable path for the "*NavMeshAgent*".



- Then again create an Empty GameObject and to it add a "*NavMeshAgent*", a "*Capsule Collider*", "*AnimationListener*" script and an "*Animator*" component.
- This will be the Base GameObject structure for all our guards.



- Then right click on the prefab created, and select “Prefab Variant”.
    - **Prefab Variant :** inherits the properties of another Prefab, called the base.



- This prefab variant will inherit all the components we add to the "*GuardBase*" prefab.
- Although, Prefab Variants can also override their properties.
- Follow these steps for every guard.

Step 02: Script & Workflow

- The Scripts workflow is like this:
    - Guards > TailGuard > GuardBehavior > GuardController > GuardManager > LevelController

- The GuardBehavior is further extended as:
    - GuardBehavior > Patrolling & Pursuing

- Patrolling further classifies:
    - PatrolCommand > PatrolMoveTo > PatrolRotate > PatrolWait

- Pursuing follows along with "Pursuing, Predicting Player's Next Position and Rotating towards Player" code logic.
- There is also a Guard Health Script used to used to keep track of guard's health and do damage to it.
- Also, there is an additional script, "*DebugGuardDamager*" which creates the GUI Button which when clicked, does damage to the guards.
- We will be using "***Command Designing Pattern***" in this assignment.
- We will be implementing CommandPattern in the "*PatrolCommand*" and "*GuardBehavior*" scripts.

❖ Guards.

- First, we create the Base Script called "*Guards*".
- This will have the *MonoBehaviour* attached to it.
- This will act as a base interface class which will consists of all the key behaviour elements of a guard, like *GuardVision, Waypoints* and *GuardHealth*.

```
TailGuard.cs   Guards.cs M ×   GuardBehavior.cs U   GuardPursueBehavior.cs U   GuardHealth.cs U
Assets > Scripts > Guards > Guards.cs > Guards > TakeDamage(float damageTaken, Transform instigator)
1   using System;
2   using System.Collections;
3   using System.Collections.Generic;
4   using UnityEngine;
5
    19 references
6   public class Guards : MonoBehaviour     //Assignment - 03
7   {
        5 references
8       public GeneralData generalData;
        2 references
9       public WayPoints waypointsList;
        3 references
10      public VisionData visionData;
11
        2 references
12      public Action<float, Transform> OnDamageTaken = delegate { };
13
        1 reference
14      public void TakeDamage(float damageTaken, Transform instigator)
15      {
16          OnDamageTaken.Invoke(damageTaken, instigator);
17      }
18
19  }
20
```

```
[System.Serializable]   //We needed to add this bcoz it is a Sub-Class under an Existing Class
1 reference
public class GeneralData
{
    1 reference
    public float maxHealth = 100;
    1 reference
    public float attackDamage = 25;
    1 reference
    public float patrolMoveSpeed = 0.5f;
    1 reference
    public float pursuitMoveSpeed = 1.0f;
    0 references
    public float maxSpeed = 10.0f;
    1 reference
    public float attackRotateSpeed = 2;
}

[System.Serializable]
1 reference
public class WayPoints
{
    2 references
    public List<WayPointInfo> wayPoints;
}

[System.Serializable]
2 references
public class WayPointInfo
{
    1 reference
    public WayPointType wayPointType;
    1 reference
    public Transform goal;
    1 reference
    public float waitTime;
    0 references
    public Vector3 targetRotation;
}
```

```
46
47  [System.Serializable]
    4 references
48  public class VisionData
49  {
        1 reference
50      public bool visualize;  //Default value = false
        5 references
51      public float radius = 8.0f;
52
53      [Range(0, 180)]
        6 references
54      public float angle = 30.0f;
        1 reference
55      public float eyeHeight;
        1 reference
56      public float attackRange = 5;
        1 reference
57      public float searchRange = 8;
58
59      [Tooltip("Sets the amount of Raycast Lines.")]
60      [Range(0, 2)]
        1 reference
61      public int raycastLines;
        3 references
62      public int awarnessZone;
        1 reference
63      public LayerMask raycastMask;
64
65  }
66
    4 references
67  public enum WayPointType
68  {
        1 reference
69      MoveTo,
        1 reference
70      Wait,
        1 reference
71      Rotate
72  }
```

Note: *This script is not to be attached to any component, rather it will act as a base class for other scripts.*

❖ TailGuard

- Then, create a new script "*TailGuard*" for the Tail Guard.
- This script will contain no code or logic, also no MonoBehaviour, as it will inherit from the "*Guards*" base script.

```
C# TailGuard.cs ×   C# Guards.cs M   C# GuardBehavior.cs U   C# Guard
Assets > Scripts > Guards > C# TailGuard.cs > TailGuard
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

0 references
public class TailGuard : Guards     //Assignment - 03
{
    // Start is called before the first frame update
    0 references
    void Start()
    {

    }

    // Update is called once per frame
    0 references
    void Update()
    {

    }
}
```

❖ PatrolCommand

```
GuardManager.cs ×   GuardController.cs   PatrolCommand.cs ×   PatrolMoveTo.cs
Assets > Scripts > Guards > Patrolling > PatrolCommand.cs > PatrolCommand > Start()
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

3 references
public abstract class PatrolCommand      //Assignment - 03
{
    2 references
    public abstract void Start();

    1 reference
    public abstract void Update();

    1 reference
    public abstract void End();

    4 references
    public event Action OnCommandComplete;

    2 references
    protected virtual void CommandComplete()
    {
        OnCommandComplete?.Invoke();
    }

}
```

- Then we create an Abstract CommandPattern script called "*PatrolPattern*".
    - **Abstract Class:** a combination of both a normal class and an interface.
    - If you call or inherit an Abstract Class, you need to implement its Functions or methods compulsorily.
    - **Virtual:** used to modify a method, property, indexer, or event declaration and allow for it to be overridden in a derived class.

- We will create 3 Method namely, "*Start()*", "*Update()*" and "*End()*" which will be responsible for Starting the behaviour, Updating it every frame, and Ending current behavior in order to switch to next behavior.

- We also declare an Action delegate, which is called whenever a beavhior is finished.
- Then further we will create 3 more scripts, which will inherit this class.

❖ PatrolMoveTo

```
C# GuardManager.cs   C# GuardController.cs   C# PatrolCommand.cs   C# PatrolMoveTo.cs ×
Assets > Scripts > Guards > Patrolling > C# PatrolMoveTo.cs > ...
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

1 reference
public class PatrolMoveTo : PatrolCommand   //Assignment - 03
{
    6 references
    private NavMeshAgent meshAgent;
    3 references
    private Transform goal;

    //Constructor - Passing of Data from One Script to Another.
    1 reference
    public PatrolMoveTo(NavMeshAgent meshAgent, Transform goal) //Bcoz this will be called in GuardController Script.
    {
        this.meshAgent = meshAgent;
        this.goal = goal;
    }

```

```
    2 references
    public override void Start()
    {
        meshAgent.SetDestination(goal.position); //Same as wayPoint.positin;
    }


    1 reference
    public override void Update() //Moves the Guard
    {
        float distanceToGoal = (meshAgent.destination - meshAgent.transform.position).magnitude;
        if(!meshAgent.hasPath || distanceToGoal < 0.2f)
        {
            CommandComplete(); //Calls 'Event Action' delegate function.
        }
    }

    1 reference
    public override void End()
    {
        Debug.Log( meshAgent + " Moved to : " + "<color=orange>" + goal +  "</color>" + ".");
    }
}

```

- First, we delete Monobheaviour and inherit from PatrolCommand.
- In "*Start()*" function we set the Guard's Position to the Initial WayPoint.

- Then in “Update()”, we calculate the distance between the Guard and the Player and do some.
- We do a If condition where we check if the meshAgent (guard) has no path or its distance goal is less than some amount, then go to the next waypoint.
- Next we create another new CommandPattern script called “*GuardBehavior*” which contains all the Guard Behaviors.

❖ **GuardBehavior**

```
C# TailGuard.cs    C# GuardBehavior.cs U ×    C# Patro
Assets > Scripts > Guards > C# GuardBehavior.cs > ...
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

4 references
public abstract class GuardBehavior
{
    3 references
    public abstract void Start();

    1 reference
    public abstract void Update();

    3 references
    public abstract void End();

}
```

- Then we create another script “*GuardPatrolBehavior*” for patrolling.
- This will contain Move, Rotate and Wait commands.

❖ **GuardPatrolBehavior**

```
C# TailGuard.cs    C# GuardPatrolBehavior.cs U ×    C# GuardBehavior.cs U    C# PatrolMoveT
Assets > Scripts > Guards > Patrolling > C# GuardPatrolBehavior.cs > GuardPatrolBehavior > Exe
using UnityEngine.AI;

1 reference
public class GuardPatrolBehavior : GuardBehavior
{
    3 references
    private Guards guards;
    10 references
    private PatrolCommand currentCommand;
    3 references
    private NavMeshAgent meshAgent;
    4 references
    private WayPointInfo currentWaypoint; //4 Options Menu
    5 references
    private int waypointIndex;
    2 references
    private float patrolMoveSpeed;
    1 reference
    public GuardPatrolBehavior(Guards guard, float patrolMoveSpeed)
    {
        this.guards = guard;
        this.patrolMoveSpeed = patrolMoveSpeed;
        waypointIndex = 0;
        meshAgent = guard.GetComponent<NavMeshAgent>();
        ExecuteCommand();
    }

    3 references
    public override void Start()
    {
        meshAgent.speed = patrolMoveSpeed;
        ExecuteCommand();
    }

```

- We inherit from GuardBehavior.
- We create a Constructor in which we get and pass some required values and set indexes for our waypoints like 0,1,2….

```
1 reference
public override void Update()
{
    if(currentCommand != null)
    {
        currentCommand.Update();    //Moves the Guard (contains code to move the guard - from 'PatrolMoveTo' script)
    }
}

3 references
public override void End()
{

}

3 references
void ExecuteCommand()
{
    //This Line just calls the WayPointInfo Class and selects the First Index Value.
    currentWaypoint = guards.waypointsList.wayPoints[waypointIndex]; //waypointList > WayPoints > WayPointInfo

    switch(currentWaypoint.wayPointType)    //Calls Enum
    {
        case WayPointType.MoveTo:
            //This Line actually moves the Guard by accessing the 'goal' Transfrom.
            currentCommand = new PatrolMoveTo(meshAgent, currentWaypoint.goal);

            //This Function is only executed when it gets called from "Update()" method from "PlayerMoveTo' script.
            currentCommand.OnCommandComplete += NextCommand; //Subscribing using the delegate. Called from PatrolMoveTo - 'Update' method.
            currentCommand.Start(); //Sets Guard to Initial Position (i.e. wayPoint01)
            break;

        case WayPointType.Wait:
            currentCommand = new PatrolWait(currentWaypoint.waitTime);
            currentCommand.OnCommandComplete += NextCommand; //Subscribing using the delegate. Called from PatrolMoveTo - 'Update' method.
            currentCommand.Start(); //Sets Guard to Initial Position (i.e. wayPoint01)
        break;
```

- Then we create a Method where we get all the waypoints i.e., Empty GameObjects that are placed in the scene manually.
- And then we create a Switch statement where we switch between the Move, Rotate and Wait behaviors.

- ***Note:*** *We use "Switch" case and not "If Conditions" here, because we need to switch between various behaviors. Switch Case function makes it easy to do so. If-Else does not give that much compatibility to switch back and forth. Or even if we achieve it using If-Else method, the code will get long, hard to track and make changes.*

```
3 references
private void NextCommand()  //Loops the Guard Patrol Movement
{
    currentCommand.End();
    currentCommand.OnCommandComplete -= NextCommand;    //Garbage Code
    waypointIndex++;
    if(waypointIndex >= guards.waypointsList.wayPoints.Count)
    {
        waypointIndex = 0;
    }

    ExecuteCommand(); //Loops the Move command.

}

}
```

- This method contains logic for next command and loops the guard between waypoints i.e., if the guard reaches last waypoint, it sets its next waypoint to the initial waypoint.

❖ **PatrolWait**

```
GuardManager.cs   TailGuard.cs   PatrolWait.cs ×   GuardController.cs   Guards.cs   PatrolCommand.cs   Pa
Assets > Scripts > Guards > Patrolling > PatrolWait.cs > ...
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

1 reference
public class PatrolWait : PatrolCommand     //Assignment - 03
{
    2 references
    private float waitDuration;
    3 references
    private float timeWaited = 0.0f;

    1 reference
    public PatrolWait(float waitDurations)   //This will be accessed to put the 'wait for seconds' time value.
    {
        this.waitDuration = waitDurations;
    }
    2 references
    public override void Start()
    {

    }

```

- Implements the PatrolCommand Abstract Class.
- We make the guard wait for some “t” seconds of time by passing a logic that checks if the Time.deltaTime is same to the Time we entered, and

until and unless the condition is satisfied, the guard will wait or will be static .

```
        1 reference
19      public override void Update()
20      {
21          timeWaited += Time.deltaTime;
22
23          if(timeWaited >= waitDuration)  //Checks the 'waitDuration' i.e., "Wait" time entered in Inspector.
24          {
25              CommandComplete();
26          }
27      }
        1 reference
28      public override void End()
29      {
30          Debug.Log("and Waited for : " + "<color=red>" + timeWaited + "</color>" + " seconds.");
31      }
32  }
33
```

❖ GuardController

- This script is the Second Parent script that drives all the PatrolCommand Events we just created.
- It is a non-MonoBehaviour class.
- It also controls Guard's Vision, its Health, Behaviors and its Cone of Vision i.e., what he sees script.
- In this script we call all the other child scripts and pass in the required values in their respective Constructor functions.

PatrolWait.cs M GuardController.cs M

Assets > Scripts > Guards > GuardController.cs > GuardController > SetNewState(CurrentGuardState newState)

```
using UnityEngine.AI;

3 references
public class GuardController     //Assignment - 03
{
    12 references
    private GuardBehavior currentBehavior;
    8 references
    private Guards guards;
    5 references
    private GuardVision vision;
    4 references
    private GuardHealth guardHealth;
    2 references
    private NavMeshAgent meshAgent;
    3 references
    private CurrentGuardState currentState;
    2 references
    private Player player;
    3 references
    private Animator animator;

    1 reference
    public GuardController(Guards guard, Player player)  //Bcoz this will be called in GuardManager.
    {
        this.guards = guard;
        this.player = player;
        meshAgent = guard.GetComponent<NavMeshAgent>();
        animator = guards.GetComponent<Animator>();

        vision = new GuardVision(guard, guard.visionData, player);

        vision.OnObjectsInView += ObjectsInVision;
        vision.OnNoObjectsInView += NoObjectsInVision;

        SetPatrolBehavior();

        guardHealth = new GuardHealth(guards.generalData.maxHealth);
        guardHealth.OnDamageTaken += GuardDamaged;
        guardHealth.OnKilled += GuardKilled;

```

```
        guard.OnDamageTaken += (damageAmount,damageSource) => guardHealth.TakeDamage(damageAmount);

    }

    1 reference
    private void GuardDamaged(float damageAmount, float maxHealth)
    {
        damageAmount = Mathf.Clamp(damageAmount, 0, maxHealth);
        animator.SetBool("TakeHitFront", true);
        Debug.Log(guards.gameObject.name + " damaged for " + damageAmount + " Max Health " + maxHealth);
    }

    // Update is called once per frame
    1 reference
    public void Update()
    {
        vision.Update();    //Contains the View Cone & Detection Logic.
        currentBehavior.Update();   //From GuardBehavior Script. (Command Pattern)
    }

    1 reference
    private void ObjectsInVision()
    {
        SetNewState(CurrentGuardState.Pursuing);    //Sets the "newState" to Pursuing.
    }

    1 reference
    private void NoObjectsInVision()
    {
        SetNewState(CurrentGuardState.Patrolling);  //Sets the "newState" to Patrolling.
    }

    2 references
    private void SetNewState(CurrentGuardState newState)
    {
        if(currentState == newState)    //IMP: This line loops the Patrolling action.
        {
            return;
        }
        switch(newState)
        {
            case CurrentGuardState.Patrolling:  //Called from the above line.

                //The below line is IMP bcozin here we set the currentState to some state,
                //so as it can execute-break-execute seamlessly
                currentState = CurrentGuardState.Patrolling;
                SetPatrolBehavior();
            break;

            case CurrentGuardState.Pursuing:
                //The below Line...  same as the above.
                currentState = CurrentGuardState.Pursuing;
                SetPursueBehavior();
            break;

        }
    }

    2 references
    private void SetPatrolBehavior()
    {
        if(currentBehavior != null)
            currentBehavior.End();
        currentBehavior = new GuardPatrolBehavior(guards, guards.generalData.patrolMoveSpeed);
        currentBehavior.Start();
    }

```

- Again, we use "*Switch*" statement here instead of If-Else for better convenience and as they provide back and forth switch between any cases.
- I tried the using the if-else statements instead of the switch cases, but faced problems in returning back to Patrolling State whenever I entered the Pursue State.
- I discovered the reason to use switch statements because they give freedom to Execute-Break-Execute-Return to various States.

```
1 reference
private void SetPursueBehavior()
{
    if(currentBehavior != null)
        currentBehavior.End();
    currentBehavior = new GuardPursueBehavior(guards, player.ObjectData.transform, guards.generalData.pursuitMoveSpeed);
    currentBehavior.Start();
}

1 reference
private void GuardKilled()
{
    Debug.Log("Guard Killed!");
    currentBehavior.End();
    currentBehavior = new GaurdDeathBehavior(meshAgent, animator, vision);
    currentBehavior.Start();
}
}

8 references
public enum CurrentGuardState
{
    3 references
    Patrolling,
    3 references
    Pursuing
}
```

❖ GuardManager

- This script is the Main Parent script that controls the GameController and Guard Scripts.
- It is a MonoBehavior script which is created manually in the "*LevelController*" script.
- We crate a reference to "*GuardManager*" script and create an instance of it.
- We create this instance in the Start() function of the LevelController.

PatrolWait.cs M | LevelController.cs M | GuardManager.cs M | GuardC

Assets > Scripts > Level > LevelController.cs > ...

```
using System;
using System.Collections;
using UnityEngine;
using UnityEngine.AI;

4 references
public class LevelController : MonoBehaviour
{
    1 reference
    public int levelID;
    1 reference
    public Action OnLoadComplete = delegate { };
    1 reference
    public Action<float> OnLevelComplete = delegate { };
    2 references
    public Action<Levels.Data> OnLevelLoadRequest = delegate { };
    2 references
    public Action OnExitRequest = delegate { };
    0 references
    public Action OnPickupCollected = delegate { };

    5 references
    private CameraController cameraController;
    6 references
    private UIController uiController;
    16 references
    private Player player;
    3 references
    private GuardManager guardManager;  //Assignment - 03
    3 references
    private TimeController timecontroller;  //Assignment - 01
    4 references
    private LevelStatsController levelStatsController;

```

```
    uiController = new UIController(player, cameraController.MainCameraTransform,
        levelID, timecontroller, levelStatsController);

    uiController.OnLevelLoad += (level) =>      //Assignment - 02 (to 'GameController')
    {
        OnLevelLoadRequest(level);
    };

    uiController.OnExit += () =>        //Assignment - 02 (to 'GameController')
    {
        OnExitRequest();
    };

    _ = new LevelVFXController(dependancies.vfxLibrary, player.Controller);

    //'transfrom' here simply means "this.transform".
    //We are using transform bcoz the level controller is an Empty GameObject with Transform Component.
    guardManager = new GuardManager(transform, player); //Assignment - 03

    guardManager.player = player;   //Assignment - 03

    OnLoadComplete();
}

1 reference
public void OnLevelLoadResume()
{
    timecontroller?.StartTime();
    Debug.Log("OnLevelLoadResume Called!");
}
```

- Then, back in the "*GuardManager*" script, we get all the Guards present in the scene dynamically, by creating a Dictionary which contains "*Guards"* as ***Key*** and "*GuardController"* as ***Value***.
    - **Dictionary:** used to find objects with the "*Key*" tag and assign them some values using the "*Value*" tag.
- We then create a List of Guards and get every guard present in the scene and assign "*GameController*" script to each and every guard.

PatrolWait.cs M | GuardManager.cs M | GuardController.cs M

Assets > Scripts > Guards > GuardManager.cs > ...

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

2 references
public class GuardManager : MonoBehaviour   //Assignment - 03
{
    2 references
    public Player player;
    2 references
    private List<Guards> guards;
    1 reference
    private Transform levelObject;
    4 references
    private Dictionary<Guards, GuardController> guardDict; //Used bcoz we can Add or Find Components using Dictionary.
    //private List<GuardController> controller;

    1 reference
    public GuardManager(Transform levelObject, Player player)   //Using transform as LevelController has Transform component.
    {
        this.levelObject = levelObject;
        this.player = player;

        guardDict = new Dictionary<Guards, GuardController>();
        guards = new List<Guards>(levelObject.GetComponentsInChildren<Guards>());   //Gets every Guard in the levelController.
        foreach(Guards guard in guards)
        {
            guardDict.Add(guard, new GuardController(guard, player));
            //controller.Add(new GuardController(guard));
            //guards.Add(new Guards());
        }
    }

    // Update is called once per frame
    1 reference
    public void Update()
    {
        if(guardDict != null)
        {
            foreach(var guardEntry in guardDict)
            {
                guardEntry.Value.Update(); //Calls Update() from 'GuardController' sccript.
            }
        }
    }
}
```

❖ GuardVision

- This script gives the Guard a Cone of Vision which is nothing but simply multiple Raycast lines that recognise the objects that get into the vision.
- We will use these lines to detect Player and follow him if he gets inside the Cone of Vision.

- We will set a Radius, an Angle of View and an Extra Awareness Zone if the Player tries to sneak from behind, the guard would get alerted by hearing Player's footsteps (It's an Add-On feature I did).
- Also, I have added the guard's awareness functionality as if it detects the player, it Cone of Vision extends and the guard follow the Player everywhere and won't leave until he kills the player or get killed himself.

PatrolWait.cs M | GuardVision.cs M | LevelController.cs M | GuardManager.cs M

Assets > Scripts > Guards > Detection > GuardVision.cs > ...

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

public class GuardVision : MonoBehaviour
{

    public Action OnObjectsInView = delegate { };

    public Action OnNoObjectsInView = delegate { };

    public List<GameObject> objectsInCone;

    public VisionData visionData;

    private Player player;

    private Guards guard;

    public Transform playerHead;

    private bool visionEnabled = true;

    public GuardVision(Guards guard, VisionData visionData, Player player)
    {
        this.guard = guard;
        this.visionData = visionData;
        this.player = player;
        objectsInCone = new List<GameObject>();
        playerHead = player.ObjectData.Head.transform;
    }

```

```
// Update is called once per frame
1 reference
public void Update()
{
    objectsInCone.Clear(); //IMP: To delete Garbage Data.

    if(visionEnabled == false)
    {
        return;
    }

    //This cretaes an Empty Transform GameObject onto the Gurad.
    //"guard.transform.position.y + visionData.eyeHeight" - This line increments the "Y" position value,
    //so as to bring it up to the eye level of the Guard.
    Vector3 eyePosition = new Vector3
    (
        guard.transform.position.x,
        guard.transform.position.y + visionData.eyeHeight,
        guard.transform.position.z
    );

    //Calculating Distance and Angle between Player from the EyePosition of Enemy.
    Vector3 distance = playerHead.position - eyePosition;  //Did no use magnitude here bcoz we need to pass a Vector3 value in the "angle" field below.
    float angle = Vector3.Angle(distance, guard.transform.forward);

    //Debug.DrawRay(eyePosition, guard.transform.forward * 5, Color.green);

    //Used magnitude here bcoz the radius and angle values are in "float".
    if(distance.magnitude < visionData.radius && angle < visionData.angle || distance.magnitude <= visionData.awarnessZone)
    {
        float raycastDistance = Vector3.Distance(eyePosition, playerHead.position);   //Added here fo Performance sake.
        if(!Physics.Raycast(eyePosition, distance, visionData.radius, visionData.raycastMask))
        {
            objectsInCone.Add(player.ObjectData.gameObject);
            Debug.Log(objectsInCone);
        }
    }

```

```
        //Checks whether the Player is in the View Radius of the Guard or Not.
        if(objectsInCone.Count == 0)
        {
            OnNoObjectsInView();
        }
        else
        {
            OnObjectsInView();
        }

        if(visionData.visualize)    //Default value = false
        {
            DrawAwarnessZone();
            DrawCone();
        }

    }

    1 reference
    public void Enable()
    {
        visionEnabled = true;
    }
    1 reference
    public void Disable()
    {
        visionEnabled = false;
    }
```

```
    private void DrawCone()
    {
        /////* Draw Outer Lines */////

        //Draws a line starting from the Guard's Foward Direction(Origin Point)
        //to the Radius of the Circle(visionData.radius).
        Vector3 scaledForward = guard.transform.forward * visionData.radius;

        //Rotates the line drawn.(in +ve)
        Vector3 rotatedForward = Quaternion.Euler(0, visionData.angle, 0) * scaledForward;
        Debug.DrawRay(guard.transform.position, rotatedForward, Color.green);

        //Rotates the line drawn in Opposite Direction.(i.e., in -ve)
        rotatedForward = Quaternion.Euler(0, -visionData.angle, 0) * scaledForward;
        Debug.DrawRay(guard.transform.position, rotatedForward, Color.green);


        //////* Draw Inner Lines *////////

        var rayColor = objectsInCone.Count <= 0 ? Color.white : Color.red;  //Ternary Operator

        //Creates line(s). The value "5" represents Degree Angles i.e., for each 5 Degrees,
        //create 'n' number of lines.(visionData.raycastLines)
        int iterations = ((int)visionData.angle / 5) * visionData.raycastLines;

        for(int i = 1; i < iterations; i++)
        {
            //The value 2 is used as we are using Two Lines to craete the Cone.
            float rotateAmount = visionData.angle / iterations * 2 * i - visionData.angle;
            rotatedForward = Quaternion.Euler(0, rotateAmount, 0) * scaledForward;
            Debug.DrawRay(guard.transform.position, rotatedForward, rayColor);
        }
    }

```

```

    //If Player tries to seek from behind the Guard, then Guard detects the Player.
    1 reference
    private void DrawAwarnessZone()
    {
        var rayColor = objectsInCone.Count <= 0 ? Color.blue : Color.yellow;  //Ternary Operator

        for(int i = 0; i < 36; i++)
        {
            Vector3 endPoint = Quaternion.Euler(0, i * 10, 0) * new Vector3(0, 0, visionData.awarnessZone);
            Debug.DrawLine(guard.transform.position, guard.transform.position + endPoint, rayColor);
        }
    }

}

```

❖ **GuardPursueBehavior**

- This is similar to PatrolBehavior, and implements the "*GuardBehavior*" Command.
- It is this script, where we add functionalities to the guard to look towards the Player and Rotate, looking forward to the Player as he changes position.
- Then the guard follow the player till a certain amount of range, and then Attack the Player if he reaches a certain amount of distance.
- In short, we will create 2 radius ranges for the guard, on in which he follows the player, if the player is in the 1st range, and when the player reaches in the 2nd range, the guard attacks.

PatrolWait.cs M | GuardPursueBehavior.cs U | GuardVision.cs M | LevelController.cs M | GuardMa

Assets > Scripts > Guards > Pursuing > GuardPursueBehavior.cs > ...

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

1 reference
public class GuardPursueBehavior : GuardBehavior    //Assignment - 03
{
    6 references
    private Guards guards;
    12 references
    private NavMeshAgent meshAgent;
    2 references
    private Animator guardAnimator;
    5 references
    private Transform targetObject;
    5 references
    private VisionData visionData;
    1 reference
    private float pursueMoveSpeed;
    2 references
    private float attackRotateSpeed;
    3 references
    private bool playerInRange;
    2 references
    private Vector3 targetPreviousPosition;

    1 reference
    public GuardPursueBehavior(Guards  guard,Transform targetObject, float pursueMoveSpeed)
    {
        this.guards = guard;
        this.targetObject = targetObject;
        this.pursueMoveSpeed = pursueMoveSpeed;
        this.attackRotateSpeed = guards.generalData.attackRotateSpeed;
        guardAnimator = guards.GetComponent<Animator>();
        this.visionData = guards.visionData;
        meshAgent = guards.GetComponent<NavMeshAgent>();
        //targetPreviousPosition = targetObject.position;
    }

```

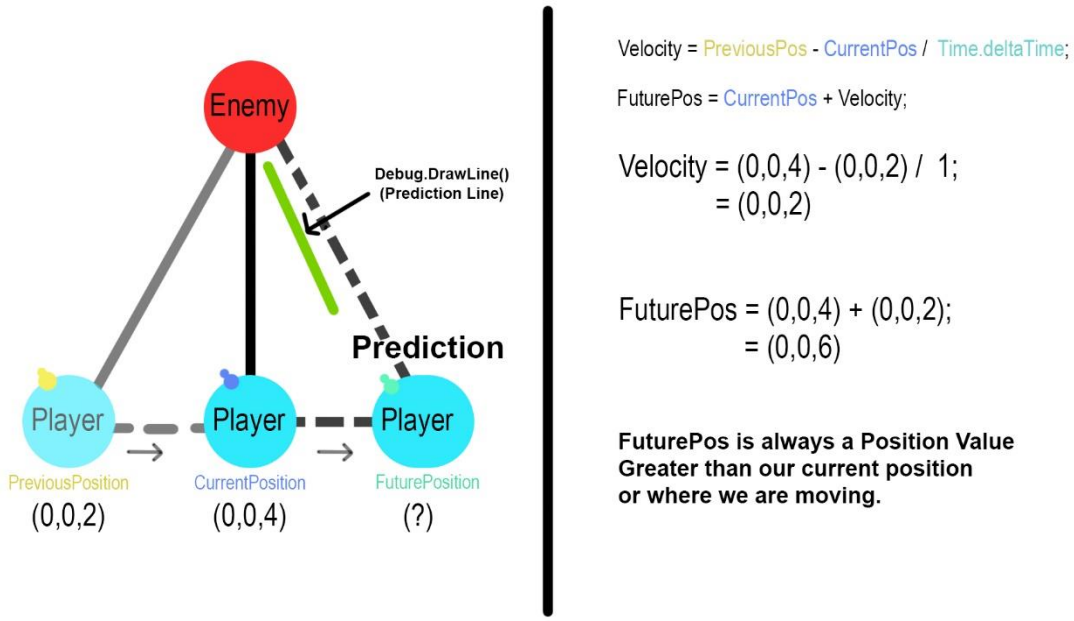
```
3 references
public override void Start()
{

}

1 reference
public override void Update()
{
    float distanceToTarget = Vector3.Distance(meshAgent.transform.position, targetObject.position);
    if(distanceToTarget <= visionData.attackRange || playerInRange)
    {
        playerInRange = true;
        meshAgent.isStopped = true;
        meshAgent.updateRotation = false;
        meshAgent.velocity = Vector3.zero;

        visionData.radius = 20;
        visionData.awarnessZone = 20;

        //Do not use this (LookAt) bcoz it suddenly looks at the player without any Rotation or Time interval.
        //guards.transform.LookAt(targetObject.transform, Vector3.up);
        RotateTowardsTarget();

        guardAnimator.SetTrigger("Attack");
    }
    else
    {
        meshAgent.isStopped = false;
        meshAgent.updateRotation = true;
        //meshAgent.SetDestination(targetObject.transform.position);    //Moves the Guard towards the Player.
        meshAgent.SetDestination(PredictFuturePosition());  //Predicting using Player's Position.
    }

```

- In this function, I increased the Cone of Vision's radius so that the guard follows the player everywhere in the map, once detected.

```
        if(distanceToTarget >= visionData.searchRange)
        {
            playerInRange = false;
            //visionData.radius = 5;
            //visionData.awarnessZone = 5;
        }

        targetPreviousPosition = targetObject.position;

    }

    1 reference
    private Vector3 PredictFuturePosition()
    {
        Vector3 targetCurrentPosition = targetObject.position;

        if(Time.deltaTime == 0)
        {
            return targetCurrentPosition;
        }

        //The prediction is done by getting Player's Movement (in position X, Y & Z co-ordinates)
        //i.e., veclocity = currentPosition - previousPosition / Time.deltaTime
        //and then (prediction) futurePos =  currentPosition + veclcity
        //Calculation of prediction
        Vector3 targetVelocity = (targetCurrentPosition - targetPreviousPosition) / Time.deltaTime;
        Vector3 futurePos = targetCurrentPosition + targetVelocity;

            //This is Extra Code to draw line for visualization purpose
            Vector3 end = new Vector3(futurePos.x, futurePos.y, futurePos.z + 5);
            Debug.DrawLine(futurePos, end, Color.blue);

        return futurePos;

        //Note: FuturePos is always a Position Value Greater than where we are moving.

    }
```

- The Predict logic works like this:
    - First we get the Player's "***PreviousPosition***" which is the position as soon as the Player gets detected.
    - Then we store it in a variable "***targetPreviousPosition***".
    - Then we also get Player's "***CurrentPosition***" and store it.
    - We get this value inside the "***Predict()***" function.
    - Then we calculate the Velocity of the Player by subtracting the Player's Current Position by its Previous Position and dividing it by ***Time.deltaTime*** (as we take frame change time into consideration to avoid any jitter effect or bug like - Guard gets teleported to a new location instantly without any animation or time interval).
    - Then we Add the Player's Current Position and the Guard's Target Velocity to predict Player's Next Position.

- ***Note:*** *Player's Future Position is always a Position Value Greater than its current position or where we are moving. Also, the prediction only applies when we move the Player. If the player is stable the prediction code does not predict Future Position - as the Player's Previous Position and Current Position is same which eventually returns a 0 value (after Subtraction).*



(I created this Image for my reference – Pleas also look if the logic is right)

```
        1 reference
        private void RotateTowardsTarget()
        {
            Vector3 goalDirection = targetObject.position - meshAgent.transform.position;
            goalDirection.y = 0;
            goalDirection = goalDirection.normalized;
            float stepAmount = attackRotateSpeed * Time.deltaTime;
            Vector3 newDirection = Vector3.RotateTowards(guards.transform.forward, goalDirection, stepAmount, 0.0f);
            meshAgent.transform.forward = newDirection;

        }

        3 references
        public override void End()
        {
            meshAgent.isStopped = false;
            meshAgent.updateRotation = true;
        }
    }
```

❖ GuardHealth

C# PatrolWait.cs M | C# GuardHealth.cs U × | C# GuardPursueBehavior.cs U

Assets > Scripts > Guards > Health > C# GuardHealth.cs > ...

```
2 references
public class GuardHealth    //Assignment - 03 Part I
{
    2 references
    public Action<float, float> OnDamageTaken = delegate { }
    2 references
    public Action OnKilled = delegate { };
    2 references
    private float maxHealth;
    6 references
    public float currentHealth;
    1 reference
    public  GuardHealth(float maxHealth)
    {
        this.currentHealth = maxHealth;
        this.maxHealth = maxHealth;
    }

    1 reference
    public void TakeDamage(float damageTaken)
    {
        currentHealth -= damageTaken;

        OnDamageTaken.Invoke(currentHealth, maxHealth);

        if(currentHealth <= 0)
        {
            OnKilled.Invoke();
            currentHealth = 0;
        }
    }

    0 references
    public bool IsAlive
    {
        get
        {
            return currentHealth > 0;
        }
    }
}
```

❖ DebugGuardDamager

- This script creates a button on screen which Damages the Guard's Health by some amount.

```
C# PatrolWait.cs M    C# DebugGuardDamager.cs U ×    C# GuardHealth.cs U    C# GuardPu
Assets > Scripts > Guards > Health > C# DebugGuardDamager.cs > ...
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEditor;
5
    0 references
6   public class DebugGuardDamager : MonoBehaviour
7   {
        7 references
8       Guards[] guards;
9       //private GuardHealth guardHealth;
        2 references
10      bool damageButton;
11
        0 references
12      private void Start()
13      {
14          //guardHealth = this.GetComponent<GuardHealth>();
15          //guardHealth = new GuardHealth(0);
16      }
17
18      // Update is called once per frame
        0 references
19      void Update()
20      {
21          #if UNITY_EDITOR
22              guards = gameObject.GetComponentsInChildren<Guards>();
23          #endif
24      }
25
```

- The method **#if…#endif** used in Update() is known as ***Platform Independent Compilation.***
- This compiles only in the Editor and no in the build.
- Like if we use **UNITY_PS4** and add some code in it, the it will only compile or execute when the Game runs on a *PS4*.

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

        //if(GUI.Button(new Rect(10, 70, 50, 30), "Click"))
            //Debug.Log("Clicked the button with Text!");

        for(int i = 0; i < guards.Length; i++)
        {
            if(guards[i] == null)
            {
                continue;
            }

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

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

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

                /*if(guardHealth.currentHealth == 0)
                {
                    damageButton = false;
                    Debug.Log("You cannot Damage any Further!");
                }*/
            }

            //GUI.enabled = true;

        }
    }
}
```

❖ GuardDeathBehavior

C# DebugGuardDamager.cs U | C# GaurdDeathBehavior.cs U × | C# GuardHealth.cs U | C# GuardPursueBehavior.cs U | C# GuardVi

Assets > Scripts > Guards > Death > C# GaurdDeathBehavior.cs > ...

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

1 reference
public class GaurdDeathBehavior : GuardBehavior
{
    12 references
    private NavMeshAgent meshAgent;
    4 references
    private Animator animator;
    3 references
    private GuardVision vision;
    3 references
    private float currentTime;
    2 references
    private float waitTime = 5.0f;

    1 reference
    public GaurdDeathBehavior(NavMeshAgent meshAgent, Animator animator, GuardVision vision)
    {
        this.meshAgent = meshAgent;
        this.animator = animator;
        this.vision = vision;
    }
    3 references
    public override void Start()
    {
        animator.SetBool("Dead", true);
        meshAgent.ResetPath();
        meshAgent.isStopped = true;
        meshAgent.updateRotation = false;
        meshAgent.velocity = Vector3.zero;
        meshAgent.GetComponent<CapsuleCollider>().enabled = false;  //GetComponentInChildren also works
        meshAgent.GetComponent<NavMeshAgent>().enabled = false;
        vision.Disable();
        Debug.Log("<b>Guard's Body will Fade Out in 5 Seconds.</b>");

    }

```

```
1 reference
public override void Update()
{
    currentTime += Time.deltaTime;

    if(currentTime >= waitTime)
    {
        animator.SetBool("Dead_Fade", true);

        //Fade Out Delay - Need to set it bcoz, if we set to to just "waitTime" i.e.(5 seconds)
        //then it disables the Guard GameObject before playing the "Dead_Fade" animation.
        //So to disable it only after the Fade animation is played, we need to add delay of some time seconds.
        //Note: The Time values should always be above 2, bcoz it takes 1 second to complete the Fade animation.
        if(currentTime >= waitTime + 2)
            meshAgent.gameObject.SetActive(false);   //Big Savior
    }
}

3 references
public override void End()
{
    //To bring the guard back to life
    animator.SetBool("Dead", false);
    meshAgent.isStopped = false;
    meshAgent.updateRotation = true;
    meshAgent.GetComponent<CapsuleCollider>().enabled = true;
    meshAgent.GetComponent<NavMeshAgent>().enabled = true;
    vision.Enable();
}

}
```

- In here I have created a Death Fade Out Animation using the Animation clip.
- When the Guard is Dead, after 5 seconds the Death_Fade Out animation plays.

- ***Note:*** *The comparison value I have set to 2, because the Fade Out Animation requires some time to complete. It's value always should be set to 2 or above.*

- I have set a Trigger called "*Dead*" and a Bool called "*Dead_Fade*" and passed the respective conditions to the connection nodes.

- ***Note:*** *I have purposely not used the default "IsDead" Bool as it is a Boolean value. When you use a True Boolean state, it remains constant. If I had used the IsDead the Guard would die and then Respawn again because after that the Guard enters the "AnyState" action in Animator. That's why I created a new "Dead" Trigger because it can switch from trigger to bool. And finally stop at Bool "Dead_Fade".*

- Also, I made the Damage Button disappear after the guard enters the Dead State.

Step 03: What have I learnt

All the above things mentioned.

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