Create a folder under Assets named “Scripts”. We are going to make a total of 5 C# scripts, so let's get started! Right click on the empty Scripts folder and Create> C# script, rename it to “IEnemyState”. This is going to be an interface that will allow us to manage a sort of code contract between our different scripts. Change the code to be as the following:

```
using UnityEngine;
using System.Collections;

public interface IEnemyState
{

    void UpdateState();

    void OnTriggerEnter (Collider other);

    void ToPatrolState();

    void ToAlertState();

    void ToChaseState();

}
```

Next, we are going to make the script describing our first state. Create under the Scripts folder a new script named “PatrolState” and change the code like this:

```
using UnityEngine;
using System.Collections;

public class PatrolState : IEnemyState

{
    public void UpdateState()
    {

    }

    public void OnTriggerEnter (Collider other)
    {

    }

    public void ToPatrolState()
    {

    }

    public void ToAlertState()
    {

    }

    public void ToChaseState()
    {

    }
}
```

Notice that this class inherits the interface we created earlier. All these empty functions that are inherited from the IEnemyState are going to be needed to the other state scripts we are going to create. So what you need to do, is first create all the other state scripts we need and then paste the code above to each new script. Right click under Scripts and create C# scripts named AlertState and ChaseState. Remember: When pasting the code from the one script to the other, do not forget to change line 4 from `public class PatrolState : IEnemyState` to `public class ChaseState : IEnemyState` and `public class AlertState : IEnemyState` accordingly.

Now let's create our fifth and last script, named StatePatternEnemy. Change the code like this:

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class StatePatternEnemy : MonoBehaviour
5  {
6      public float searchingTurnSpeed = 120f;
7      public float searchingDuration = 4f;
8      public float sightRange = 20f;
9      public Transform[] wayPoints;
10     public Transform eyes;
11     public Vector3 offset = new Vector3 (0,.5f,0);
12     public MeshRenderer meshRendererFlag;
13
14
15     [HideInInspector] public Transform chaseTarget;
16     [HideInInspector] public IEnemyState currentState;
17     [HideInInspector] public ChaseState chaseState;
18     [HideInInspector] public AlertState alertState;
19     [HideInInspector] public PatrolState patrolState;
20     [HideInInspector] public NavMeshAgent navMeshAgent;
21
22     private void Awake()
23     {
24         chaseState = new ChaseState (this);
25         alertState = new AlertState (this);
26         patrolState = new PatrolState (this);
27
28         navMeshAgent = GetComponent<NavMeshAgent> ();
29     }
30
31     // Use this for initialization
32     void Start ()
33     {
34         currentState = patrolState;
35     }
36
37     // Update is called once per frame
38     void Update ()
39     {
40         currentState.UpdateState ();
41     }
42
43     private void OnTriggerEnter(Collider other)
44     {
45         currentState.OnTriggerEnter (other);
46     }
47 }
```

To PatrolState.cs, modify the code like this:

```
using UnityEngine;
using System.Collections;

public class PatrolState : IEnemyState

{
    private readonly StatePatternEnemy enemy;
    private int nextWayPoint;

    public PatrolState (StatePatternEnemy statePatternEnemy)
    {
        enemy = statePatternEnemy;
    }

    public void UpdateState()
    {
        Look ();
        Patrol ();
    }

    public void OnTriggerEnter (Collider other)
    {
        if (other.gameObject.CompareTag ("Player"))
            ToAlertState ();
    }

    public void ToPatrolState()
    {
        Debug.Log ("Can't transition to same state");
    }

    public void ToAlertState()
    {
        enemy.currentState = enemy.alertState;
    }

    public void ToChaseState()
    {
        enemy.currentState = enemy.chaseState;
    }

    private void Look()
    {
        RaycastHit hit;
        if (Physics.Raycast (enemy.eyes.transform.position, enemy.eyes.transform.forward, out hit, enemy.sightRange) && hit.collider.CompareTag ("Player")) {
            enemy.chaseTarget = hit.transform;
            ToChaseState();
        }
    }

    void Patrol ()
    {
        enemy.meshRendererFlag.material.color = Color.green;
        enemy.navMeshAgent.destination = enemy.wayPoints [nextWayPoint].position;
        enemy.navMeshAgent.Resume ();

        if (enemy.navMeshAgent.remainingDistance <= enemy.navMeshAgent.stoppingDistance && !enemy.navMeshAgent.pathPending) {
            nextWayPoint =(nextWayPoint + 1) % enemy.wayPoints.Length;

        }


    }
}
```

Basically what is happening here, is that we tell the enemy to transition from a patrol state to a

chasing state if it locates the Player, or transition to an alert state if it collides with the player. If not and it remains in patrol state, then a flag on top of its head (we will elaborate on this later on), will remain green.

To AlertState.cs modify the code like this:

```
using UnityEngine;
using System.Collections;

public class AlertState : IEnemyState

{
    private readonly StatePatternEnemy enemy;
    private float searchTimer;

    public AlertState (StatePatternEnemy statePatternEnemy)
    {
        enemy = statePatternEnemy;
    }

    public void UpdateState()
    {
        Look ();
        Search ();
    }

    public void OnTriggerEnter (Collider other)
    {

    }

    public void ToPatrolState()
    {
        enemy.currentState = enemy.patrolState;
        searchTimer = 0f;
    }

    public void ToAlertState()
    {
        Debug.Log ("Can't transition to same state");
    }

    public void ToChaseState()
    {
        enemy.currentState = enemy.chaseState;
        searchTimer = 0f;
    }

    private void Look()
    {
        RaycastHit hit;
        if (Physics.Raycast (enemy.eyes.transform.position, enemy.eyes.transform.forward, out hit, enemy.sightRange) && hit.collider.CompareTag ("Player")) {
            enemy.chaseTarget = hit.transform;
            ToChaseState();
        }
    }

    private void Search()
    {
        enemy.meshRendererFlag.material.color = Color.yellow;
        enemy.navMeshAgent.Stop ();
        enemy.transform.Rotate (0, enemy.searchingTurnSpeed * Time.deltaTime, 0);
        searchTimer += Time.deltaTime;

        if (searchTimer >= enemy.searchingDuration)
            ToPatrolState ();
    }


}
```

When an enemy is in alert state, all he has to execute is a look around action to make sure that the player is no longer in sight. As long as the enemy stays in the alert state, the flag is going to be yellow. Again, if the enemy locates the player, he transitions to the chase state.

Finally, to ChaseState.cs modify the code like this:

```
using UnityEngine;
using System.Collections;

public class ChaseState : IEnemyState

{

    private readonly StatePatternEnemy enemy;


    public ChaseState (StatePatternEnemy statePatternEnemy)
    {
        enemy = statePatternEnemy;
    }

    public void UpdateState()
    {
        Look ();
        Chase ();
    }

    public void OnTriggerEnter (Collider other)
    {

    }

    public void ToPatrolState()
    {

    }

    public void ToAlertState()
    {
        enemy.currentState = enemy.alertState;
    }

    public void ToChaseState()
    {

    }
```

```
    private void Look()
    {
        RaycastHit hit;
        Vector3 enemyToTarget = (enemy.chaseTarget.position + enemy.offset) - enemy.eyes.transform.position;
        if (Physics.Raycast (enemy.eyes.transform.position, enemyToTarget, out hit, enemy.sightRange) && hit.collider.CompareTag ("Player")) {
            enemy.chaseTarget = hit.transform;

        }
        else
        {
            ToAlertState();
        }

    }

    private void Chase()
    {
        enemy.meshRendererFlag.material.color = Color.red;
        enemy.navMeshAgent.destination = enemy.chaseTarget.position;
        enemy.navMeshAgent.Resume ();
    }


}
```

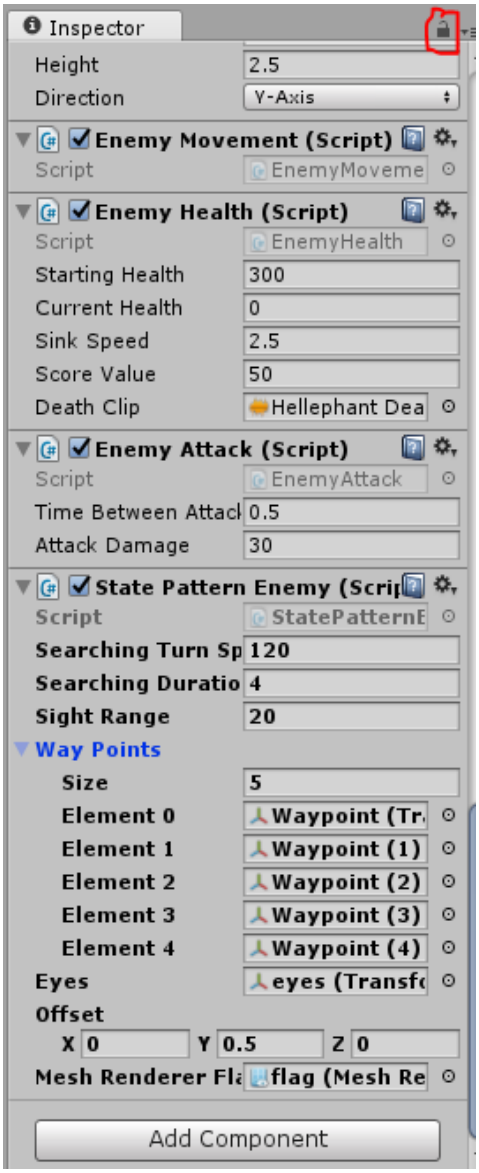
During this state, the flag on top of the enemy's head is red and while the enemy can see the player, he remains in this state. Otherwise it transitions to an alert state, from where he will take another

decision.

Open the FSMtutorial project (located on D2L page) and drag the StatePatternEnemy script on the Hellephant prefab in the hierarchy. What the enemy elephant needs, is a pair of eyes and a flag above its head so we can properly debug if all of our scripts are functioning correctly. What do I mean by flag? Basically we will make a plain 3D cube, parent it to the Hellephant, position it above its head and check the colour changes. It should be green when the enemy is patrolling, red when it is chasing and yellow when it is in alert state.

Create a 3D cube named “flag”, and after you parent it to the Hellephant, change its Y axis as high as you prefer. After that, create an empty gameobject named “eyes”, parent it again to the Hellephant and position it to X: 0, Y: .9, Z: .4.

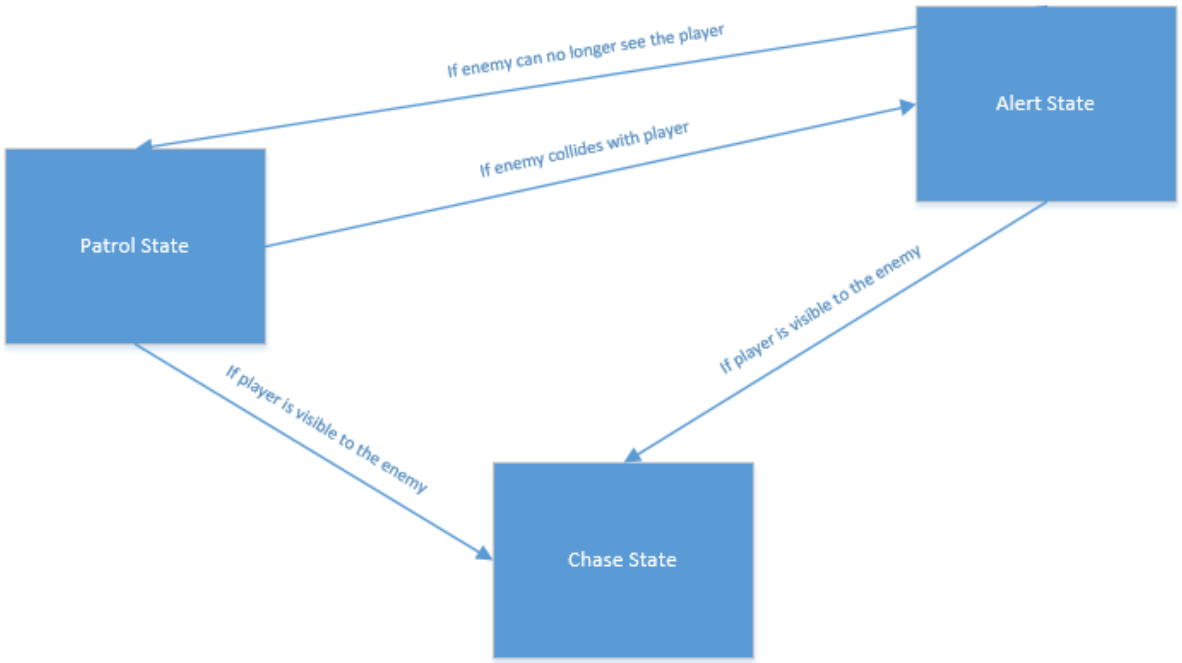
Another part we need are the waypoints. These are the spots the Hellephant can recognise and access while patrolling. Simply create as many empty gameobjects as you want (I created just 5 of them) and scatter them around. Now while you have clicked on the Hellephant and you can see the StatePatternEnemy script on the inspector, lock the inspector by pressing the small lock on the top right corner, and then shift select all of your waypoints and drag them on the Way Points field on the StatePatternEnemy script.



Save your scene and project and then run the game. Watch the Hellephant change the colour on its

flag while its transitioning through states. When you are right in front of it, the flag is bright red but if you hide behind an object, then the flag will turn yellow and thengreen.

So, why did we use a technique like this and not a bunch of “if statements”? Because this way we keep an organized code that can be extended by someone else. It is less messy and this is what is called a Finite State Machine. Imagine this whole process like the states and transitions you saw during the Mecanim Tutorials. The FSM we have after all this code is the following:



So you can easily see that we keep all the possible outcomes we designed for the game more organized instead of just throwing if statements everywhere, and it is easy for someone to keep track of what has been already implemented.