CODE NINJAS®

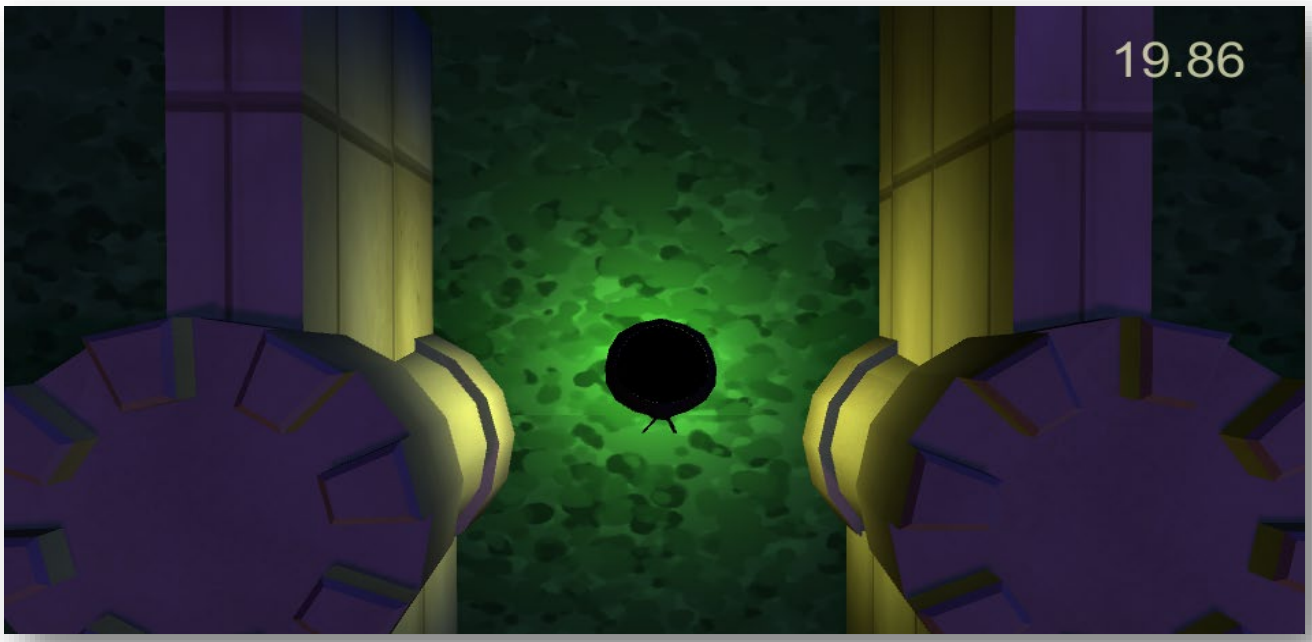
Silver Belt Ninja Guide
Activity 09: Labyrinth

Activity 9

Labyrinth

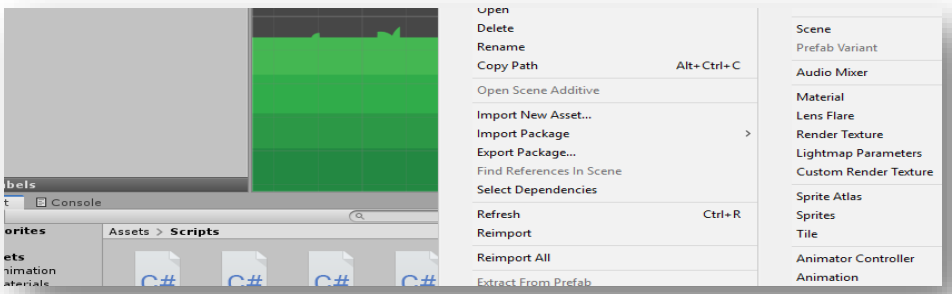
A familiar maze awaits you. There's just one trick – you can no longer control Codey the Ninja. You could probably tell Codey what to do programmatically, by writing code telling Codey how many steps to take in which direction until the exit is found, but is there a better way? Unity has a built-in system known as the **NavMesh** that you can use instead. What does this **NavMesh** do?

The **NavMesh** is an AI navigation system that comes with Unity that is very powerful, but the essentials are very basic. It tells a target object, called an agent, where it can and cannot move. It then uses this information to figure out the shortest path to its destination. You will help Codey escape the labyrinth from Find the Exit using this NavMesh system. You will learn how to apply the NavMesh, which decides what is and isn't a walkable terrain, to a scene, and how to interact with the NavMeshAgent component by having it control elements of Codey's automatic behavior.

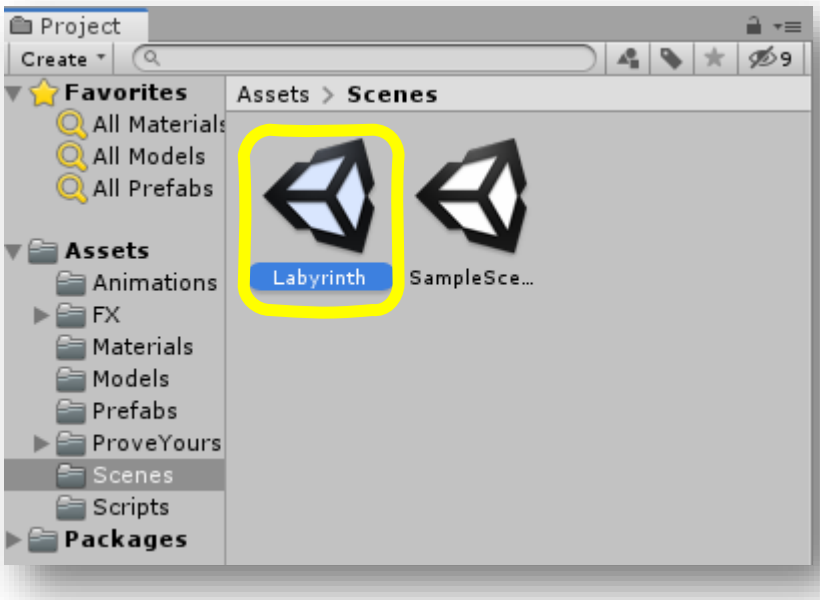


1 Start a new Unity Project and name it *YOUR INITIALS – Labyrinth*. Select **3D core** then **Create**.

2 We've created a starter pack to give you a head start! To use it, import the **Activity 09 - Labyrinth.unitypackage** by going to **Assets > Import Package > Custom Package> All > Import**.



3 To open the starter package, double-click on the **Labyrinth** scene. You can find this in the **Project** tab under **Assets > Scene > Labyrinth.**



4 This will load our **Hierarchy** with game objects alongside what looks like the original starting version of Find the Exit.

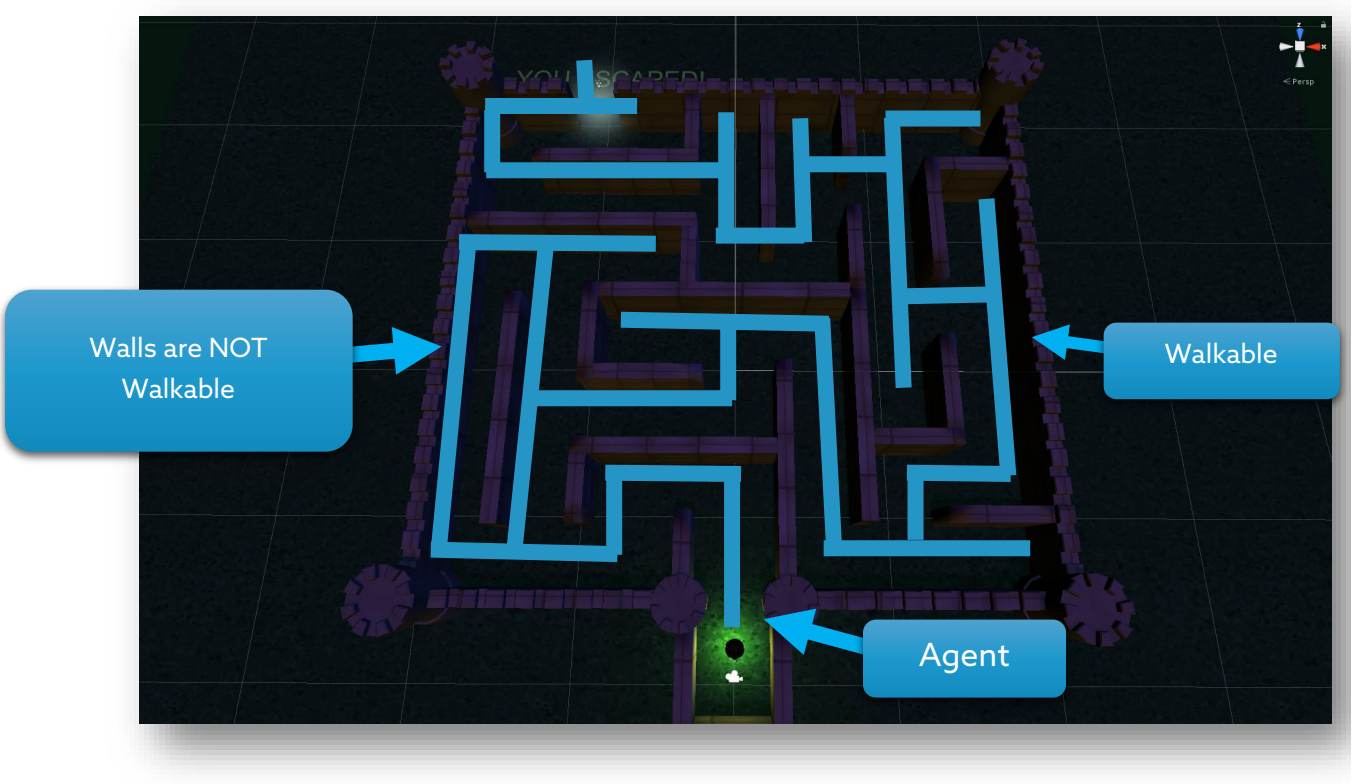
5 **Play** your game and see what happens.

The maze is set up, and the walls and effects work, but the Player has no movement code.

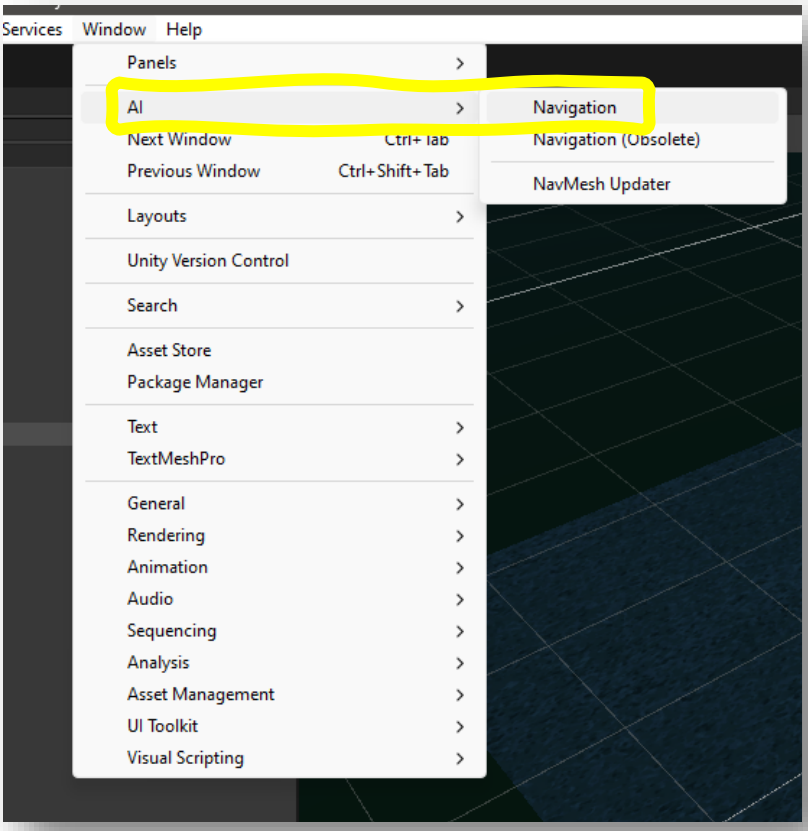
6 In order to navigate Codey to the exit, the game will need **three pieces** of information provided by you.

- What parts of the scene are "**walkable**"; where should Codey be allowed to move?
- What parts of the scene are "**non-walkable**"; where Codey will have to move around or over?
- Who is the "**agent**", the object trying to navigate the scene?

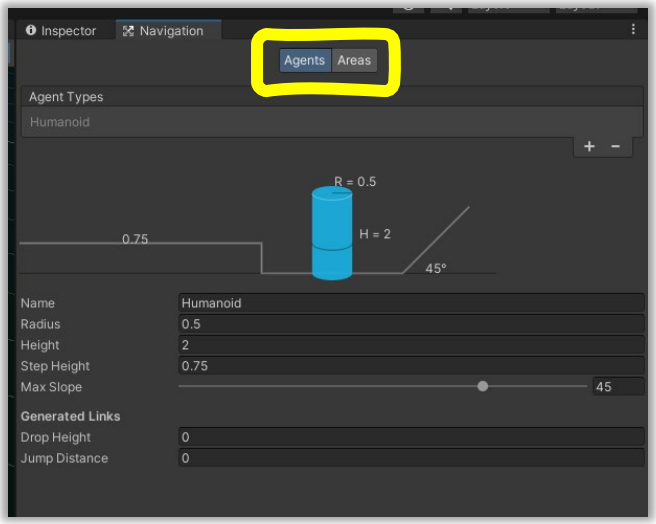
Below you can see all the answers to the questions from above.



7 You will need to add a window next to your Inspector window that shows naviation information. To do this, go to the menu, and find your way to **Window > AI > Navigation**. If it is not here, make sure to install the "AI Navigation" package from the Unity Registry.

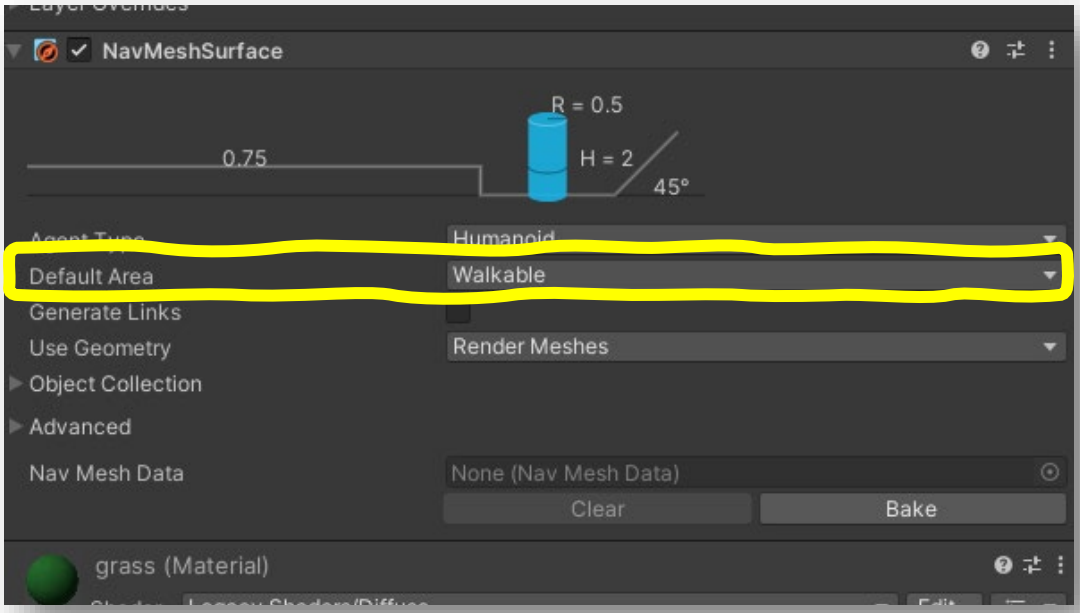


A new window will be added to your Unity interface. You'll see two different sub-menus in the Navigation panel, **Agents** and **Areas**.

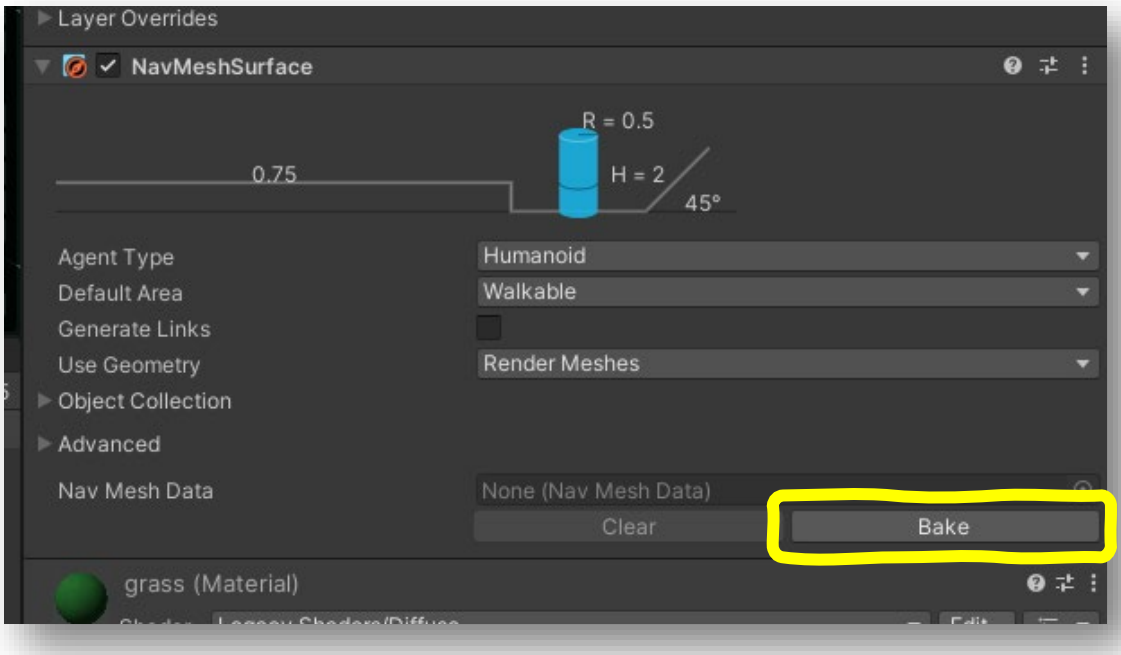


8 In your Hierarchy select the Ground Object. Add the **NavMeshSurface** component.

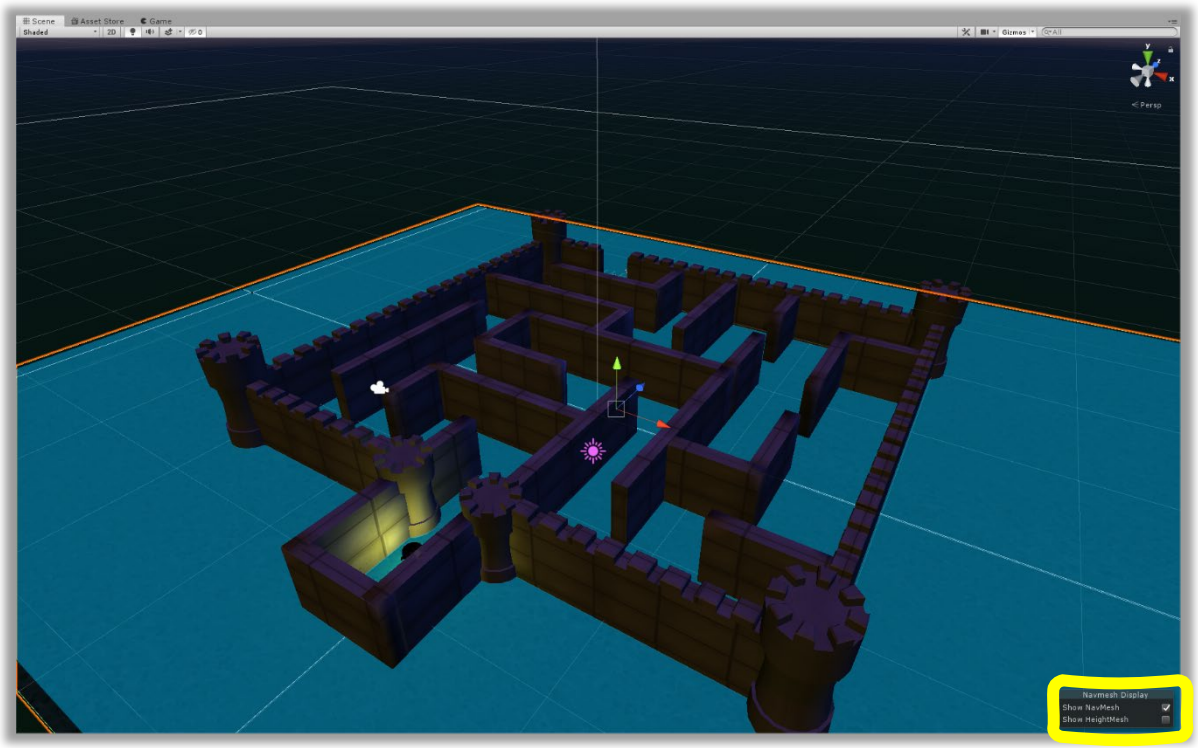
By default the **Navigation Area** is set to **Walkable**, leave that as it is.



9 At the bottom of the component menu click on Bake.

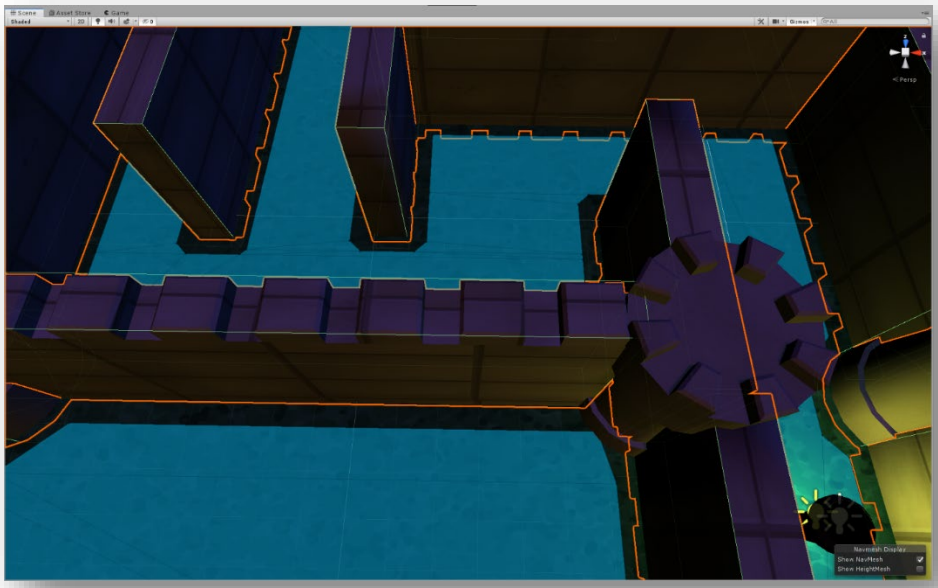


10 After Unity has completed “baking” the Ground will turn blue. This means that the NavMesh has been built unto the scene.



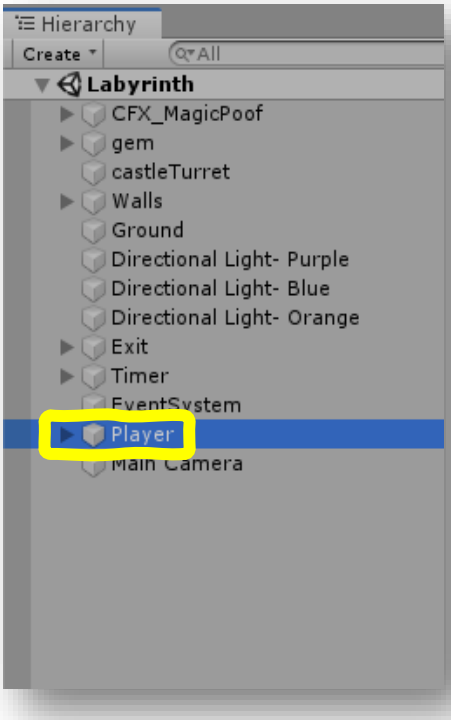
At the bottom right, you’ll see a menu to turn this blue highlighted area for the NavMesh on or off. See what it does when you uncheck it.

11 Look at your **Scene**, notice the gaps where the walls are in the ground NavMesh.



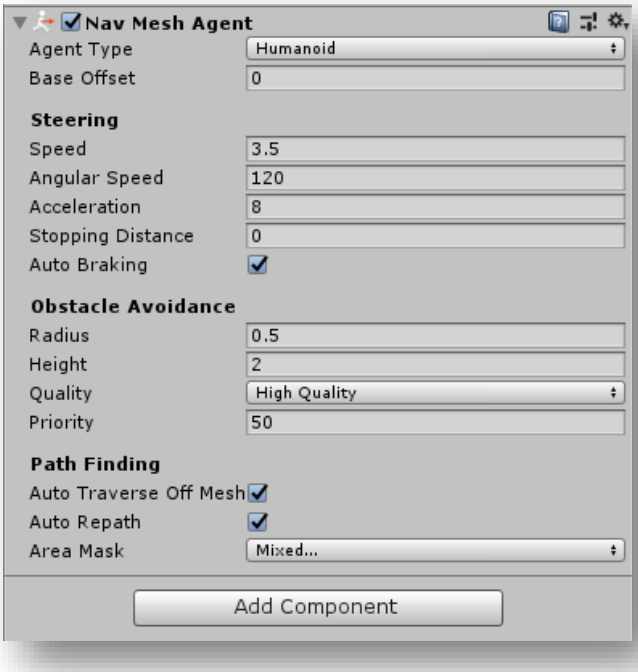
12 The scene now has the information where NavMeshAgents can and cannot walk. All we have left to do is assign an object as an agent. Can you guess which object will be our agent?

It's going to be our **Player** object! Select the Player in the Hierarchy.



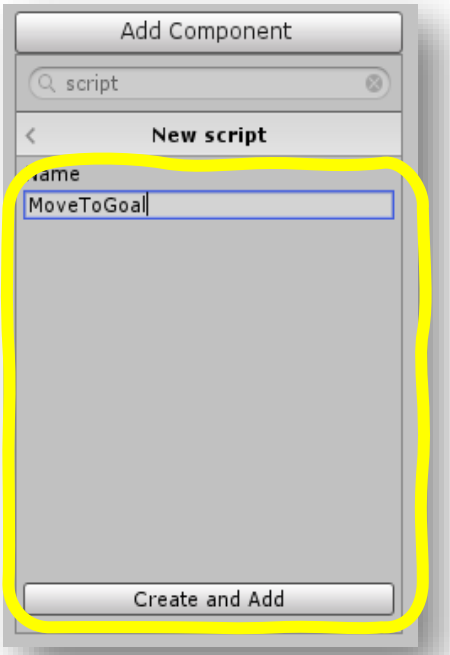
13 In the previous two parts the NavMesh was not a component, but the **NavMeshAgent** is. In the **Inspector** we want to click **Add Component** and type **NavMeshAgent**.

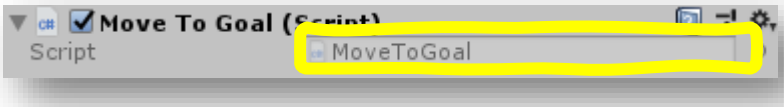
14 For now the properties won't be changed.



15 While the NavMeshAgent does a lot for us,we still need to provide our Player with more information. With the Player still selectd in the

Hierarchy, go to **Inspector** and click **Add Component**. Type in **New Script** and call it **MoveToGoal,** because that is exactly what it's going to tell the Player to do!
Open up the **MoveToGoal** script by double clicking on it.





16 At the top of every script, we've been telling Unity what we need from it's library of functions and capabilities. We need to add `Using UnityEngine.AI;`

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.AI;
5
```

17 We are going to need to initialize three variables. Type `public Transform goal;` as this will help the **Player** know it's target destination. Secondly type `private Animator animator;` so we can give Codey their running animation. Lastly type `private NavMeshAgent agent;` this way we set the agent itself.

```
6   public class MoveToGoal : MonoBehaviour
7   {
8       public Transform goal;
9       private Animator animator;
10      private NavMeshAgent agent;
11
```

In the screenshot above the `Start()` and `Update()` functions have been deleted, but will be added later. You can decided to keep them or delete them.

18 Let's work on our Start() function. We are going to assign the animator as `animator = GetComponentInChildren<Animator>();` so we can get the the Animator component from Codey.

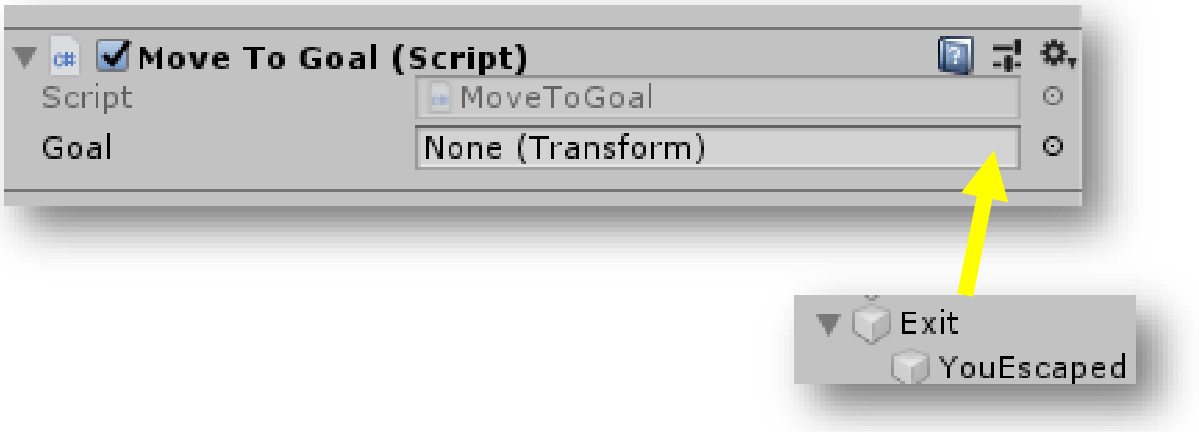
Next, type `agent = GetComponent<NavMeshAgent>();` as we want to store the NavMeshAgent components to agent.

Right after the code you just wrote type, `agent.destination = goal.position;` this way our agent will know our goal's position.

```
12      private void Start()
13      {
14          animator = GetComponentInChildren<Animator>();
15          agent = GetComponent<NavMeshAgent>();
16          agent.destination = goal.position;
17      }
```

Wait a minute, we have not assigned our goal so how will the agent know the goal position? **Save** your script and return to Unity.

19 Since we set up our goal as a public variable, we can assign it's value in the **Inspector**. Make sure you are still in the **Player object** and find the **MoveToGoal** script. Next to Goal, drag the **Exit** game object.



The agent variable will now know what the goal position is and where it has to move to!

20 **Play** your game and see what happens. Codey is able to find his way to the Exit with no help or input!

21 Here we learn about the first and most important NavMeshAgent built-in properties **destination**.

Setting the destination will tell the NavMeshAgent where it needs to go.

This is why we set destination to our goal variable, **Exit**.

You can notice that Codey doesn't go through walls, this is because earlier we **Baked** the **Walkable** and **Non-Walkable** surfaces of the scene and it did all the work for us.

22 So far Codey will just float to the goal with no animations. Let's fix that!

In the original Find the Exit, the **player input** would trigger his animations. Since the player doesn't give the game input anymore, we need a new condition to trigger animations.

We will be doing this by another important property of the NavMeshAgent, the Boolean property `hasPath`.

As long as the NavMeshAgent has a **path** to their set destination this willl return `true`.

23 When was the last time that you used the **Animator**? We simply need to use `SetBool` based on whether or not `hasPath` is returning `true` or `false`.
If the NavMeshAgent has a path to it's destination the `"isRunning"` animation is set to `true`. Otherwise, if the NavMeshAgent does not have a path to it's destination, the `"isRunning"` animation is set to `false`.

Type the following in the `Update()` function:

```
19      private void Update()
20      {
21          if (agent.hasPath)
22          {
23              animator.SetBool("isRunning", true);
24          }
25          else
26          {
27              animator.SetBool("isRunning", false);
28          }
29      }
30  }
```

24 **Save** your script and **playtest** your game. Codey will now run properly all the way to the exit without any input!

If everything works properly let's **submit** your game before moving on.

25 Now that you know the basics of NavMesh, let's get a little creative with the next Prove Yourself activity.