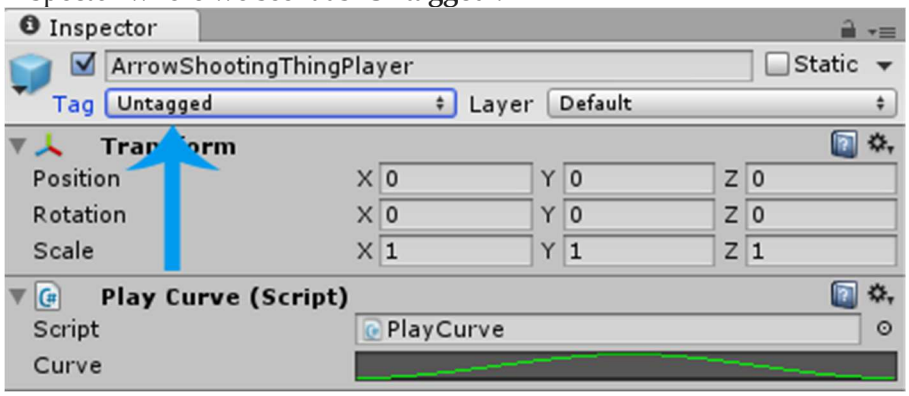
Unity Real Time Strategy (RTS) Game – Part 2

Tagging

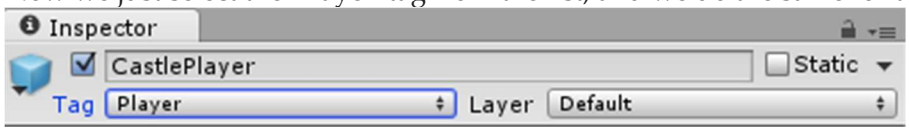
The Player and the Npc are enemies. Their units need some way to find all enemy units, so they can decide which one to attack.

There are all kinds of different ways to do this. For example, we could create a list of type ArrowShootingThingPlayer and another list of type ArrowShootingThingNpc and then add & remove the units all the time.

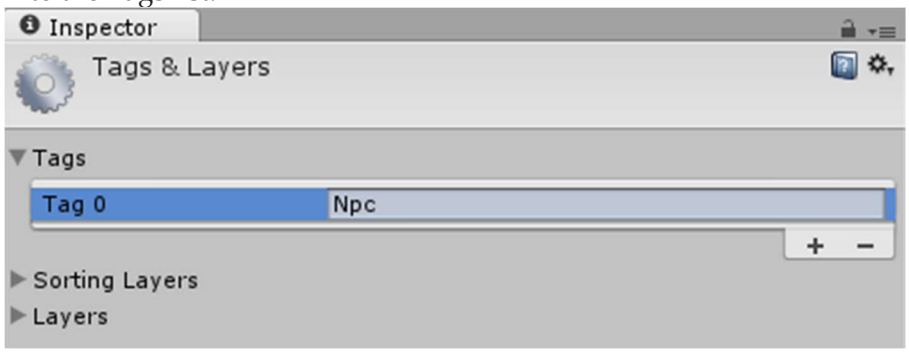
However, the easiest way is to use Unity's Tag system, which takes care of this already. The tag system is really easy to use. Let's select our ArrowShootingThingPlayer prefab and take a look in the Inspector where we see it as '**Untagged**':



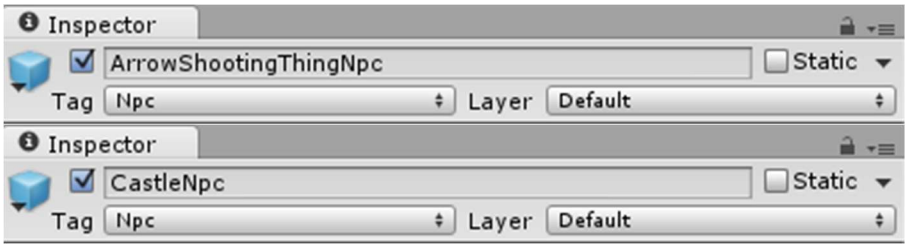
Now we just select the 'Player' tag from the list, and we do the same for the CastlePlayer prefab:



We can also also add a new Tag by selecting 'Add Tag...' from the list and then inserting a new tag into the Tags list:



After adding the 'Npc' tag, we select our ArrowShootingThingNpc and CastleNpc prefabs and then assign the 'Npc' tag to both of them:



Tagging the prefabs gives us the ability to find all Player *(or Npc)* prefabs in the game world by simply using **FindGameObjectsWithTag**:

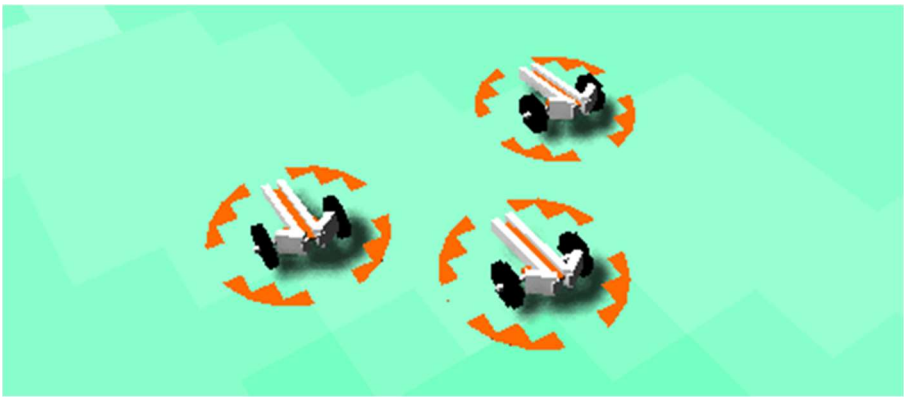
```
GameObject[] units = GameObject.FindGameObjectsWithTag("Player");
```

This will come in handy very soon in or scripts.

Selections

The Concept

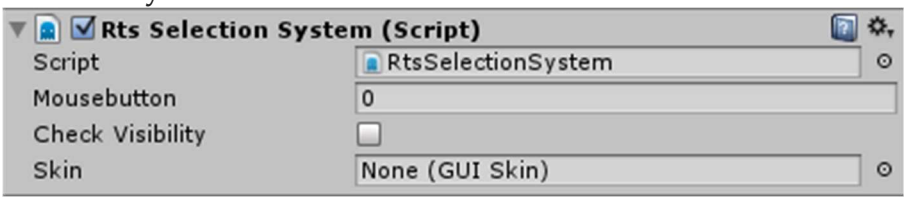
The player needs a way to select units and then move them around the map. The Script should make a selection circle visible for all selected units like this:



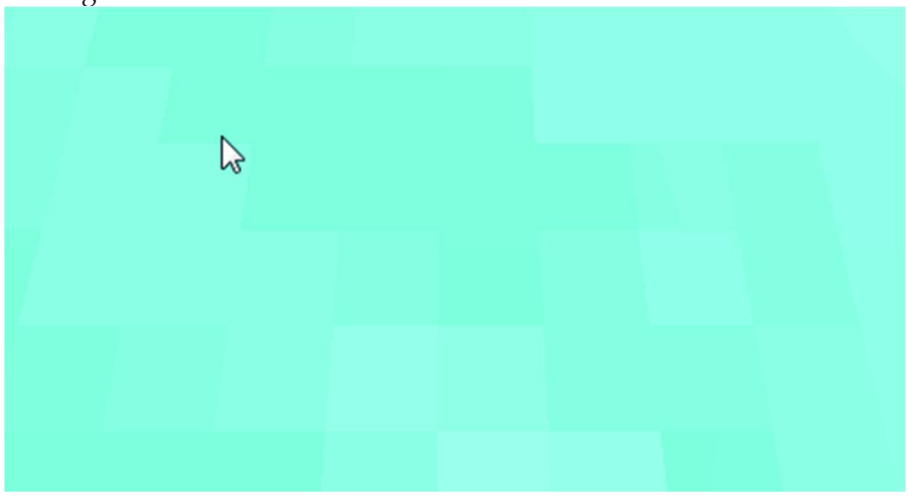
Using the RTS Selection System

Unity doesn't come with a rectangle selection system by default and it requires a decent amount of Unity knowledge to get it to work properly. Just use the RTS Selection System script already provided in the scripts folder for now.

Afterwards we will select our **Main Camera** and then click on **Add Component->Scripts->RTS Selection System**:



If we press **Play** then we can already press and drag the left mouse button to see the selection rectangle:



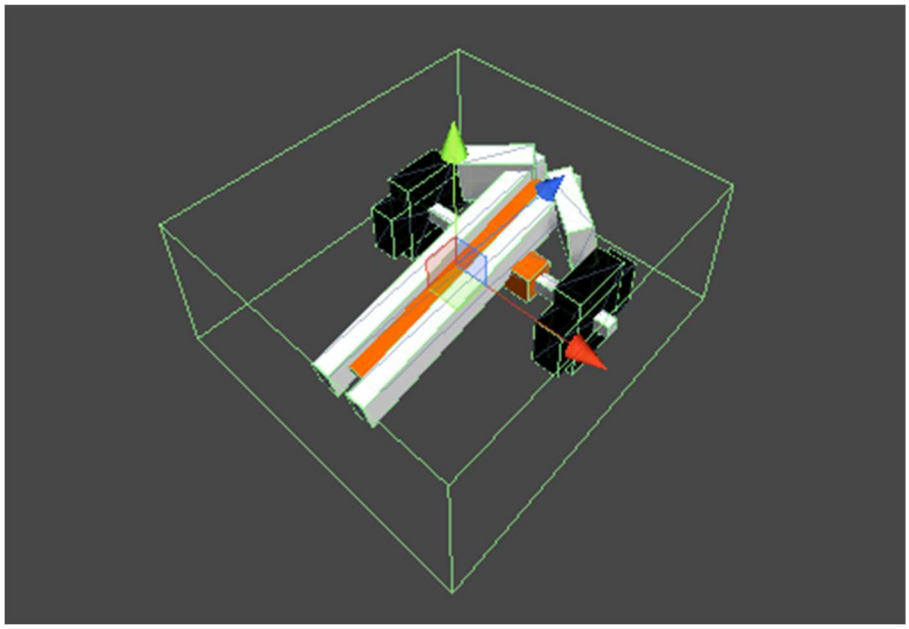
And as soon as we let go of the mouse button, our RTS Selection System will automatically call **OnSelect** in each unit that is inside the rectangle, as well as **OnDeselect** for each unit that was previously selected and is not selected anymore.

The Collider

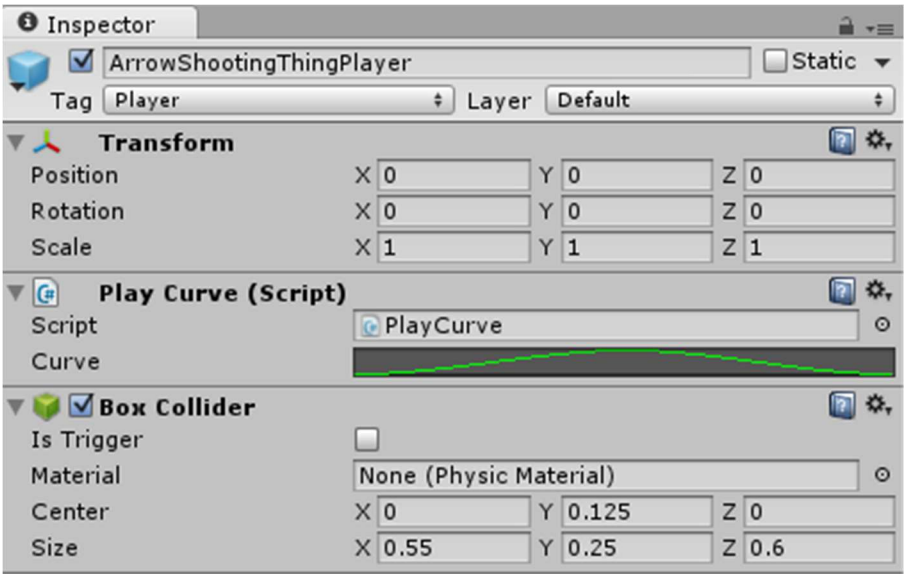
OnSelect and **OnDeselect** will only be called on GameObjects that have colliders attached to them, so let's take care of that. We will add a Box Collider to the ArrowShootingThingPlayer prefab by:

- Dragging it into the Scene
- Adding the collider
- Adjusting the collider so it looks properly
- Pressing apply
- Deleting it from the Scene again

Here is how it looks in the Scene:



And here is the current Prefab:



Adding the Selection Circle to the Prefabs

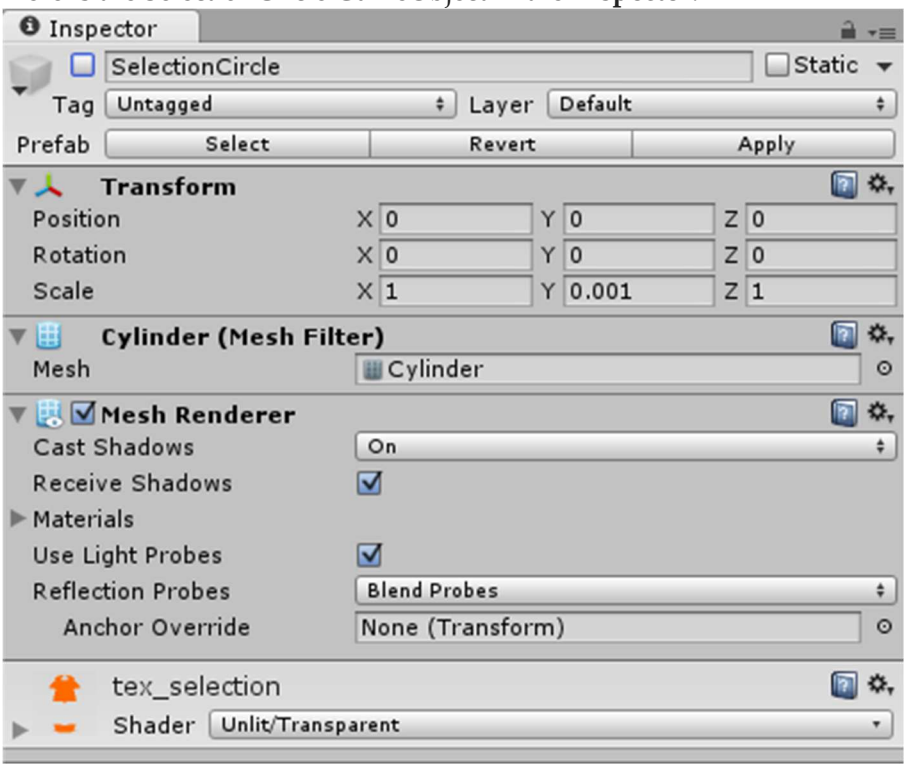
We will need some kind of visual feedback to let the player know that a unit is selected. A selection circle is the perfect choice, so let's make one. We start by dragging our ArrowShootingThingPlayer unit into the game world so we can modify it.

We can add a cylinder to it by right clicking the **ArrowShootingThingPlayer** object and selecting **3D Object->Cylinder**. We will name it **SelectionCircle**:

Now we take a look in the Inspector and:

- Remove the collider
- Change the Y Scale to 0.001
- Add a little 8x8 pixel texture from our **Assets** folder by dragging it onto the object in the Hierarchy: **tex_selection**
- In the Inspector, change the Texture Shader to Unlit/Transparent
- Disable the GameObject so that it's not shown by default
  Note: we can disable it with the checkbox at the very top, next to the name.

Here is the SelectionCircle GameObject in the **Inspector**:



Using OnSelect and OnDeselect

Almost done, now we just have to make use of the previously mentioned **OnSelect** and **OnDeselect** functions that will be automatically called by our **RTS Selection System**.

Let's select the ArrowShootingThingPlayer object in the **Hierarchy** and then click on **Add Component->New Script**. We will name it **PlayerSelection**, select **CSharp** as the language, move it into our **Scripts** folder and then open it:

```
using UnityEngine;
using System.Collections;

public class PlayerSelection : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

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

    }
}
```

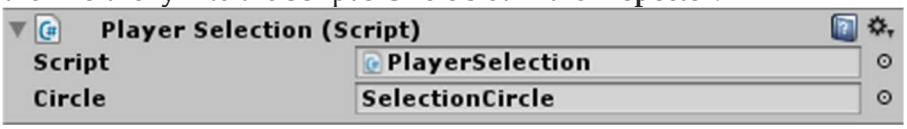
We won't need the **Start** or the **Update** function, so let's remove both of them. Instead we will use **OnSelect** and **OnDeselect**:

```
using UnityEngine;
using System.Collections;

public class PlayerSelection : MonoBehaviour {

    // OnSelect is called by the RTS Selection System
    void OnSelect() {

    }

    // OnDeselect is called by the RTS Selection System
    void OnDeselect() {

    }
}
```

We will use those functions to enable and disable our selection circle:

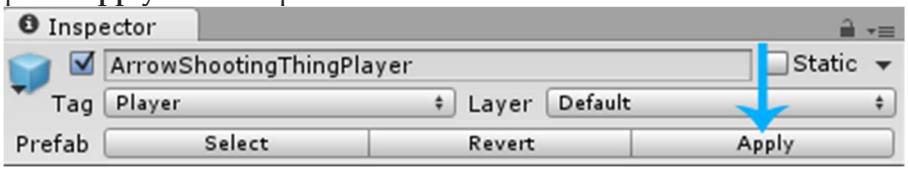
```
using UnityEngine;
using System.Collections;

public class PlayerSelection : MonoBehaviour {
    // Selection Circle
    public GameObject circle;

    // OnSelect is called by the RTS Selection System
    void OnSelect() {
        circle.SetActive(true);
    }

    // OnDeselect is called by the RTS Selection System
    void OnDeselect() {
        circle.SetActive(false);
    }
}
```

After saving the Script, we can drag the the object's **SelectionCircle** GameObject from the **Hierarchy** into the Script's **Circle** slot in the **Inspector**:



And that's all. If we press **Play** then we can now select and deselect our ArrowShootingThingPlayer unit:

To save the whole thing, we select ArrowShootingThingPlayer in the Hierarchy and then press **Apply** in the Inspector:



This saves the modifications to the Prefab. Now we can delete it from the Hierarchy again.

Player Unit Movement

Now that we have selections, we can move the Player's units around very easily. Let's select our **ArrowShootingThingPlayer** prefab, click on **Add Component->New Script**, name it **MoveByPlayer**, select **CSharp** as the language, move it into our Scripts folder and then open it:

```
using UnityEngine;
using System.Collections;

public class MoveByPlayer : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

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

    }
}
```

We can remove the **Start** function because we won't need it.
We will need a way to find out if the unit is currently select. It is selected whenever the selection circle is enabled, so let's add a public **GameObject** variable where we can drag the SelectionCircle into later on:

```
using UnityEngine;
using System.Collections;

public class MoveByPlayer : MonoBehaviour {
    // Selection Circle
    public GameObject circle;

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

    }
}
```

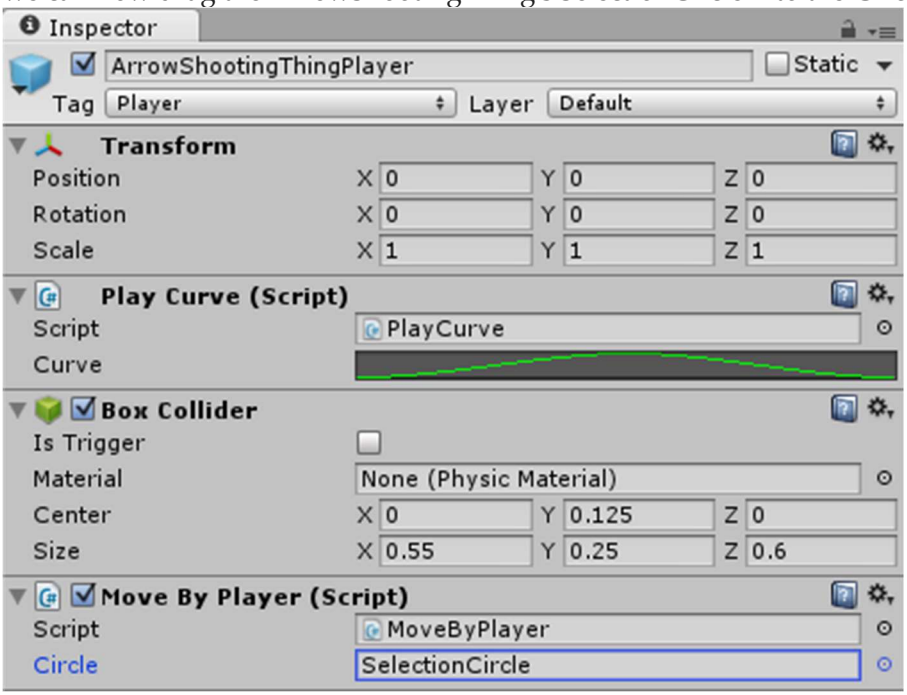
We want the unit to move whenever the player does a right mouse click, and of course only if the unit is selected *(or in other words: if the circle is enabled)*:

```
void Update () {
    // Rightclicked while selected?
    if (Input.GetMouseButtonDown(1) && circle.activeSelf) {
        // ToDo move...
    }
}
```

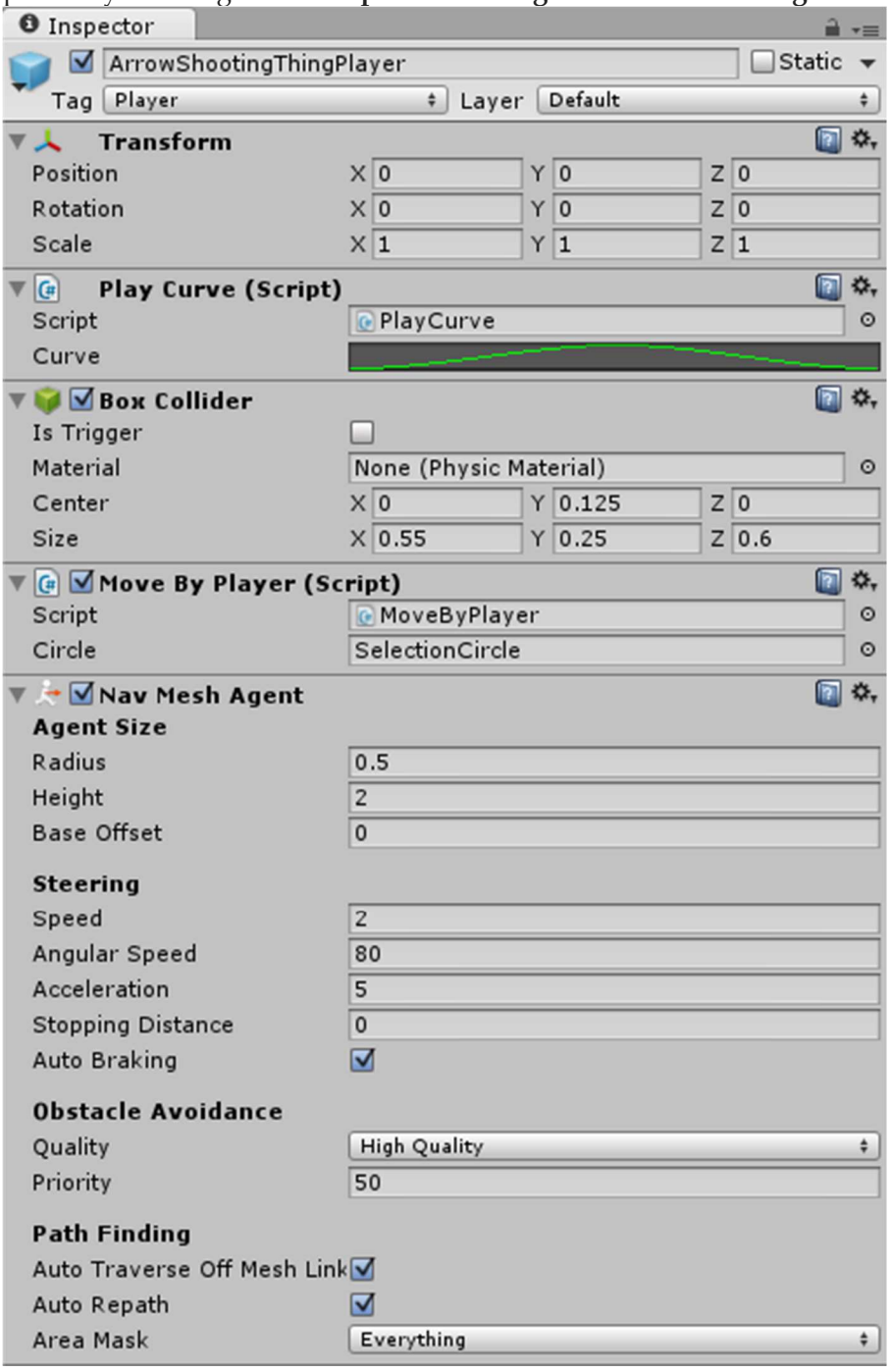
*Note: a GameObject's **activeSelf** property returns true if it is enabled, and false otherwise.*
Now we still have to find out where the player clicked at. As we did before, we will do this via **ScreenPointToRay** and **Physics.Raycast**:

```
void Update () {
    // Rightclicked while selected?
    if (Input.GetMouseButtonDown(1) && circle.activeSelf) {
        // Find out where the user clicked in the 3D world
        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;
        if (Physics.Raycast(ray, out hit)) {
            // ToDo move to hit.point
        }
    }
}
```

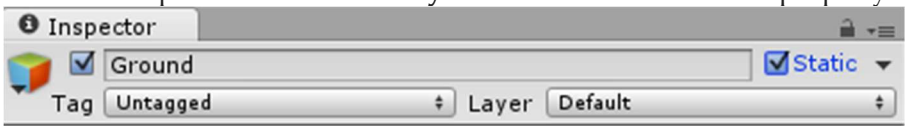
Before we continue with the movement, let's save the Script and take a look at the **Inspector** where we can now drag the ArrowShootingThing's SelectionCircle into the **Circle** slot:



Alright now we still have to find a way to move the whole thing to the wished position. What sounds complicated is really easy thanks to Unity's built in navigation system. This system requires each moving thing to have a **NavMeshAgent**, so let's add one to our ArrowShootingThingPlayer prefab by selecting **Add Component**->**Navigation**->**Nav Mesh Agent** in the **Inspector**:

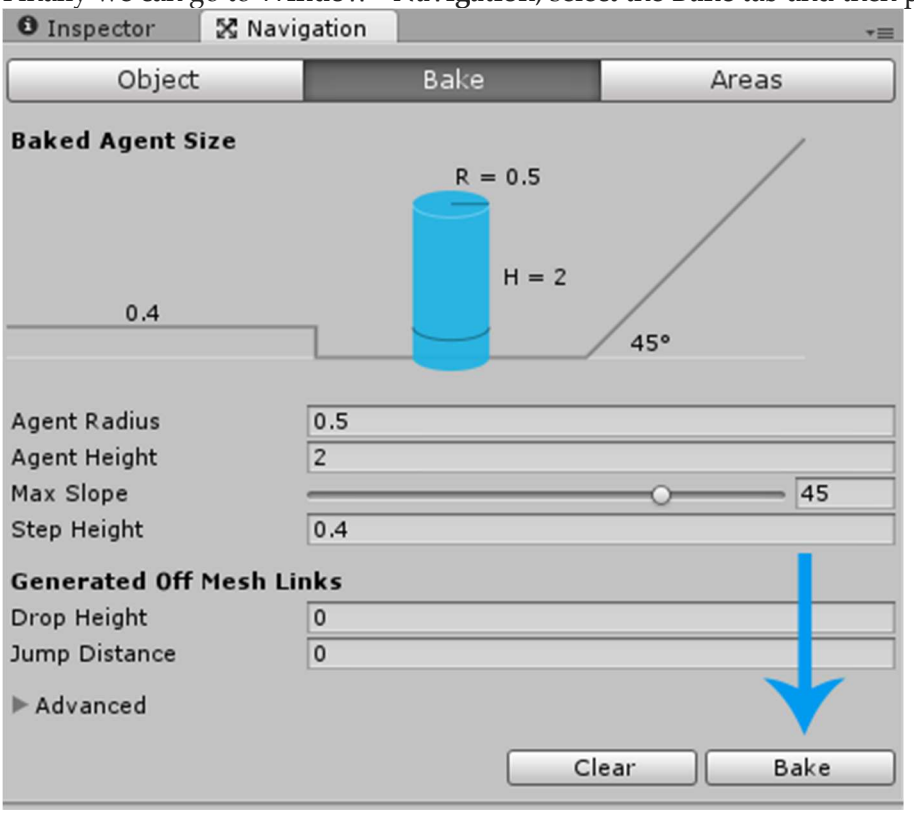


Now we still have to let Unity know which things in our game world can be walked on. Let's select the **Ground** plane in our **Hierarchy** and then enable the **Static** property in the **Inspector**:

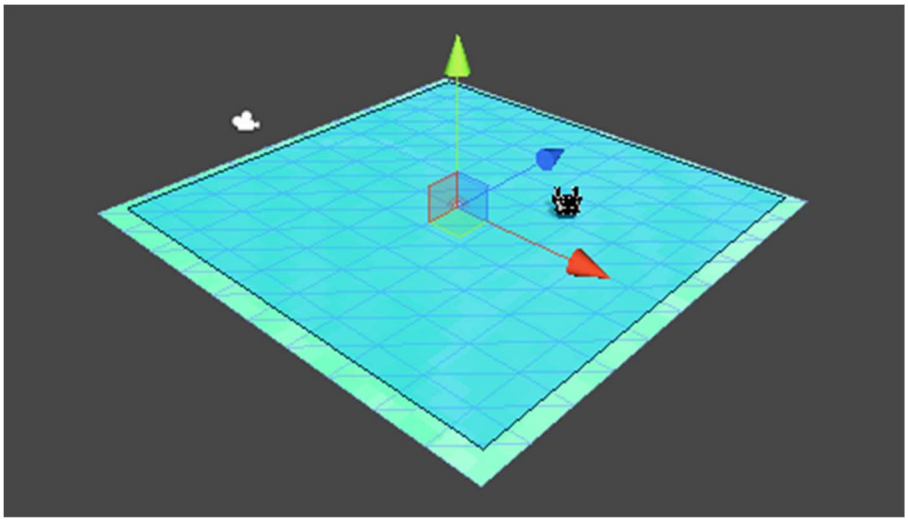


Note: this pretty much tells Unity that it can be walked on.

Finally we can go to **Window->Navigation**, select the **Bake** tab and then press **Bake**:



Afterwards we can see the NavMesh in the Scene view:



Note: this is pretty much the walkable area. All NavMeshAgents will be able to navigate to any point in that area.

Finally we can make our units walk by setting the Nav Mesh Agent's **destination** property from within our MoveByPlayer script like so:

```
GetComponent<NavMeshAgent>().destination = ...
```

Here is our final script:

```
using UnityEngine;
using System.Collections;

public class MoveByPlayer : MonoBehaviour {
    // Selection Circle
    public GameObject circle;

    // Update is called once per frame
    void Update () {
        // Rightclicked while selected?
        if (Input.GetMouseButtonDown(1) && circle.activeSelf) {
            // Find out where the user clicked in the 3D world
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            RaycastHit hit;
            if (Physics.Raycast(ray, out hit))
                GetComponent<NavMeshAgent>().destination = hit.point;
        }
    }
}
```

If we press **Play** then we can see it in action by selecting a unit and then right clicking somewhere on the Ground.

Note: feel free to play around with the Nav Mesh Agent's properties in order to get the perfect movement speed for the units.