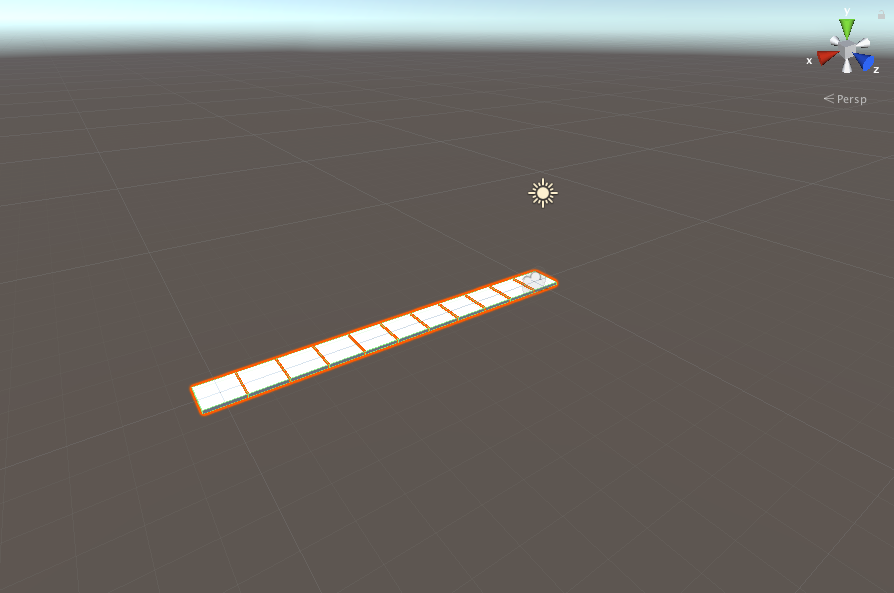
**Isometric character movement**

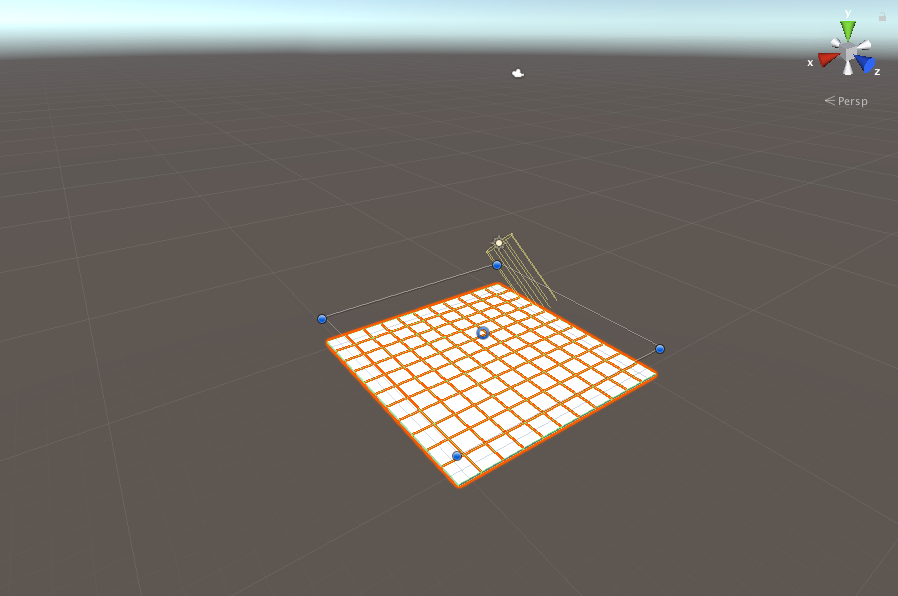
This tutorial will show you how to create a character controller within a 3D isometric grid view.

**1. Setup the playfield and isometric camera**

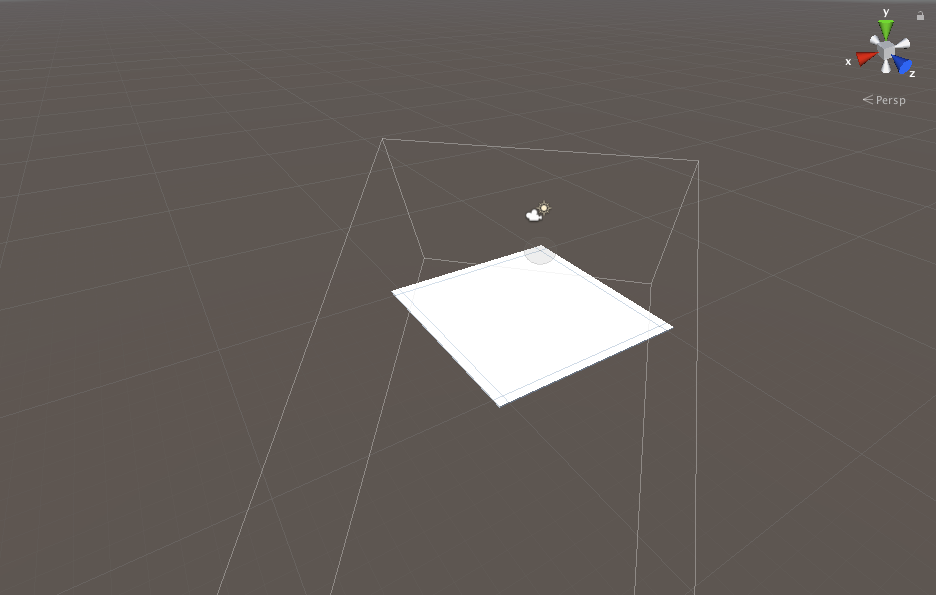
First, create a cube and reset the transform component. Reduce the the Y scale to create a platform and duplicate it. You can then move each duplicated platform once along the X axis to create column.



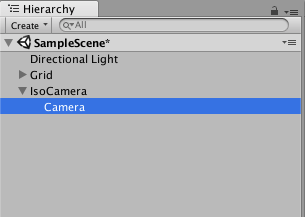
Duplicate this column and move it along the Z axis to create a grid (for this example, the grid is 10 x10)



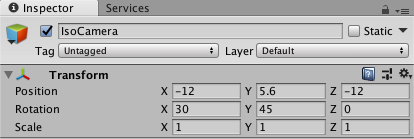
Next, we need to create the isometric camera. Delete the default main camera and create an empty GameObject and rename it to something like “Isocamera”. I’ve set the X rotation to 30 and Y rotation to 45 to create a downward facing angle over the playing field.

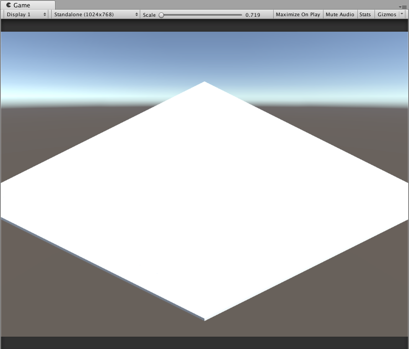


Then, create a new camera and set this as the child your “Isocamera”. Make sure to set the tag to “Main Camera”. Set the projection mode to Orthographic and adjust the position of the Isocamera to fit the playing field on screen.

****

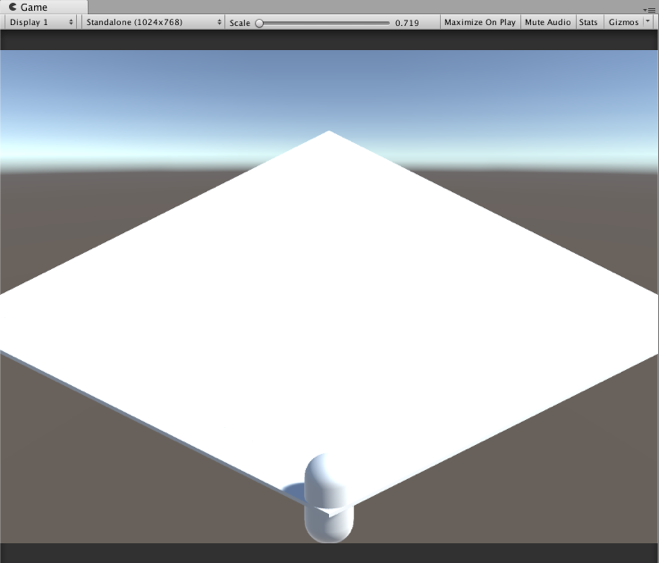
I found these transform positions the most beneficial for fitting the playing field on screen.

****

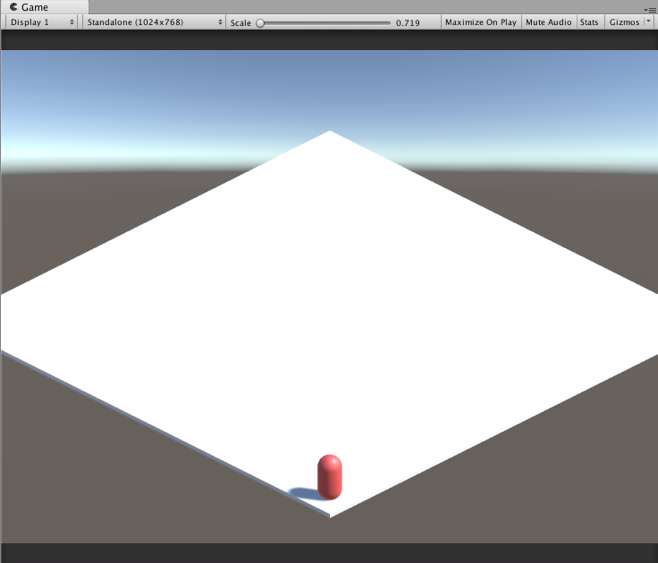
****

**2. Setup the character**

Next, we’re going to create our character. Create a new 3D capsule, rename this in the Hierachy to “Player” and reset the transform.

****

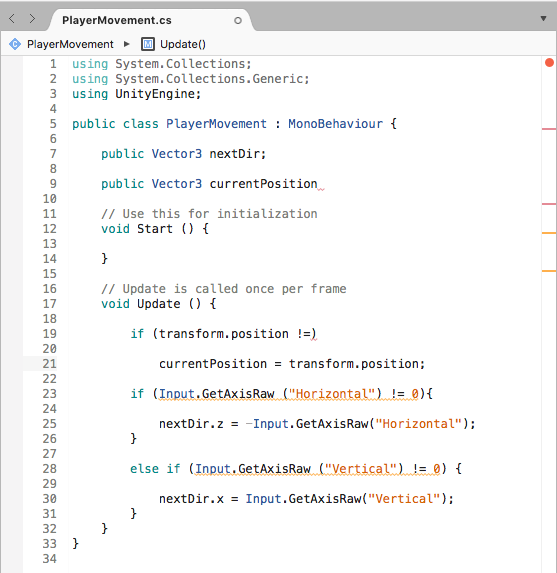
Adjust all the scale values as well as the Y transform to .5 so the player sits above the grid. Create a new coloured material and drag and drop this on to your player in the scene view. For this example, I’ve chosen a red colour for the material so the player stands out.

****

**3. Create the movement script**

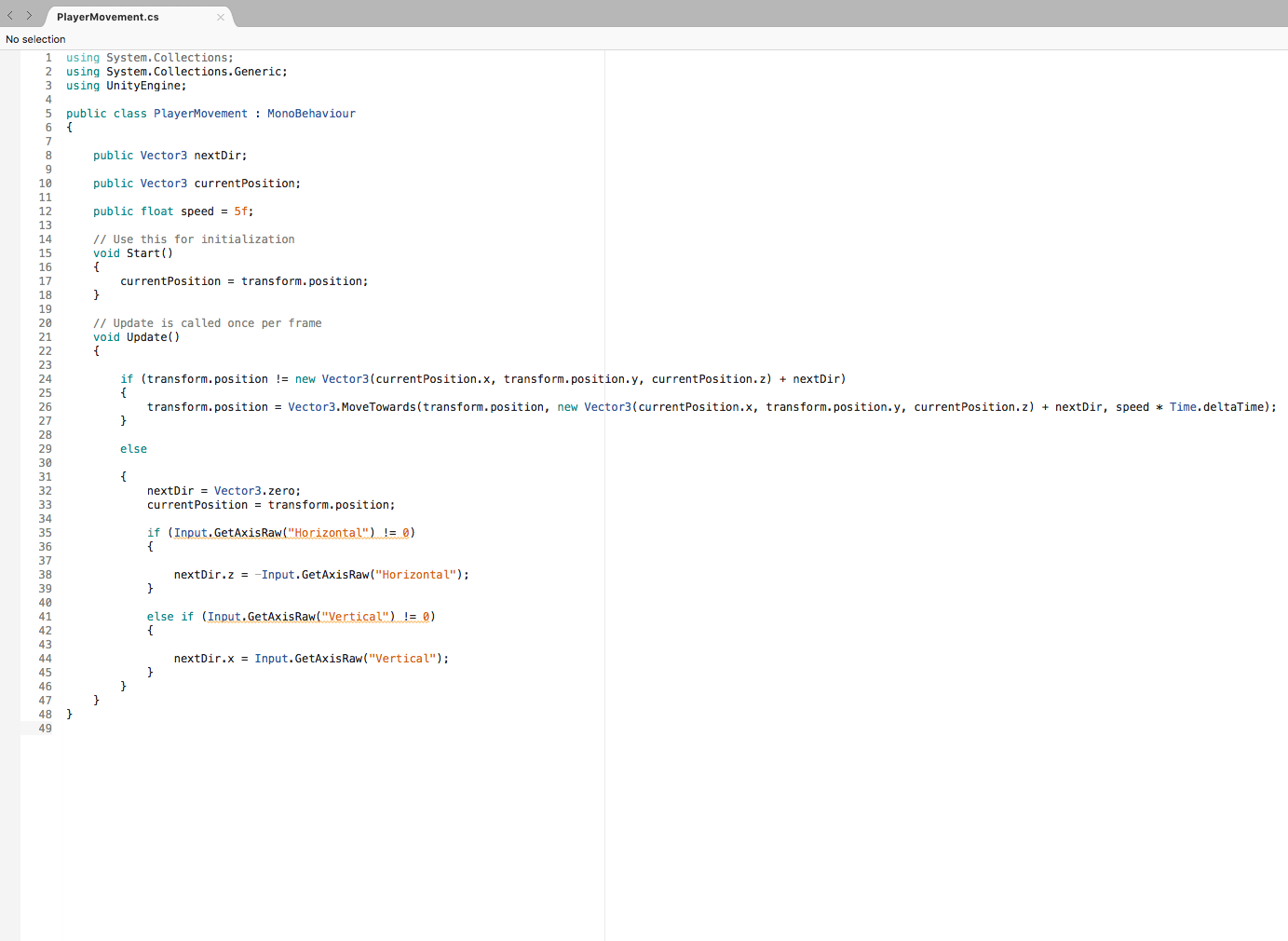
To create the PlayerMovement script, we’re going to rely on raw axis inputs and detection of the players current and potential transform positions to control the player, moving them in a single step manner.

Create a new C# script called PlayerMovement, create a public Vector 3 called nextDir and currentPosition. In Void Update, we can use if statements to determine the current position of the player and the next direction to be determined using GetAxisRaw.



We’ll then create a new public float called speed and set this to 5f. We’ll use this in a new Vector 3 when defining the transform position to determine character speed.

By defining the currentPosition of X and Y alongside the transform position of y, we ensure that only the X and Y axis’ are being checked for potential movement. We’ll also use MoveToward to utilise line 21. We then need to set the nextdir.z to 0 in the else condition to ensure that the player doesn’t move continuously after an input is made.



Add this script to your player and you can now move around the grid space. In the next tutorial, I’ll create collision detection for props around the space and board boundaries as well as create event triggers.