**Tutorials**

**Movement**

* We want the player to move along a surface that is void of friction. This is so our player object is able to glide along the stage fluently with no bumps in the road. So begin by creating a physics material and setting the friction values to 0. Apply this to the ground plane.
* In a C# script, we use the line rb.AddForce(x,y,z \* time.deltatime); to cause the player object to move in any direction we assign a value to. To make this easier for us to edit, I’ll create a variable and attach this to the x,y or z value. We do this to be able to edit the force of our movement in the Unity editor rather than having to open up the script and save it each time. This will save time and make things a lot easier going forward.
* A screen shot of a computer code

  Description automatically generated
* Let’s dissect the line ‘public float forwardForce = 2000f;’. The word ‘public’ can be substituted for the word ‘private’, this changes whether or not we can see this value in the Unity editor. Because that is a main point to this variable we will keep it public. Next is ‘float’, this just refers to a decimal number, ‘forwardForce’ is the name applied to moving forward and ‘2000f’ is the default value of the forward force, which is able to be changed in the Unity editor.
* A computer screen shot of a program

  Description automatically generated
* Here we assign controls. For moving left and right it is essentially the same thing as earlier, instead affecting the x axis. We attach a variable to it also. We use velocity change to create a gradual transition between moving left and right, and we use ‘-‘ to go the opposite direction.

**Camera**

* Next we create a script to cause the camera to follow the player object.
* A screen shot of a computer program

  Description automatically generated
* Here, the line between the curly brackets is self explanatory. The camera position is set to equal the player position at the distance it is placed at in the Unity editor. We add an ‘offset’ variable, Vector3, which stores three float values, the x ,y and z values. This allows us to make the camera follow behind the player object rather than be set to the centre of the player object.

**Collision**

* Now we will create a script to implement collision between the player object and obstacles in the game. Collision will work as long as the objects colliding have a rigid body component and a collider component.
* A computer screen with text and symbols

  Description automatically generated
* You want to create a tag in Unity and set that tag to every obstacle in the environment, this script will affect any object assigned to that tag. This will save time as you don’t have to individually apply the script to the objects which can help when you have lots of them.
* By setting movement to false, our player object will cease to apply the forward force and the game will come to a halt, ensuring that you have come in contact with an obstacle.

**UI Elements**

* Now we need something on the screen to indicate the basic score that correlates to the distance travelled by the player object. To do this we create a text game object in the canvas and assign a script to it.
* A computer screen shot of a program

  Description automatically generated
* You create a variable indicative of the style of score display that you want. We use the player object’s position as of the Z axis to tell Unity to update the score based on it. The further we travel forward the higher the number becomes. ‘ToString’ converts the number value to text which is displayed in Unity.

**Pickups**

* Lastly, I’ll add some collectibles into the game. I add in some sphere game objects and give them each a rigid body component and box collider.
* A screen shot of a computer code

  Description automatically generated
* We create variables that act as the score in this script. This script logs the score for us based on another script which will notice when our player object collides with a collectible. This line acts as the UI element which will show us what our score is based on the next script.
* A screen shot of a computer code

  Description automatically generated
* We use the line ‘Destroy’ to indicate the collectible that gets removed from the game once we pick it up. This script acts in conjunction with the previous script where this script notifies Unity when we pick up an item, and the other script will convert that information into a UI element onscreen.
* I’ve set each collectible to increase the score by 10.