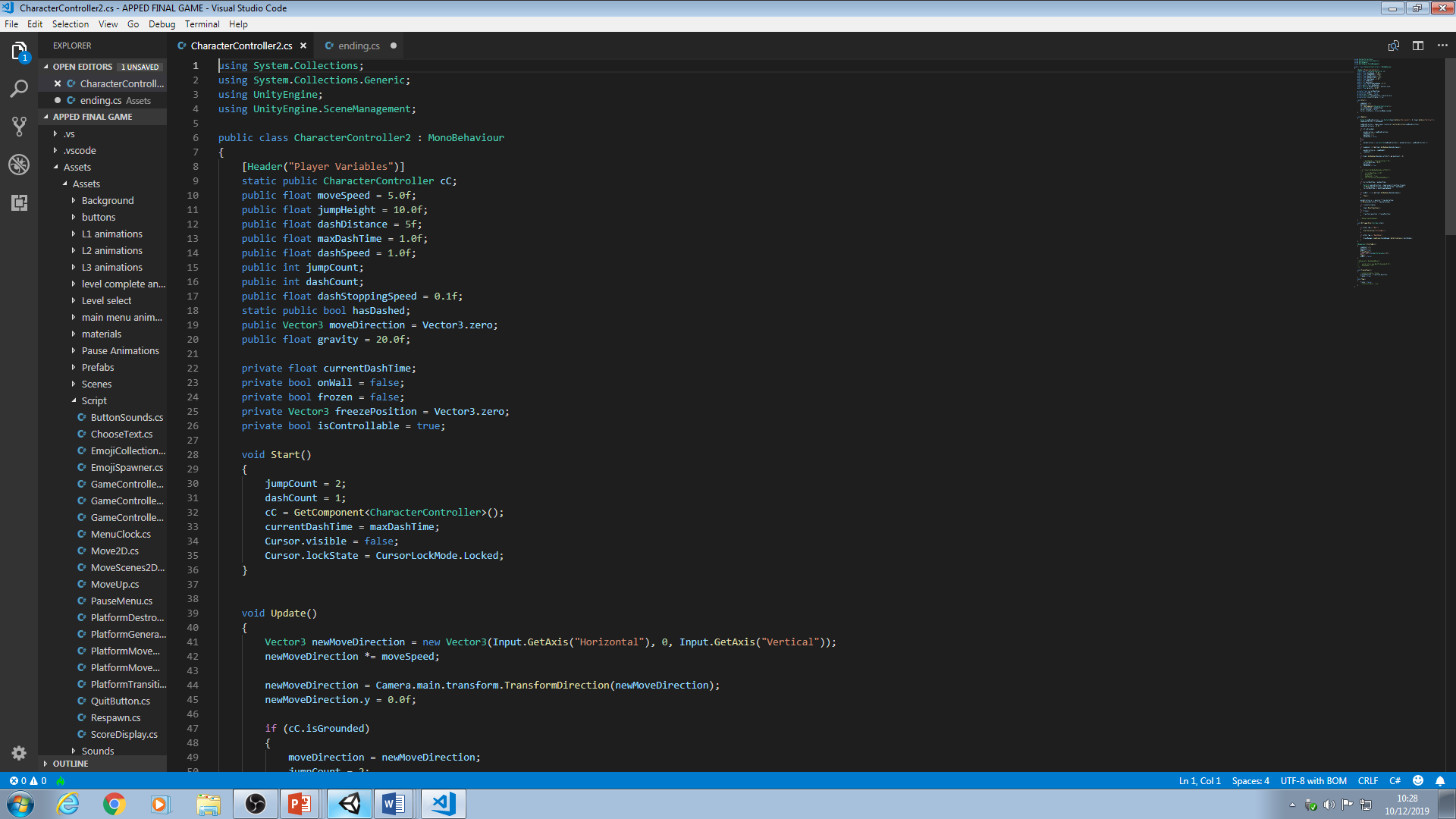
Programming Tutorial 1: Player Controller

For the 1st tutorial I will show you how to program and successfully apply a Player Controller that is designed to be used for an obstacle course style game, although this Player Controller can easily be used for any other game. For the Player Controller I decided to use the Unity Character Controller component instead of the Rigidbody System as Unitys physics can sometimes be wonky and is a bit floaty and unprecise, not something you want in an obstacle course game.

1. Create a capsule to be used for the player and apply a Character Controller component as well as a capsule collider. Create a new script and place it on the player as well, call it Player Controller.
2. First, create all the variables that you see in the image, their use will be explained when they are used.
3. In the start function we will set some values. We will set the jumpCount to 2 and dashCount to 1, when we code the jump and dash feature this will stop you being able to infinitely dash and jump. We also need to reference the CharacterController so we can actually apply motion to our character so type cC = GetComponent<CharacterController>(); Also type Cursor.visible = false; and Cursor.lockstate = CursorLockMode.Locked; this lock the mouse and disables it
4. In the update function is where we put code that needs to repeat such as our movement code. Type Vector3 newMoveDirection = new Vector3(Input.GetAxis(“Horizontal”), 0, Input.GetAxis(“Vertical”)); This creates a new vector that gets the input values of WASD. Then type newMoveDirection \*= moveSpeed; This multiplies the values from WASD input by your movespeed. Next, type newMoveDirection = Camera.main.transform.TransformDirection(newMoveDirection); This sets the movement to wherever the Camera is facing and tells you to transform in that direction, so you walk wherever you face.
5. Next, we need to apply these values to the CharacterController so the player actually moves. We need to check if the player is grounded so create an IF statement and say cC.isGrounded then within that IF statement set the moveDirection = newMoveDirection; which will tell the character controller to move based on the values we have given newMoveDirection. Also jumpCount and dashCount need to be set to 2 and 1 again as every time the player jumps and lands they need to be set back otherwise the player will only be able to double jump and dash once. Next we need to create an ELSE statement and type moveDirection = new Vector3(newMoveDirection.x, newMoveDirection.y, newMoveDirection.z); This controls the player while he is not grounded and allows you to move around in the air.
6. Next, create another IF statement and type jumpCount > 0 && Input.GetKeyDown(KeyCode.Space) which checks if the jumpCount is over one AND the space key is pressed, it will then execute the jump code and this is why the jumpCount needs to be reset as you won’t fulfil the IF condition if it’s 0. Next, type moveDirection.y = jumpHeight; which sets Y value of move to the jump height. Also type jumpCount--; so every time you jump it takes one away, allowing you to double jump.
7. After this, in the update method type moveDirection.y -= gravity \* Time.deltaTIme; which change the Y value of the player by the gravity variable to simulate gravity. Next type cC.Move(moveDirection \* Time.deltaTime) this is called every frame and will apply the moveDirection value constantly to the Character Controller move function
8. Next, we’ll code the dash and first we need to make an IF statement again then type Input.GetKeyDown(KeyCode.LeftShift) && dashCount > 0. Then type currentDashTime = 0.0f which will fulfil another IF statement and activate it that we’ll code in a minute. Don’t forget to type dashCount-- so you can’t dash infinitely.
9. Next, type currentDashTime < maxDashTime in an IF statement then inside the statement type Vector3 cameraDirection = Camera.main.tranform.forward; Which grabs the value of where the camera is facing. Then type cC.Move(cameraDirection \* Time.deltaTime \* dashSpeed); This moves the character controller in the direction of the camera which is then multiplied by the dashSpeed. Then type currentDashTIme += dashStoppingSpeed; which increases the current dash time by 0.1 until it reaches 1 which then stops the dash.
10. Next, were going to make the player be able to stick to walls. Create an IF statement with the bool frozen in, then in the statement type transform.position = freezePosition; which sets the transform of the player to the freezePosition variable which will be set when the player freezes. Next, create an OnTriggerEnter function and within that create another IF statement that checks the tag of the collided object to see if it’s a wall so type other.tag == “Wall” then within that statement type StartCoroutine(“StickToWall”);
11. Next, create a coroutine method by typing IEnumerator StickToWall(){} The name of the method must always be whatever you called it when you started the coroutine, in this case it is StickToWall. In the method the jumpCount and dashCount must be set to 2 and 1 again so when you jump from the wall you can jump and dash. Then type onWall = false; then FreezePlayer(); to activate that method. Next, type yield return new WaitForSeconds(1.5f); which tells compiler to wait 1.5 seconds before it executes the next piece of code which will make the player fall from the wall, meaning the player can stick to walls for 1.5 seconds. Then type Thaw(); which is another method we will create soon and finally type onWall = false.
12. Next, we need to create a method for freezing the player so type void FreezePlayer(){} and in the method type freezePosition = transform.position; which sets the freezePosition variable to the position of wherever the player gets frozen then type frozen = true which will activate the previous if statement earlier and change the transform.positon to the freeze position keeping it there while freeze is true.
13. Create another method called Thaw(){} and type frozen = false; this will allow the player to move again
14. The dashSpeed variable controls the dash distance. Set it to around 17 in the inspector to dash.

