Learning Journal

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Code | What I learned | Date |
| Character Movement | RigidBody2D | I learned the importance of using a RigidBody2d in addition to the box collider when it came to set up the character movement. I discovered that the reason that a Rigid Body is used is because it interacts with objects through the physics engine, meaning it can receive forces to make objects move in a realistic way. A box collider 2D is also still needed so that the player can interact with the environment which is recognised upon box collision in scripting. | 04/03/2020 |
| Dialogue management and animations | Private Queue <string> sentences; | I found the dialogue task the hardest to understand out of everything I have learned through this project. This was because the dialogue manager not only had to sort out the queuing system for the dialogue, but also because it had to arrange the order of the animations for the dialogue box to appear and disappear (in the trigger script as well as the dialogue manager). I did learn how to string sentences together and how by using a private queue system the script can privately reference the systems dialogue through the string and set up a public organised system for the order of running the dialogue. | 04/03/2020 |
| Damage Player | Name tags (gameObject.tag == (“insert tag name”)) | I learned during this project that using tags and referencing them in the script was the easiest way to set up a system of who could damage the player and who couldn’t. By creating a tag and adding it to multiple various shaped enemies meant I didn’t have to assign a new thing to the code each time, but instead I could simply tag the new object as an “Enemy” listed in the tag and it’ll be instantly playable, and could be easily kept track of as well. | 13/04/2020 |
| Jumping Mechanic | [SerializeField] private LayerMask platformsLayerMask; | The jumping mechanic was one of the more difficult parts of this project I discovered. What I did learn however was the use of layers and their purpose in the game. By creating groups by layers, the script uses this to determine if the player is grounded when jumping. By using a Ray Cast that reflects onto the ground, the script can measure how high the player has jumped as well as the velocity. The part I still struggle using however, is the coding maths in terms of what needs to be divided and how high the player must be able to jump and what direction, which was a big challenge when it came to coding. |  |
| Enemy Movement | RayCastHit2D | Using the Ray Cast was a new system to me during this process, but I found the process very straightforward to understand. In this instance, the Ray Cast was being used for enemy movement, where a direct ray will reflect down the Y axis to find the location of the platform using ground detection. If the AI was no longer able to hit the ground beneath it, (groundInfo.collider == false) then the AI will turn around and head towards the other direction. The only downside to this meant that my project had to consist of a lot of floating platforms, and you can see in my final project that there is a lot of gaps around the floor so the AI could patrol within their assigned area in places where I didn’t use platforms. | 27/04/2020 |
| Shooting Mechanic | rb.velocity = transform.right \* speed; | In making this project, I thought that scripting the shooting mechanic would be the most time-consuming part of this project, however I discovered it was more straightforward than I thought. I learned that using a RigidBody for the bullet, much like with the player, is the most effective way in terms of movement because of how it works with the physics engine. The scripting itself was very straightforward as well, it used OnTriggerEnter2D upon impact of colliding with another 2D collision box and if that game object was an “Enemy” it would take damage (damage pre set in enemy script) and if there was no more damage left, it would initiate sequence Destroy(gameObject); getting rid of the enemy from the game. | 27/04/2020 |
| Coin Score | Destroy(gameObject); | This is probably the easiest line of code to write and understand and have used it a lot in this process of my project. This instance can be applied to any game object, provided its referenced in the script. In this instance, the coins were grouped by a tag in the hierarchy, and upon them being destroyed the script would interact with the UI which would add value to the number past the semi colon, so the current value was added. | 09/05/2020 |
| Collectables | OnCollisionEnter2D | I learned a lot about 2D collision and its importance in a platforming 2D game. Virtually every game object contained a 2D collision box that had some important purpose upon entering or triggering. In the instance of collectables, using the 2D collision was for the purpose of being destroyed later in the script, where the script would find the other game object by comparing the tag to the player. | 09/05/2020 |