**Game Programming**

4 tutorials with explanation of scripts (Project)

**Tutorial 1:**

**// For this tutorial I wanted to create Raycast system in order for the character t perceive distances therefore allowing the player to interact with certain objects when he reaches a certain distance away//**

public class Playercasting : MonoBehaviour

{ public static float DistanceFromTarget;

// Using the public static allows this code to be used in conjuncture to different scripts therefore essential in order to link multiple components together, even though static is a public variable it does not show on the inspector therefore in order to check the distance via the inspector I made another public variable //

public float ToTarget;

void Update() {

RaycastHit Hit;

//IF statements are used to determine the value in this case it will be recording the RayCast number, The use of Vector3 is to give the number a certain direction therefore I used forward as to only record the distance from certain object that are in front of my character //

if (Physics.Raycast(transform.position, transform.TransformDirection(Vector3.forward), out Hit))

// The lines of code ToTarget=Hit.Distance is for the internal variable to be stated. I then had to write a line for the public variable connecting it to the internal variable to make sure that the value is the same and therefore able to be viewed in the inspector panel//

{ ToTarget = Hit.distance;

DistanceFromTarget = ToTarget;

}

}

}

//This script is then attached to the player controller to record how far away the player is from certain objects//

Tutorial 2:

//For this tutorial I wanted to create a notice board in which the player can interact with, this follows nicely with the Raycast script I made in the first tutorial as now my player can perceive distances. Therefore, I will use that to make my player interact with the notice board at a set distance//

public class Quest001Take : MonoBehaviour

//For this section I had to state all my variable in which I will be trying to integrate as well as the different UI elements such as using a different camera for viewing the quest (NoticeCam)//

{ public float TheDistance;

public GameObject ActionDisplay;

public GameObject ActionText;

public GameObject UIQuest;

public GameObject ThePlayer;

public GameObject NoticeCam;

void Update()

//For this part I had to reference the previous script to combine the two elements together first I had to reference the script itself then the variable within the script I want to use//

{ TheDistance = PlayerCasting.DistanceFromTarget;

}

void OnMouseOver()

// This is used to see whether the mouse is hovering over the certain trigger which I want to activate//

{

if (TheDistance <= 2)

// This distance was determined via testing the ToTarget distance in the inspector panel to accurately see what value makes sense for the ActionText and the ActionDispaly to be displayed//

{

ActionDisplay.SetActive(true);

ActionText.SetActive(true);

//This is used to display the two functions, set to true in order to display them//

}

if (Input.GetButtonDown("Action"))

// This is used to actually check if the action button is being triggered//

{

if (TheDistance <= 2)

// This is referred to as a nested function which means multiple functions leading into eachother//

{

ActionDisplay.SetActive(false);

ActionText.SetActive(false);

UIQuest.SetActive(true);

NoticeCam.SetActive(true);

ThePlayer.SetActive(false);

}

// These line of codes simply tell the system to trigger certain functions such as the Notice Camera and the UIQuest//

}

}

// this section is used to turn the functions off when we are not looking at the notice board acting as an inverse rule//

void OnMouseExit()

{

ActionDisplay.SetActive(false);

ActionText.SetActive(false);

}

}

**Tutorial 3:**

// for this tutorial I wanted to create a quest manager sort of script that will record what quest the player has active//

public class QuestManager : MonoBehaviour {

// for the first line I used int active quest number to keep track of what quest is being active//

public static int ActiveQuestNumber;

public int InternalQuestNumber;

void Update () {

InternalQuestNumber = ActiveQuestNumber;

//In order for the script to work I needed to make the internal quest equal the Active quest//

}

}

**Tutorial 4:**

//For this script I wanted to create the different quest for this I will do 3 different quest. This script will also cause the objectives of the quest to appear once the quest is accepted. This will bring all the other codes as well as actions together as now I will be able to accept the quest notice and trigger the the actual quest and the objectives of the quest//

// first I have to tell the script that I am using UI components this is done by at the very top adding in this line//

using UnityEngine.UI;

public class Quest001Buttons: MonoBehaviour

{

// For this section I need to address all the game objects which will be used//

public GameObject ThePlayer;

public GameObject NoticeCam;

public GameObject UIQuest;

public GameObject ActiveQuestBox;

public GameObject Objective01;

public GameObject Objective02;

public GameObject Objective03;

//to use a button, one must put public void rather than just void in order for it to work, for this section I need to address what will happen once I accept the quest. Firstly, I want the player to be able to resume playing therefore having to set it to true, then I want the NoticeCam to disappear therefore it has been set to false. I also want the UIquest to disappear, The Coroutine is being used as it cannot be done in a void class and this is for the animation of the objectives to play//

public void AcceptQuest()

{

ThePlayer.SetActive(true);

NoticeCam.SetActive(false);

UIQuest.SetActive(false);

StartCoroutine(SetQuestUI());

}

// all of these functions will be called when the accepts button is pressed, therefore, this section I have to address all the objectives in which I want the player to see//

IEnumerator SetQuestUI()

{

// For the text components I am addressing both the main component which is Text with a capital and addressing the subcomponent of that which has a lower-case T “.text”//

ActiveQuestBox.GetComponent<Text>().text = "My First Weapon";

Objective01.GetComponent<Text>().text = "Reach the clearing in the wood";

Objective02.GetComponent<Text>().text = "Open the chest";

Objective03.GetComponent<Text>().text = "Retrieve the weapon";

QuestManager.ActiveQuestNumber = 1;

yield return new WaitForSeconds(0.5f);

ActiveQuestBox.SetActive(true);

yield return new WaitForSeconds(1);

Objective01.SetActive(true);

yield return new WaitForSeconds(0.5f);

Objective02.SetActive(true);

yield return new WaitForSeconds(0.5f);

Objective03.SetActive(true);

yield return new WaitForSeconds(9);

ActiveQuestBox.SetActive(false);

Objective01.SetActive(false);

Objective02.SetActive(false);

Objective03.SetActive(false);

}

// “QuestManager” has been changed to = 1 in order to address that this is the first quest, I use the” yield return new waitforseconds” purely for how long I want the objectives to remain in the players screen. In the end of this section I have to set all of them to false in order to turn all of these off//

public void DeclineQuest()

{

ThePlayer.SetActive(true);

NoticeCam.SetActive(false);

UIQuest.SetActive(false);

}

}

// The next step was to address weather the player pressed the decline button and nothing happened. There for these lines of script were made so that when the player pressed it the player will be able to resume playing and the NoticeCam, UIQuest will turn off from the players screen//

This concludes all 4 tutorials which I used to make up my project.