CW1 Tutorial – Patrolling

The first steps would be to start up a project and insert a plane.

Next you should insert 3 cubes with the scale 0.3, 0.3, 0.3. These should be randomly spread out along the plane, these should be named Waypoint 1, Waypoint 2, Waypoint 3.

A sphere should be placed on the plane and a script should be added to it. It can be called anything, but I would suggest Patroller.

Once the script has been opened, we would need to define some variables.

public Transform[] waypoints;

public int Speed;

private int waypointIndex;

private float dist;

The variables are to adjust the speed, creating the array to store the waypoints and to calculate the distance between the points.

The next steps are to create a new function “Partol”

void patrol()

{

transform.Translate(Vector3.forward \* speed \* Time.deltaTime);

}

The patrol function is what allows the sphere to move.

Creating a new function for the Increase index would be the next step. The increase index would be how we create the array and how many steps would be in the array

void IncreaseIndex()

{

waypointIndex++;

if (waypointIndex >= waypoints.Length )

{

waypointIndex = 0;

}

transform.LookAt(waypoints[waypointIndex].position);

}

The next steps are to fill in the start function.

void Start()

{

waypointIndex = 0;

transform.LookAt(waypoints[waypointIndex].position);

}

In the start function the first line of code is to set the waypoint index to 0. The second line ensures that the sphere is facing the correct waypoint in the waypoint array.

The final step for the patrolling would be to add to the update function.

void Update()

{

dist = Vector3.Distance(transform.position, waypoints[waypointIndex].position);

if(dist < 1f)

{

IncreaseIndex();

}

patrol();

}

In the update function the first lines of code, this is where we would define what “Dist” is. This is where the array will be stored and it will progress to the next step of the array after the sphere has reached the waypoint.

CW1 Tutorial – Reload the Game

The first step would be to create a new script, that would be used to reload the game. The first thing that would be needed is to add a new unity engine.

using UnityEngine.SceneManagement;

A new private void would be the next thing that is needed.

private void OnTrigggerExit(Collider other)

{

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);

}

The private void would be for the trigger enter, this means that once the player enters the trigger of the sphere the game will reload.

CW1 Tutorial – Movement

The first step would be to insert the object that we want to move. For the example I added a cylinder.

The next step would be to create a new script, this script would control he characters movement.

The first thing to do on the script is to create a public float so that the speed can be adjusted if needed without changing the code.

public float speed = .1f;

The next step is to write the code for the movement.

void Update()

{

float xDirection = Input.GetAxis("Horizontal");

float zDirection = Input.GetAxis("Vertical");

Vector3 moveDirection = new Vector3(xDirection, 0.0f, zDirection);

transform.position += moveDirection;

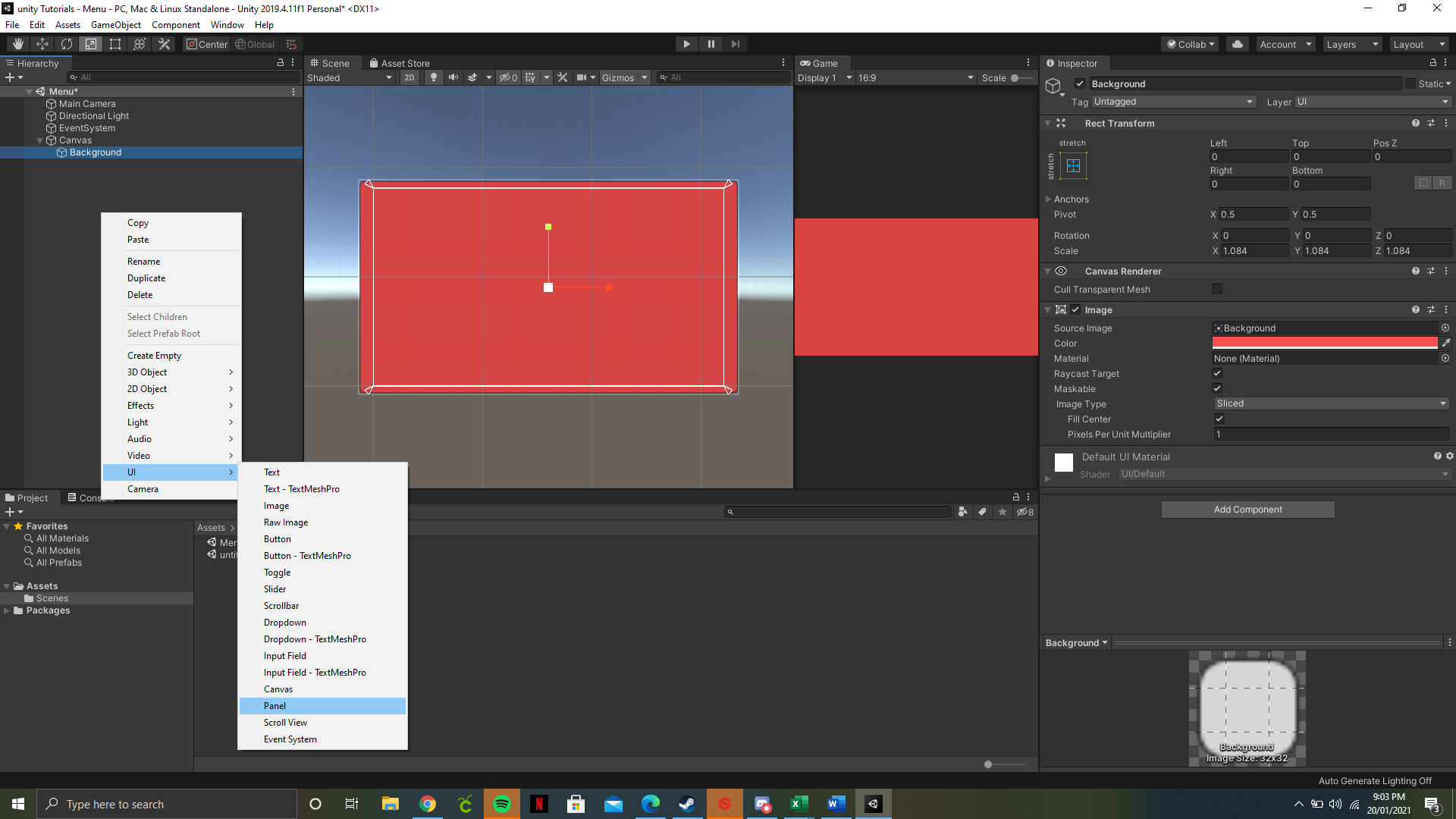
}

The first two lines get the input from the keys the that are pressed. The next line takes the float from the first two lines and uses them as the new vectors for the movement. The final line will make the cylinder move according to the new vector3.

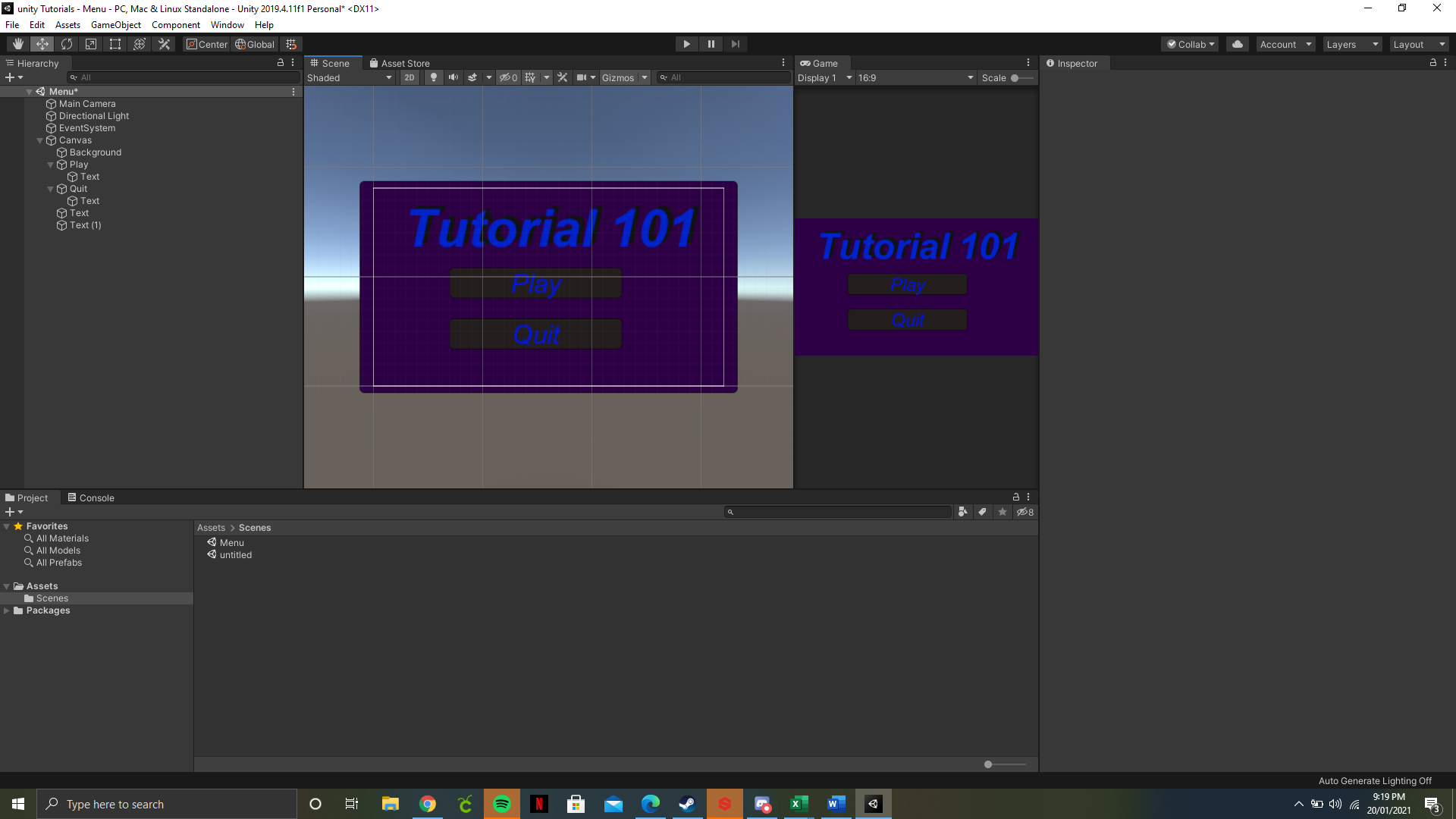
CW1 – Main Menu

The first step would be to create a new scene, this is where the main menu would be stored so renaming the scene to “Menu” or “Main Menu” would be appropriate.

The first thing to do would be to create a UI Panel, this panel will create a UI canvas automatically. The panel is what we would use to change the colour of the background to what we desire.



The next step would be to implement buttons for the menu. For this tutorial we will only cover the play and quit buttons. Once the two buttons are created, they should be renamed to “Play” and “Quit”. They can also be recoloured to fit the colour scheme of the game.



The next thing that is needed is to create a script to give the buttons the functions that they need.

We would remove the start and update functions because they aren’t needed for this script. We would need to ensure that we are using Unity Engine Scene Management. That would need to be added at the top. We would need to create new functions, the functions we need are the public void to start the game and the public void to quit the game.

public void PlayGame()

{

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);

}

public void QuitGame()

{

Application.Quit();

}

The code within the “PlayGame” function would take the active scene, which in this case would be the main menu and it go onto the next scene in the queue. The way to view and edit the queue would be by checking the build settings, that’s where it will show the queue of scenes.

The next step would be to add the Menu script to the canvas. Next the canvas would need to be dragged to the “On click” section for the button. We would need to find the Main Menu section and click on the “PlayGame” function. The same thing would need to be done for the quit button.

