***Introduction to Graphics and Mobile Gaming***

**LAB 4: Part 1**

**Moving objects**

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc36123980)

[2 Moving the chess piece 1](#_Toc36123981)

[2.1 Collision box 1](#_Toc36123982)

[2.2 The game script 2](#_Toc36123983)

[3 Room navigation 2](#_Toc36123984)

[3.1 Controls 2](#_Toc36123985)

[3.2 Camera 4](#_Toc36123986)

[4 Collision 5](#_Toc36123987)

[4.1 Physics of the room 5](#_Toc36123988)

[4.2 Physics on the chess pieces 6](#_Toc36123989)

[4.2.1 Movement direction 8](#_Toc36123990)

# Introduction

In this lab, we will look at designing a UI and implementing physics and other visual elements into the game.

The aim is to transform the chess room into a short gameplay demonstration to show you how physics, particles, and game programming work in Unity. We will do this through creating a scoring system with a timer to allow for gameplay elements to be used.

# Moving the chess piece

To enable the user to move the chess pieces, we need to have 2 things to do:

1. Add a collision box around every chess piece
2. Add the script to provide movement instructions

## Collision box

Click once on any chess piece and then in the Inspector panel, click on *Add component,* and add *“Rigidbody”* and *“Box Collider”*. The Rigidbody parameters are the following:

* Mass = 1
* Drag = 0.5
* Angular drag = 0.05
* Use Gravity ticked

Leave everything else unchanged. The Box Collider element does not need any changes, though the size can be changed to adapt better to the chess piece.

To increase the level of optimization we have, let’s change to the following options for the Mesh renderer of each chess piece:

## The game script

First, add a new empty object and rename it “moveChessPiece”, highlight it and drag the *moveChessPiece* script in the Inspector panel.

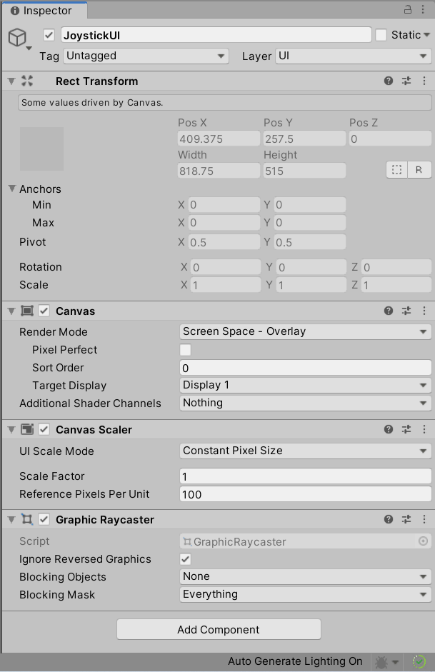
Now press Play and ensure none of the chess pieces are falling. If this does happen, select them, press ctrl+shift, and drag them to make sure they are on the same y-coordinate as the chess board.

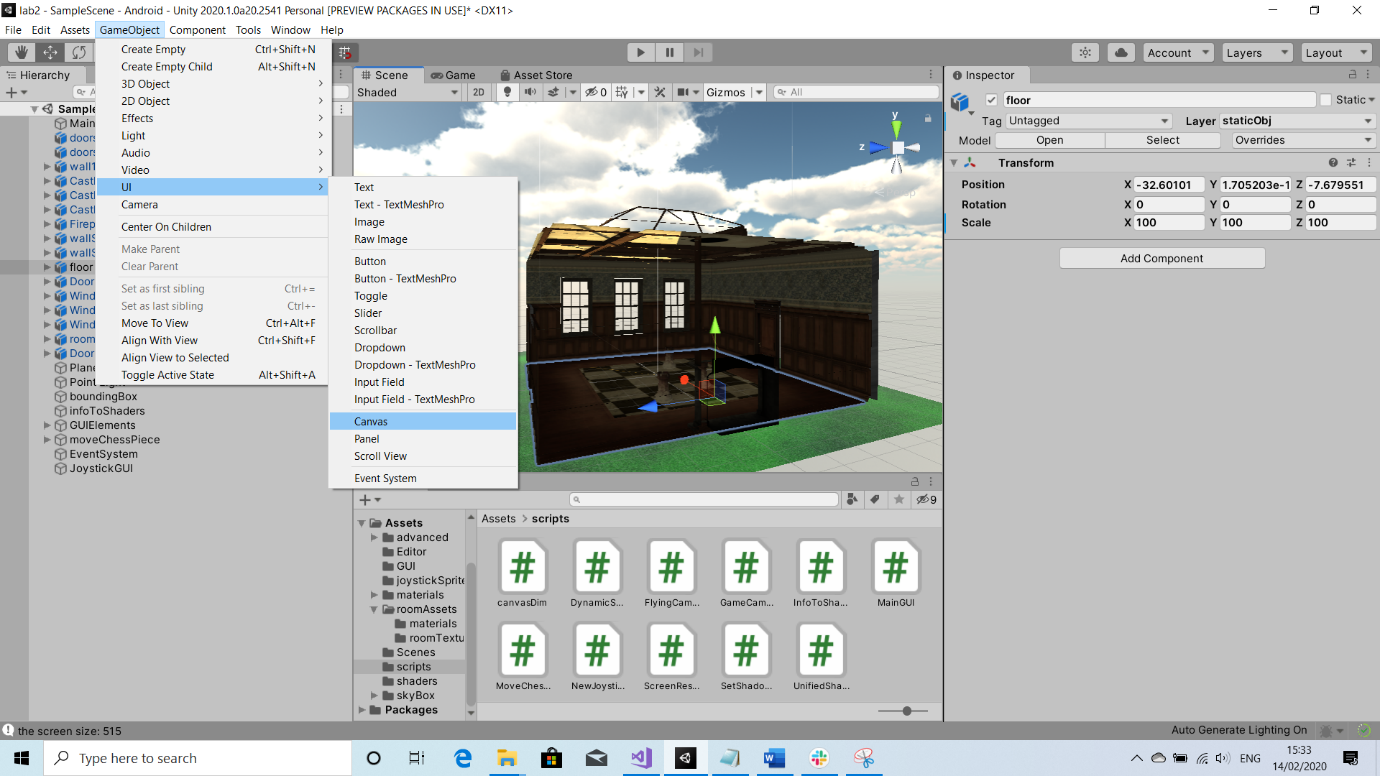
You can build and install the apk now. The screen is divided into 3 parts, so you’ll notice that areas of the screen that cannot be interacted with (the sides). This is because the sides will handle the joysticks and shadows sliders, the center handles the movements of dynamic objects.

# Room navigation

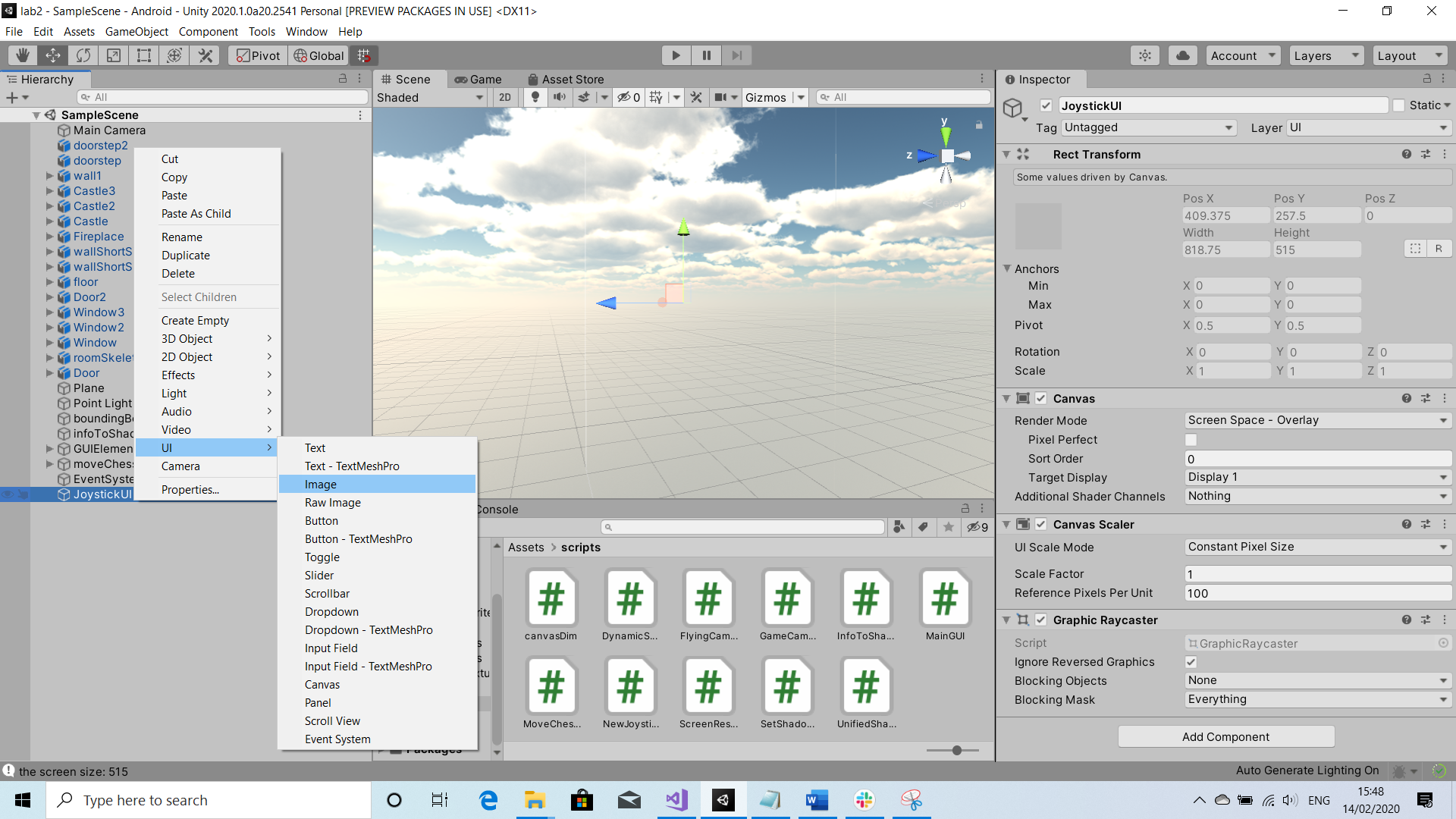
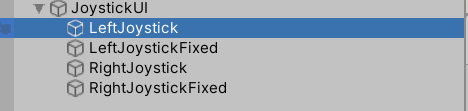
Now let’s enable the user to navigate through the room. The first thing to do is to create all the missing GUI elements and link them to the corresponding code.

## Controls

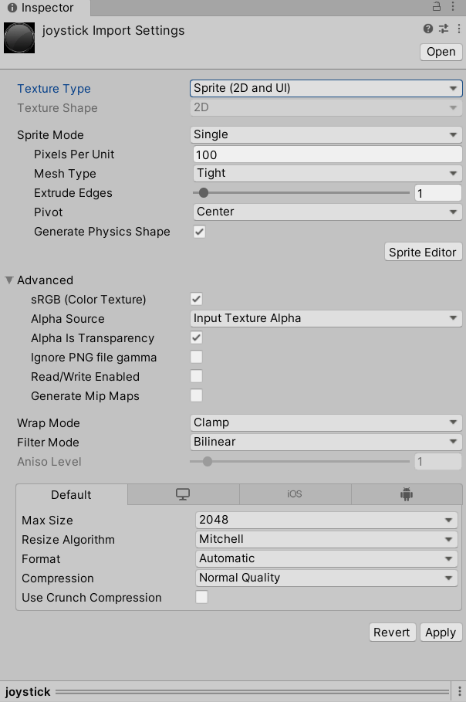
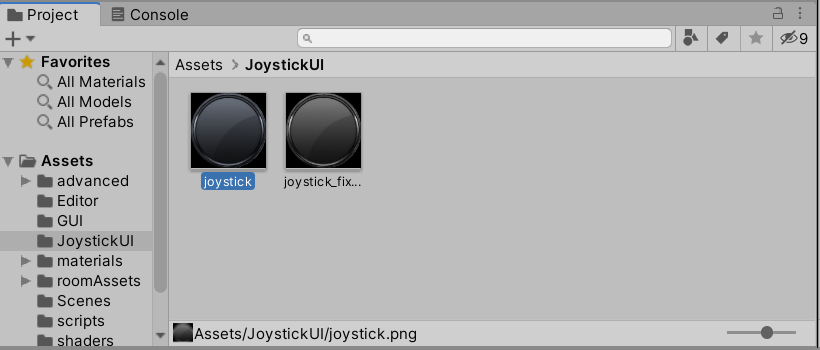
Go to GameObject > Select UI > click on Canvas, go to this object, rename it JoystickUI, and check its render mode is Screen Space-Overlay

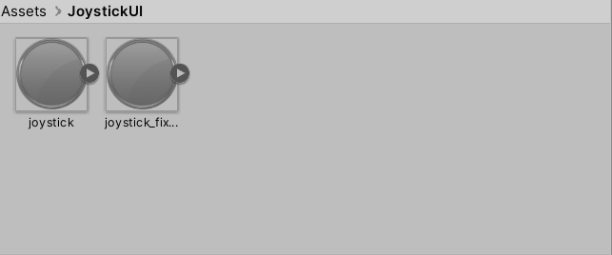


Add as children 4 Image objects to JoystickUI and rename them as follows: ***LeftJoystick***, ***LeftJoystickFixed, RightJoystick,*** and ***RightJoystickFixed***.

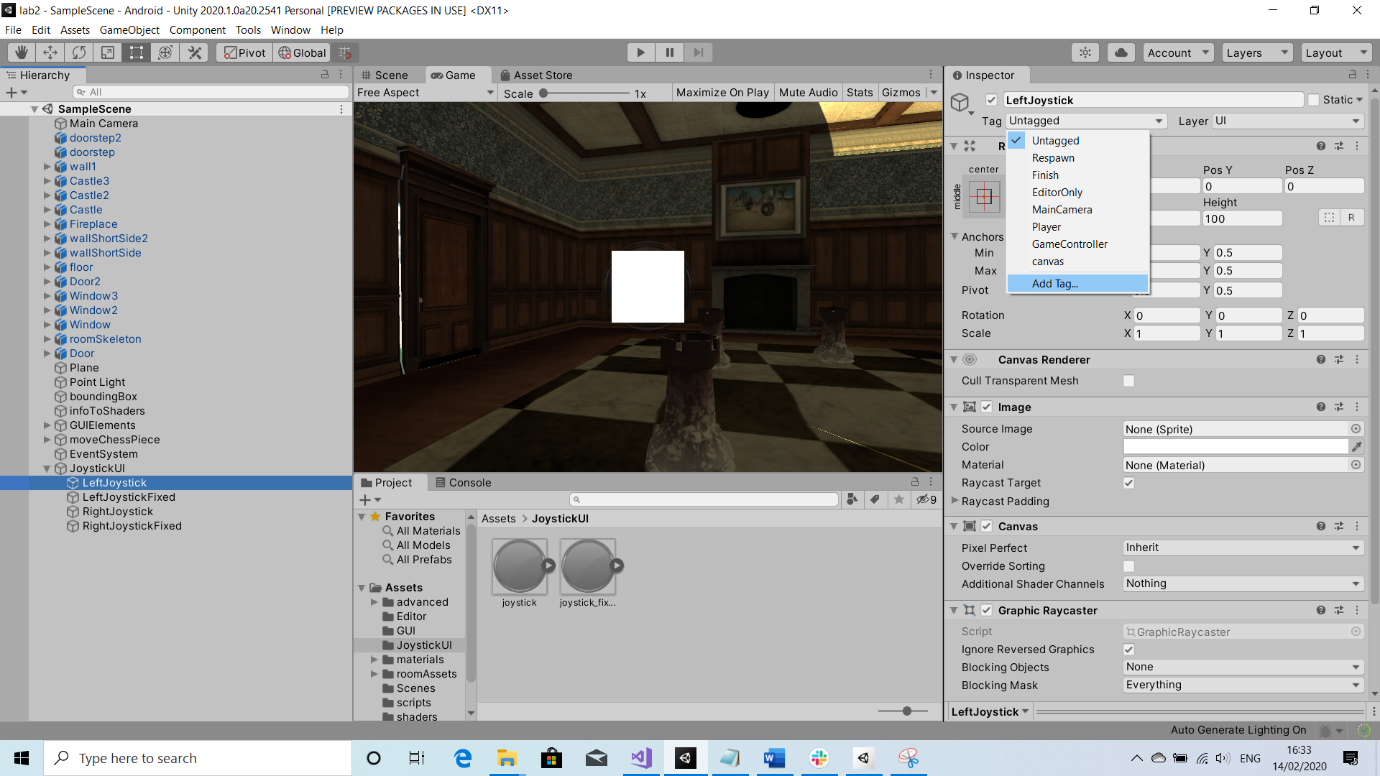
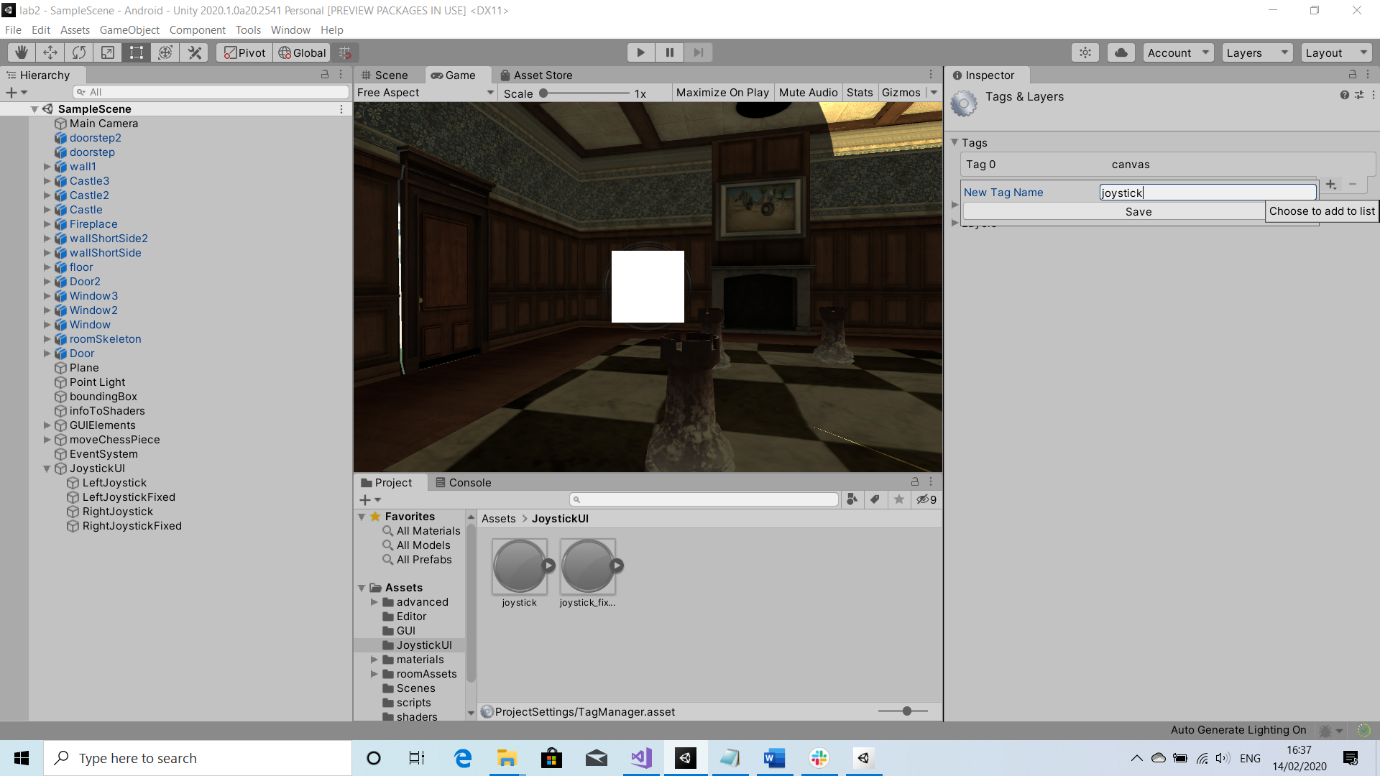


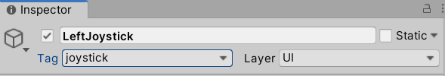
Copy joystickUI folder into your project > select joystick image in unity > go to inspector, change the texture type from Default to Sprite (2D and UI), and Apply. Do the same with joystick\_fixed img





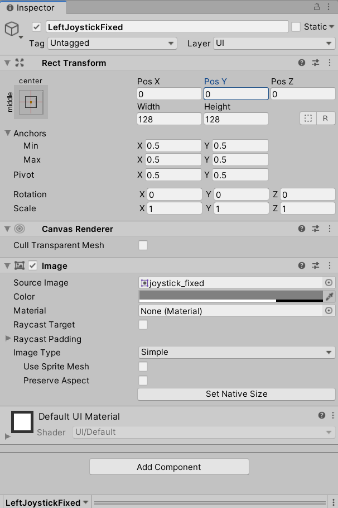
Click on leftJoystick > Go to inspector > Click on Tag > Add Tag > Click on the + sign > write joystick and save > add this tag to LeftJoystick and RightJoystick





Let’s start giving some properties to these new objects, starting with ***LeftJoystickFixed***:

* Drag the *joystick\_fixed* sprite and fill the Source Image field with it. Set the colour to RGBA = 128, 128, 128, 190. Set the Width = 128 and Height = 128, untick Raycast Target and check its Pos X, Y, Z = (0, 0, 0).



Next, let’s work on the **LeftJoystick** element:

* Drag the joystick sprite into the image field. Set the colour RGB value to 128, 128, 128, 190. Set the Width = 64 and Height = 64, check its Pos X, Y, Z = (0, 0, 0), tick Raycast Target and add graphic raycaster.
* Drag and drop the NewJoystick script from the scripts folder in the project panel to the Inspector of left joystick
* Specify the joystick type to be Left
* J Trans Clamp Value =100; Fade Out = 0.1; Fade in = 0, and Hold threshold = 0
* Link the LeftJoystickFixed image to the Fixed Joystick img field by dragging the LeftJoystickFixed element from the Hierarchy panel
* Finally, set the UI scale factor to 0.001 and the J Trans values to X = 0 and Y = 0.

We need to repeat the same 2 operations for the right joystick, remembering to apply the correct Joystick type and the correct Fixed Joystick image.

## Camera

A screenshot of a computer

Description automatically generatedTo enable the player to move around, we need to allow the camera to be used to move around the room.

Click on the main camera from the hierarchy panel and make sure the parameters are as follows:

Then, drag and drop the GameCamera script and from the scripts folder to the Main Camera Inspector tab.

Set the parameters as shown below:

A screenshot of a cell phone

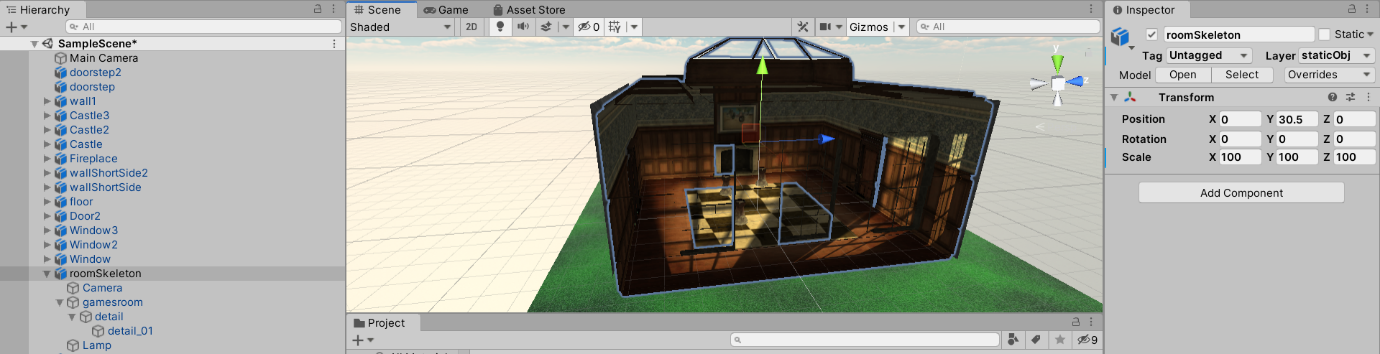
Description automatically generated

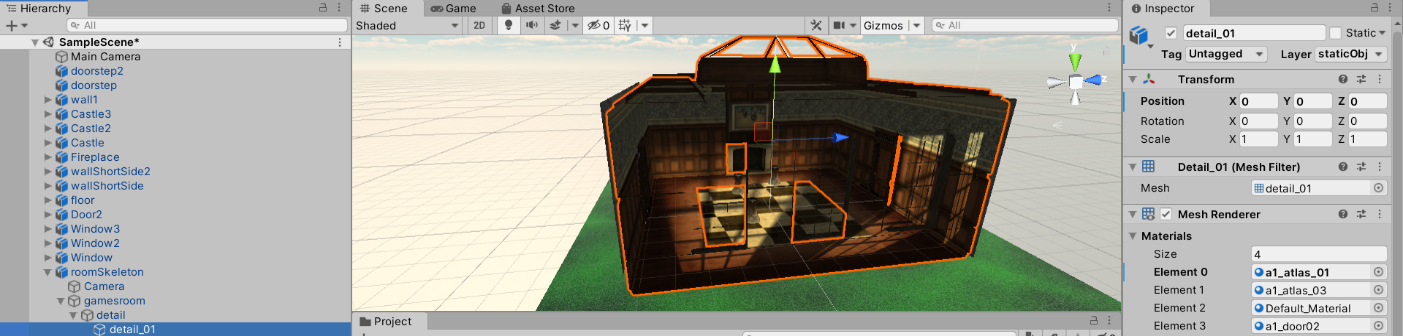
A screenshot of a cell phone

Description automatically generated

We are now ready to test our application! Build and run the apk on your device. Look at our new elements on screen and think whether they will need adjusting to suit the user experience better.

# Collision

Before implementing physics in the room, check that the roomSkeleton and detail\_01 positions are equal to zero in x and z



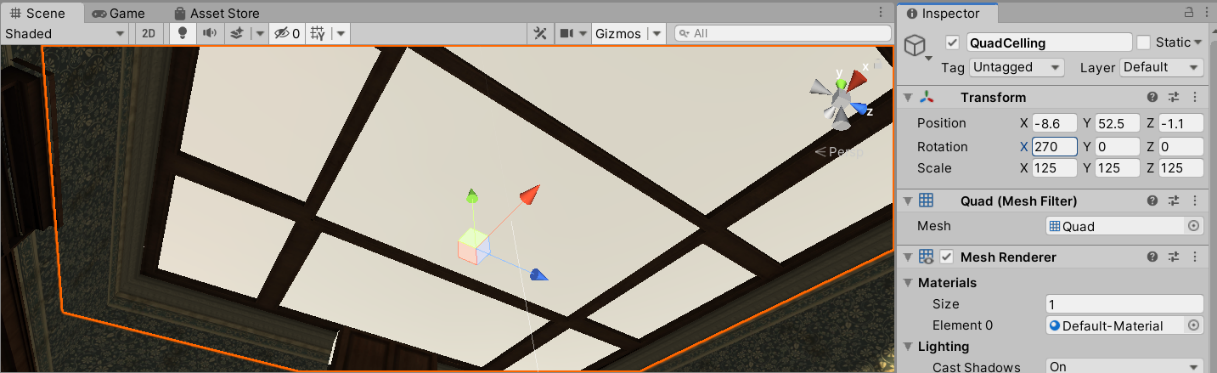
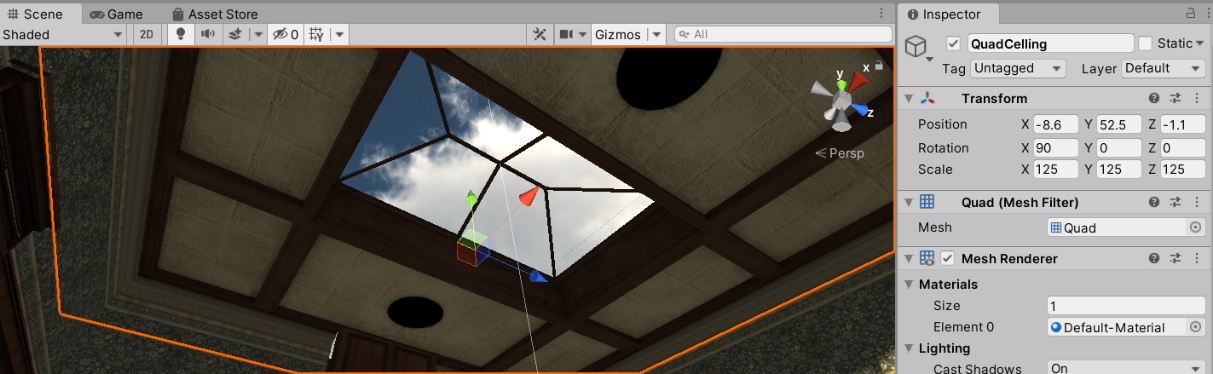
## Physics of the room

Currently, we can exit the room. There is the possibility of getting the pieces outside of the room when the pieces collide. We want to avoid this as the game playing space will exclusively be within the walls of our room.

We will need some extra 2D planes around the walls, floor, and ceiling to stop the chess pieces from exiting the room. These planes will have physics attached to them.

Create a new game object called a Quad (we use quads instead of planes as they contain fewer vertices):

* Scale the quad to match the size of the walls, floor, and ceiling (a quad for each surface)
* Place the quads with their faces pointing toward the room, otherwise quads will be ignored by chess pieces



* From the Inspector, untick Mesh Renderer to make it invisible and make sure collision is active.

Rebuild the project and install the APK onto our device, now none of the chess pieces should be able to exit the room because of collisions.

## Physics on the chess pieces

We would like to create a force in the opposite direction of a screen press, when a screen press is on a chess piece. Open your project and then go to the scripts folder and double-click on the *MoveChessPiece* script to open it in the default editor.

Define a public float at the beginning of this class and rename it *projectileForce*. Initialize it to 500, though note that we will be able to change this later from the Inspector panel of moveChessPiece element. All public declared variables can be changed from the Unity editor.

To move a chess piece, we will use the following function, which we will call ***ApplyForceAtChessPiece***:

public void ApplyForceAtChessPiece()

{

//Find vector from camera to object

Vector3 camToObjVec = hitGO.transform.position - Camera.main.transform.position;

//Normalize this vector

camToObjVec.Normalize();

//create a random vector and multiply it with the projectileForce

Vector3 force = new Vector3(Random.Range(-1.0f, 1.0f), Random.Range(-1.0f, 1.0f),

Random.Range(-1.0f, 1.0f)) \* projectileForce;

//add this vector to the rigid box as something was pushing/colliding with it

hitGO.GetComponent<Rigidbody>().AddForce(force);

}

Now, look at the rest of the code and try to understand how this works.

Inside of the OnGUI() function, there is the following **if** statement. Note the comments that will explain each portion of the code.

if (Application.platform == RuntimePlatform.WindowsEditor) <- detects if scene is playing on windows

{

// The mouse event

Event e = Event.current; <- creates an event and then will check if it’s of type mouse

if (e.isMouse)

{

//The following line gets the position of the touch on the screen (click on Windows)

Vector2 touchPos = new Vector2(e.mousePosition.x, Screen.height - e.mousePosition.y);

if (e.type == EventType.MouseDown ) <- this corresponds to a click

{

RaycastHit hit; <- this is a ray that detects collision on dynObj

Vector3 touchPos3D = new Vector3(touchPos.x, touchPos.y, 0.0f);

Ray ray = Camera.main.ScreenPointToRay(touchPos3D); <- ray starts from camera

if (Physics.Raycast(ray.origin, ray.direction, out hit, Mathf.Infinity, layerMask))

{

// A Game Object in the layerMask has been hit

hitGO = hit.collider.gameObject;

screenPoint3D = Camera.main.WorldToScreenPoint(hitGO.transform.position);

offset3D = hitGO.transform.position - Camera.main.ScreenToWorldPoint(new

Vector3(touchPos.x, touchPos.y, screenPoint3D.z));

// Store GO Y axis coordinate

goPosY = hitGO.transform.position.y; <- game obj coordinates

}

}

if (e.type == EventType.mouseUp ) <- if is just a click

{

TODO:

If an obj has been hit and user is not dragging

Call the apply force function to move obj

// Reset hit GO

hitGO = null;

}

if (e.type == EventType.MouseDrag && hitGO != null) <- if we are dragging

{

To DO:

Set dragging variable

Vector3 curScreenPoint3D = new Vector3(touchPos.x, touchPos.y, screenPoint3D.z);

Vector3 curPosition = Camera.main.ScreenToWorldPoint(curScreenPoint3D) + offset3D;

// We don't want the object moving along Y axis

curPosition.y = goPosY;

KeepObjectInsideTheRoom(ref curPosition);

// Update hit GO 3D position

hitGO.transform.position = curPosition; <- set new position

}

To DO:

Restore dragging variable when obj hasn’t been hit or user is not dragging

}

}

You now have to complete the TODOs in the code. A few hint notes:

* **hitGO** is null in the beginning, meaning no GameObject was hit
* **ApplyForceAtChessPiece** applies the force at the object detected as ‘hit’
* You might find it useful to keep track of whether the user is **dragging** their mouse on the screen. Usually, the game doesn’t start up with the user dragging the mouse on the screen

If you get stuck, navigate to the Solutions folder and copy the code if necessary.

We now look at the Update() function. Here, after the if (Input.touchCount > 0) statement, you will find a code similar to the one above, but dedicated to mobile devices.

if ((fingerId == touch.fingerId) && (touch.phase == TouchPhase.Ended || touch.phase == TouchPhase.Canceled))

corresponds to the following line for Windows:

if (e.type == EventType.mouseUp )

while this line:

if (hitGO != null && (touch.phase == TouchPhase.Stationary || touch.phase == TouchPhase.Moved))

corresponds to this one for Windows:

if (e.type == EventType.MouseDrag && hitGO != null)

You will need to fill in the code for applying the force on the chess piece on a mobile device too.

### Movement direction

You can build and run the apk now. **If you try to click on a chess piece, you will notice that objects are shot in random directions,** and there’s no effect indicating they have been hit apart from the movement.

So, the first thing we want to do is fix the direction the chess pieces travel. Navigate through the code until you reach the ApplyForceAtChessPiece() function.

Before, we defined the force and its direction for moving a chess piece as follows:

//create a random vector and multiply it with the projectileForce

Vector3 force = new Vector3(Random.Range(-1.0f, 1.0f), Random.Range(-1.0f, 1.0f), Random.Range(-1.0f, 1.0f)) \*

projectileForce;

Which, as the comment says, will generate a random vector with a random direction for the force. We don’t want that. So, what direction do we actually want?

Previously (2 lines above) you can observe that we have computed the vector from the camera position to the object position. This would be something sensible to use, that way, we can see the object being pushed in the opposite direction of the camera. So, we just replace our random vector with the *camToObjVec*:

Vector3 force = camToObjVec \* projectileForce;

We may discover an issue with the sensitivity of the touch function, if the user accidently drags the chess piece 1 millimeter; it will not be considered as a tap. To fix this, scroll down in our function until we find the if statement that handles dragging on mobile and modify the code as follows:

//Add variable to keep track of lastTouchPosition

Vector2 lastTouchPos;

…

// If the user drags the finger or keep it touching in the same place  
             else if (hitGO != null && (touch.phase == TouchPhase.Stationary || touch.phase == TouchPhase.Moved))

{

//TODO: Keep track of dragging on the screen

//LAB4\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

dragging =true;

//LAB4\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Vector3 curScreenPoint3D = new Vector3(touchPos.x, touchPos.y, screenPoint3D.z);

Vector3 curPosition = Camera.main.ScreenToWorldPoint(curScreenPoint3D) + offset3D;

// We don't want the object moving along Y axis.

curPosition.y = goPosY;

// Additionally to collision detection avoid object passing through walls as we are not using physics to move the obejct

//LAB4\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Vector3 hitV = hitGO.transform.position;

float dist = Vector3.Distance (hitV, curPosition);

if (dist <= 0.1) {

ApplyForceAtChessPiece ();

}

else {

KeepObjectInsideTheRoom(ref curPosition);

// Update hit GO 3D position.

hitGO.transform.position = curPosition;

}

}

else

{

dragging = false;

}   
            lastTouchPos.x = touchPos.x;  
            lastTouchPos.y = touchPos.y;

This should improve shooting the chess pieces.

You can now build and install the apk again to see the changes.