Virtual buttons in Unity

This uses the space invaders game from another tutorial as a base. In that the controls are done using keyboard controls, within a mobile app however, there is no keyboard the only controls are via the screen. We have a few ways of implementing touch controls in Unity. Multitouch, such as that shown in the pixelnest tutorial is one way, using gestures such as swipe, pinch and flick, that is undoubtedly useful and there are many assets on the appstore and tutorials dedicated to that. What we will look at is something a bit more useful.

Step 1: Mouse click to Tap

This needs little in the way of modification, in Unity mouse input can be mapped directly to a tap, if you use Input.GetMouseButtonDown() when you click in a PC based environment it will trigger, when you tap in a mobile environment, it will trigger. There are a few problems with this, one is that this only supports one singular finger, not multi touch, in addition, finger touch on mobile devices can move from one area to another with no movement between them whereas mouse simulation provides movement too. For this reason most of the time unless it's basic, mouse emulation based touch control isn't wanted.

Step 2: Using the Unity UI

All the Unity UI components are compatible across any input, if you tap on them, drag sliders, etc. This work brilliantly for games with a heavy reliance on simple input, you can make a simple game just from this sort of thing. Games like tic tac toe or matching games. The problem here is that you're limited to the Unity UI components, for instance, holding a virtual button down is hard to do with Unity's UI. The methodology is the same as any UI components though, and that's fairly simple. You link buttons to public methods, for instance a go right or go left method.

Step 3:

The third step is to take things a bit further, adapting and extending the UI elements (or just basic sprites) from Unity with some of your own code. That's what we will do for space invaders and this demo, it's one of the most versatile.

Implementing the virtual buttons

To the space invaders canvas, add four new buttons, we'll use one for left and one for right then one for up and one for down. Position and colour them however you want, you can even if you so want make them invisible, they'll still work.

We need to write a script that extends the button functionality, we do this using two point in classes, `IPointerDownHandler and IPointerUpHandler.` These are used to add some functionality to the class, allowing you to access special functions that recognise when a pointer is pressed and depressed on something, including a hidden pointer such as a touch screen. So, we of course can add this to the buttons we created earlier. We are going to create one simple to use script for all our commands, but you could easily separate it out, the concept is the same, set a variable true whilst you hold the button down (or the pointer down over the thing) in then, in update we check to see if that variable is true and if it is, we do whatever we need.

Without further ado, create a new C# script, call it “virtualButtons” and, after the MonoBehaviour bit, the colon after the name means that the script you are creating inherits and implements all of the functionality of the classes afterwards. In this case, monoBehaviour, IPointerDownHandler and IPointerUpHandler. We've told C# we ’re implementing new functionality for IPointerDown and Up on this particular script, almost all Unity scripts inherit monoBehaviour by default.

```
using UnityEngine.EventSystems;// Required when using Event data.


    public class virtualButtons : MonoBehaviour, IPointerDownHandler, IPointerUpHandler
    {

    // Update is called once per frame
    public bool buttonHeld;
    public bool isLeft;
    public bool isRight;
    public bool isUp;
    public bool isDown;
    public bool isFire;

    public GameObject ship;
    public GameObject shot;
```

We then create a whole bunch of scripts, most of these are for various states to recognise what the button is for, left, right, up down, fire. Then we’ve got one to hold the ship we’re controlling and one to hold the shot we fire.

Next up, underneath are two functions, OnPointerDown and OnPointerUp which are used to register whether a pointer is down over the object that the script is attached to. When it is, we set a variable to be true, when it’s not we set it to be false. That’s the basic premise of this.

```
public void OnPointerDown(PointerEventData eventData)
{
    buttonHeld = true;
}

public void OnPointerUp(PointerEventData eventData)
{
    buttonHeld = false;
}
```

```
public void Update()
{
    Vector3 movement= ship.GetComponent<Transform>().position;
    if (buttonHeld && isRight)
    {           movement+=  new Vector3(0.1f, 0,0);        }
    else if (
    {                                                      }
    else if (buttonHeld && isUp)
    {                                                      }
    else if (
    {           movement += new Vector3(0, -0.1f, 0);      }

    movement.x=Mathf.Clamp(movement.x, -5.35f, 5.35f);
    movement.y=Mathf.Clamp(movement.y, -9f, 9f);

    ship.GetComponent<Transform>().position= movement;
}
```

Here in update we can see what happens, this is for the movement.

First, we create a Vector3 to hold movement and fill it with the ship’s current position.

Then, we check if a button is held and which button.

Then we update movement by adding a new vector3 in the correct direction. Remember, positive means right, negative means left.

Challenge 1

Complete the boxes marked with a red outline, you need to add condition and make changes to movement.

After we make the changes here, we clamp the x and y position meaning it can't go above or below a certain value, saving it back into movement. Finally, we set the ship's position to this newly updated movement vector.

Drag the script onto each button and then the player ship into the correct slot for that script. It should now be testable.

A fire button

So, you've created a virtual movement set, what we're going to do now is add a fire button. Add it to the canvas and position and style it however you want. Into your script create a new public function called Fire().

Challenge 2

When the button is pressed, Fire a projectile, you might want to look at the old shipControl script to see how this was done. We don't want to use OnPointerDown for this, we instead want to use the UIButton's press. Why? Well, otherwise we end up spamming a whole lot of projectiles.

Challenge 3

Add to the game, create a pickup or more waves of enemies, create another scene that has different enemies. Perhaps even create another weapon that can only be fired occsasionally and requires a different type of button. The future is yours.

This was a brief tutorial but that's how virtual buttons and touch control work, you can, if you so wish, combine it with multitouch to get a very effective control medium.