UI Lab

We will make a shop to learn how UI elements work in Unity.

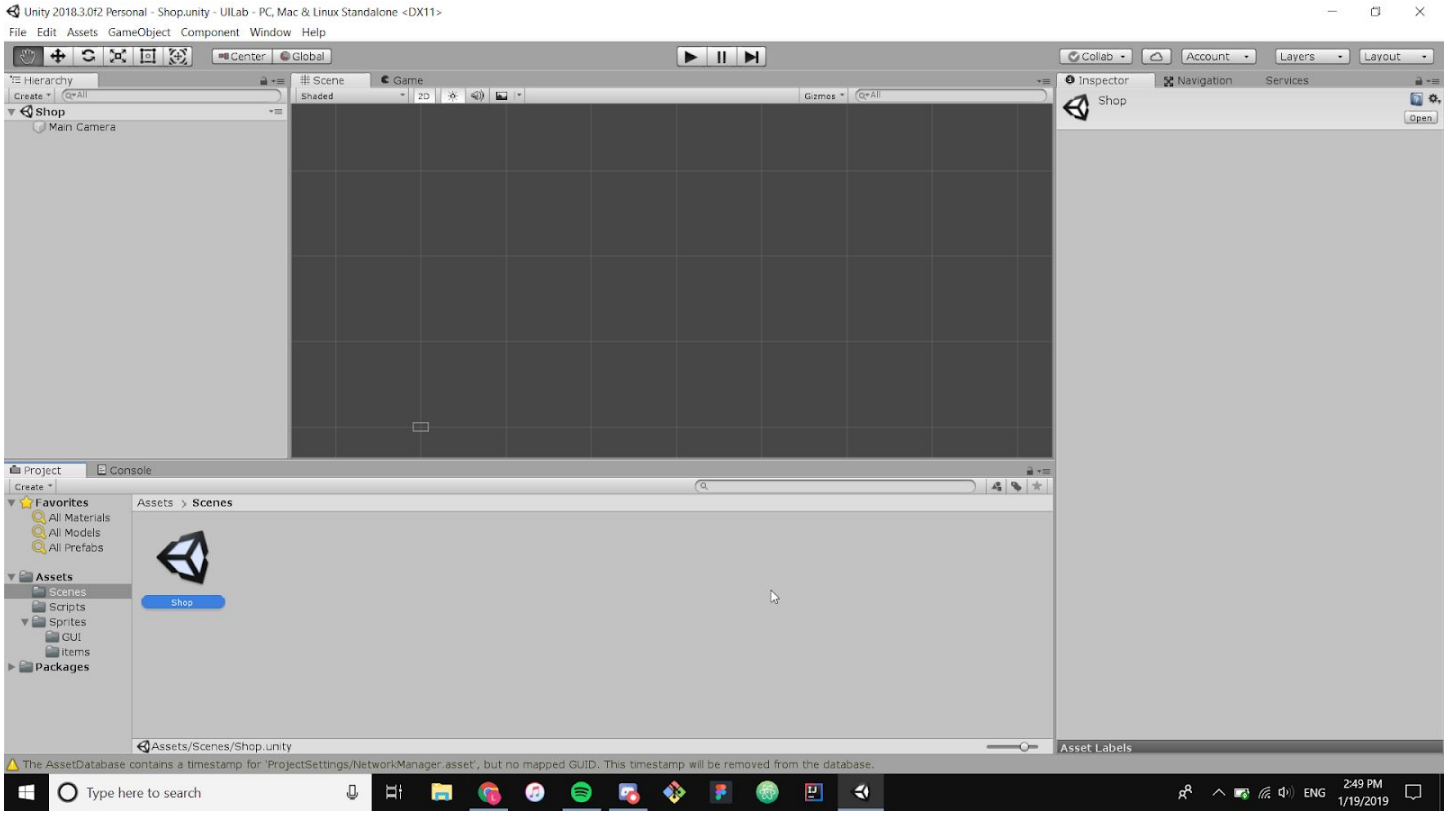
This lab is structured in two parts. Both programmers and artists will complete the first part in Unity. For programmers, the second section will teach how to write scripts in order to make UI elements do additional actions (e.i. button events and recording currency). For artists, the second section is to review the lecture on UI Art. You are only required to complete at least one of the two different second parts, but feel free to work through the other section if it interests you.

Note: Code will be given throughout and consolidated at the end.

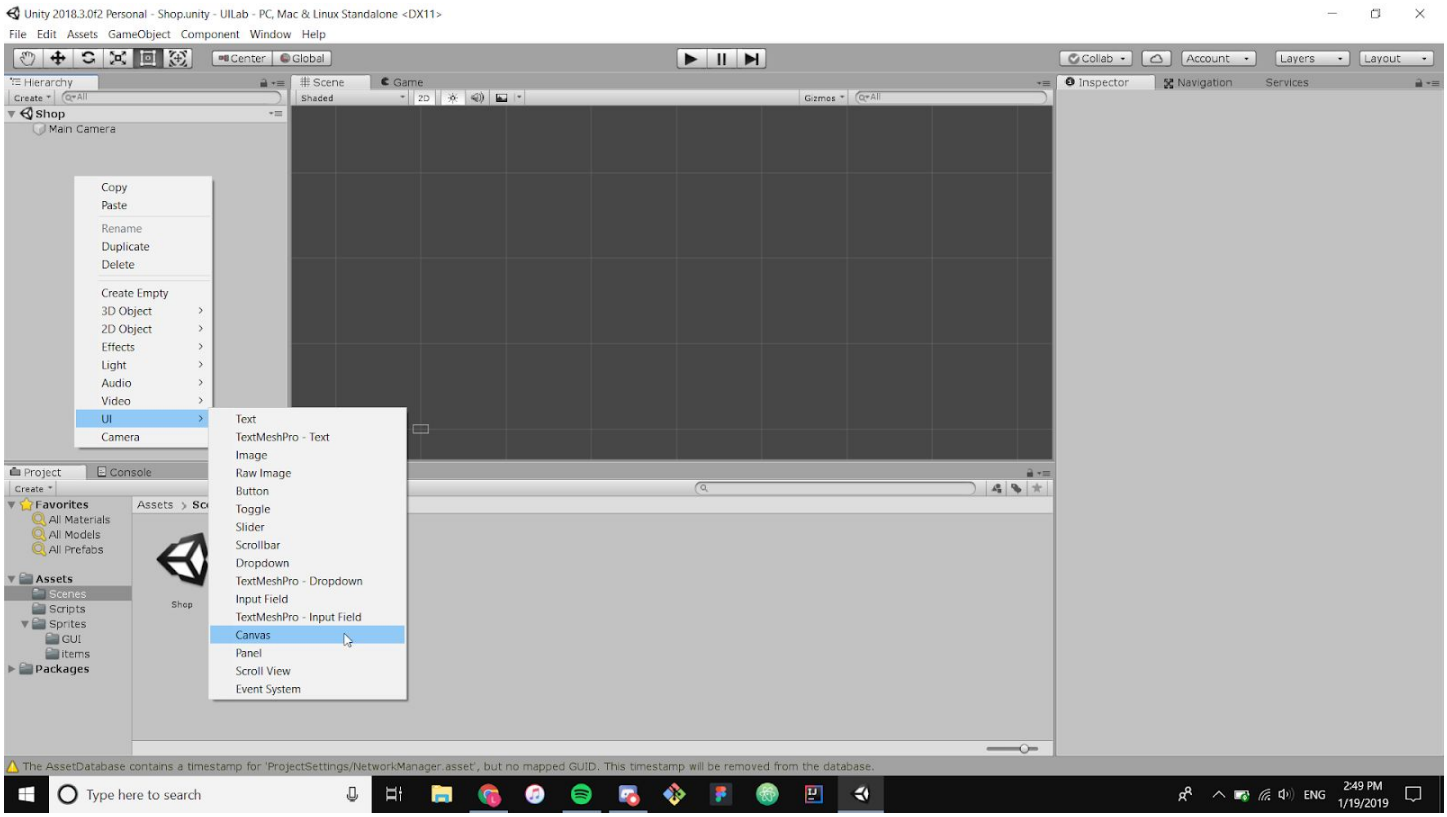
Part 1

You will be graded on having all the components and not the likeness so feel free to change the layout.

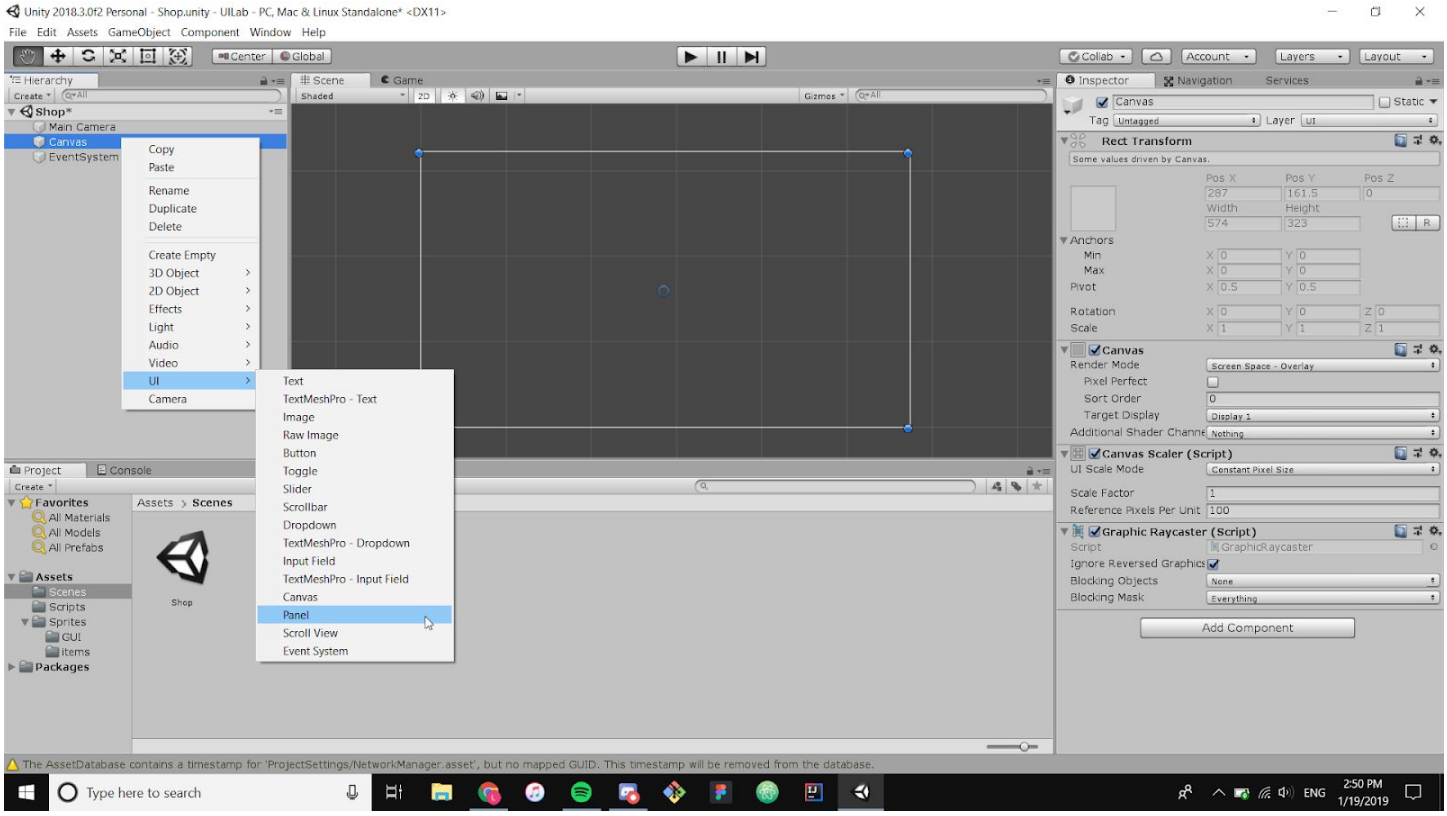
First, open up the project in Unity by double clicking on the Shop Scene in the scene folder



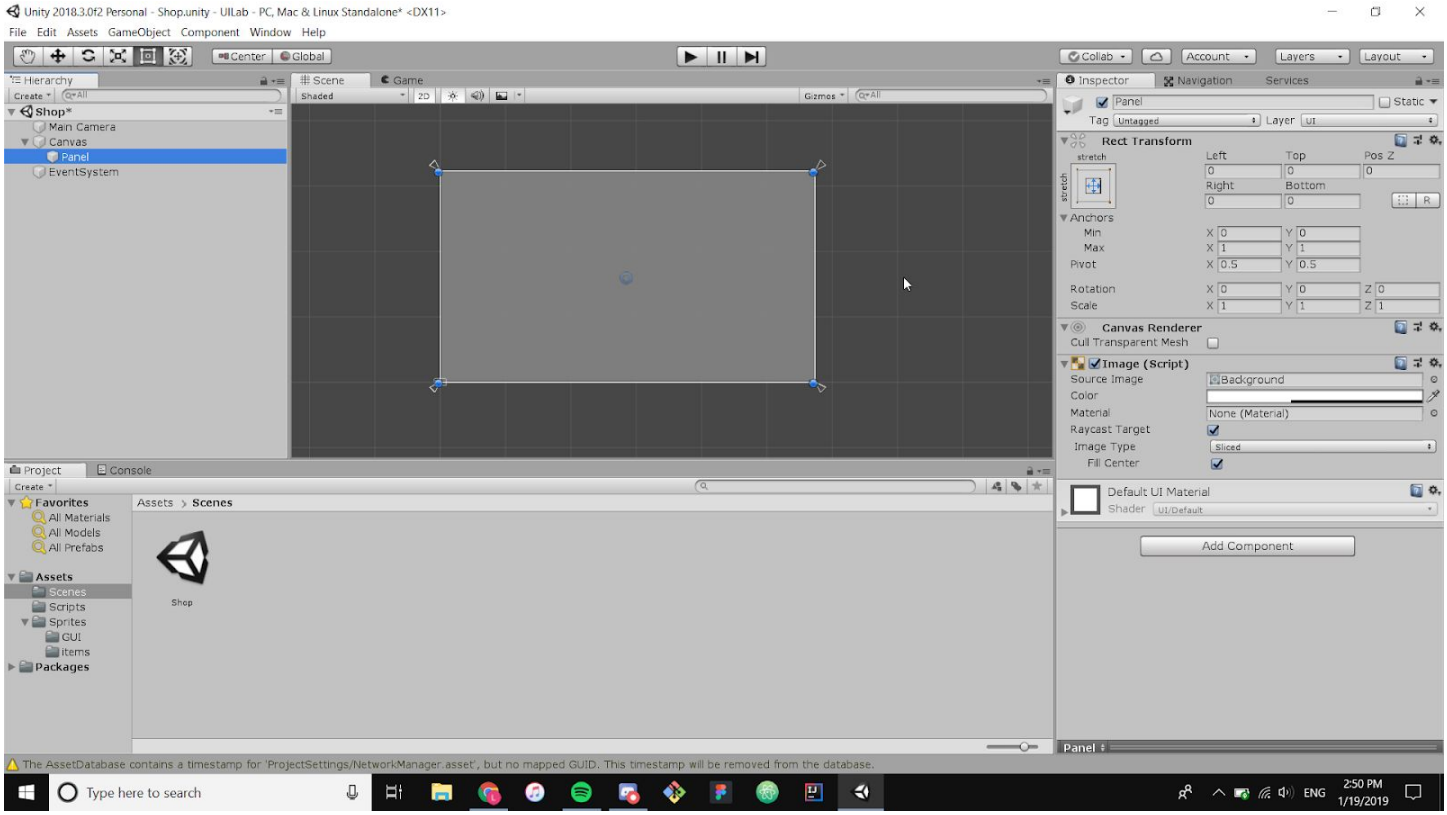
Now, let's make our first UI element. In the hierarchy on the left, right click and create a new UI element named Canvas. The canvas deals with everything UI and every UI element must be a child of a canvas. When you make a canvas, two things will appear: the canvas and an EventSystem. The EventSystem is responsible for processing and handling events in a scene. Every scene will have one EventSystem. We're not going to go into the EventSystem object in this lab, but feel free to look more into on your own time.



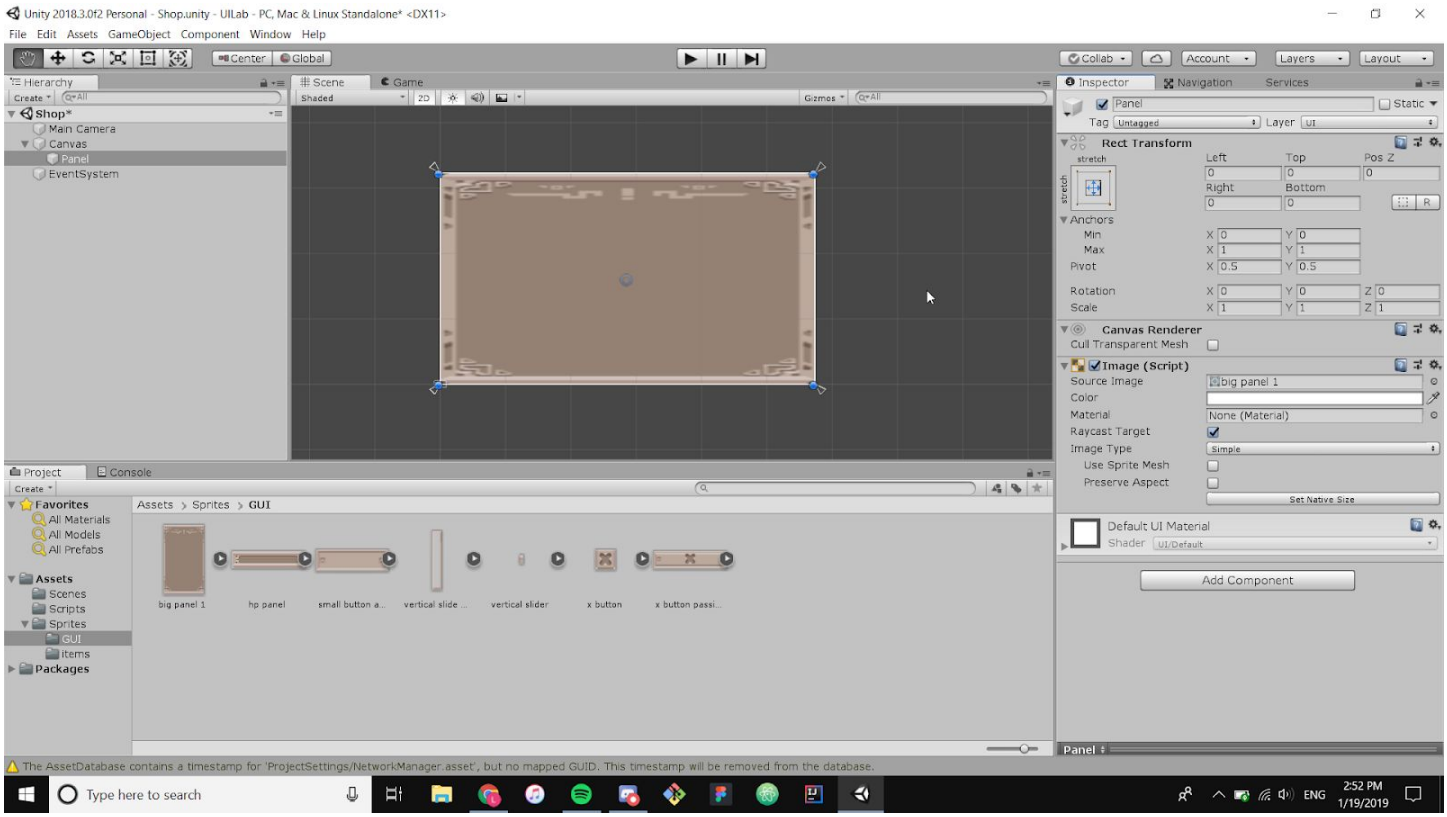
Now let's make our first UI element that we'll be able to see, the background for the shop. Select the Canvas in the hierarchy and create a new UI element called a Panel. This will be the background of the shop.



It should look something like this now. If you want to zoom or move the Scene around to get a better look at your work, you can right-click and drag to move, and also scroll in or out to zoom after clicking on Scene.

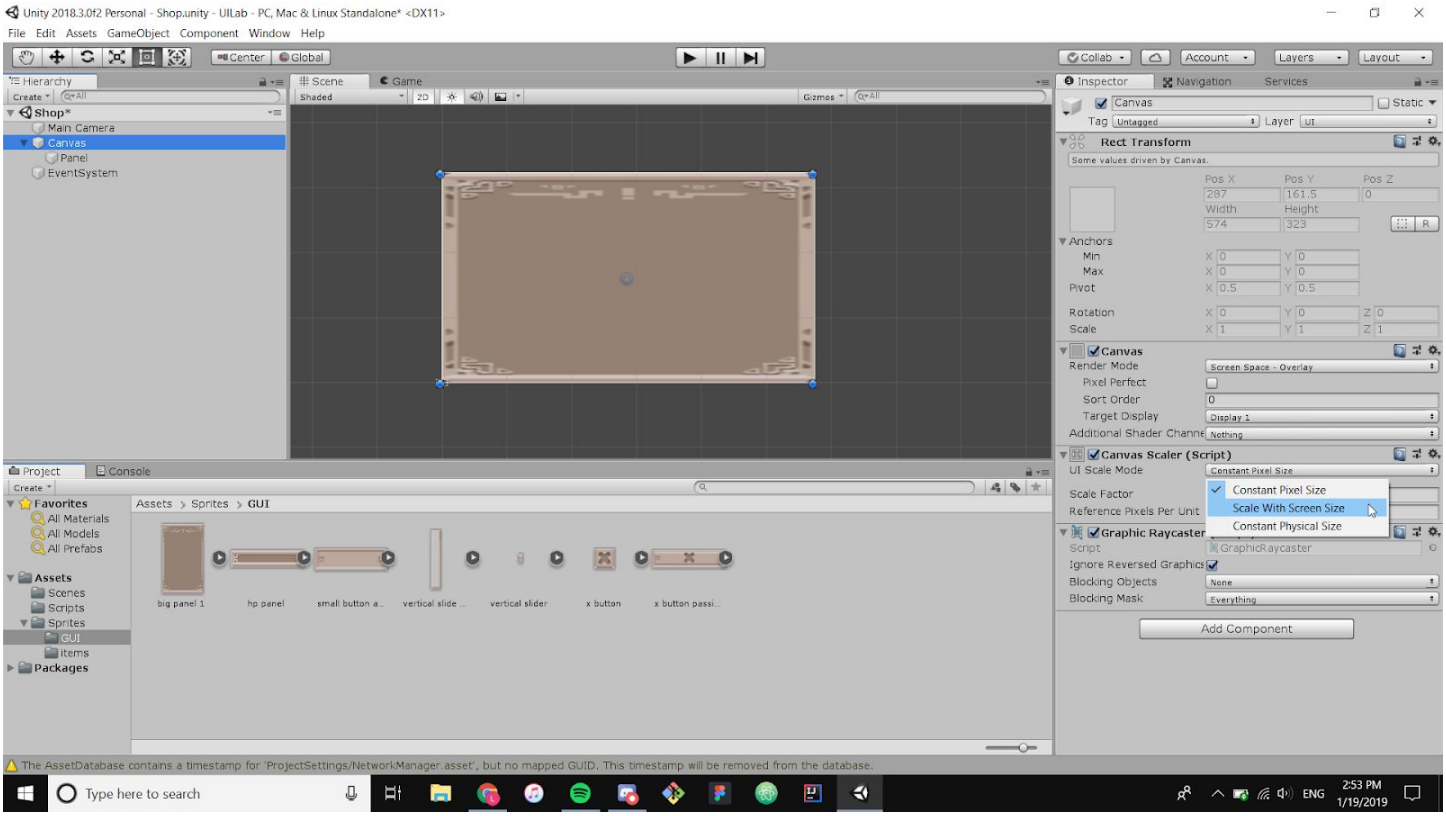


Select the panel in our hierarchy then go to inspector then “Image”. That’s the image that shows up as the panel. Go to the Sprites > GUI assets folder and drag the sprite titled *big panel 1* into the Source Image part of the Panel object. (If your panel looks semi transparent, double click on the color field in “Image (Script)” in the inspector drag the Alpha(A) variable all the way up to max).

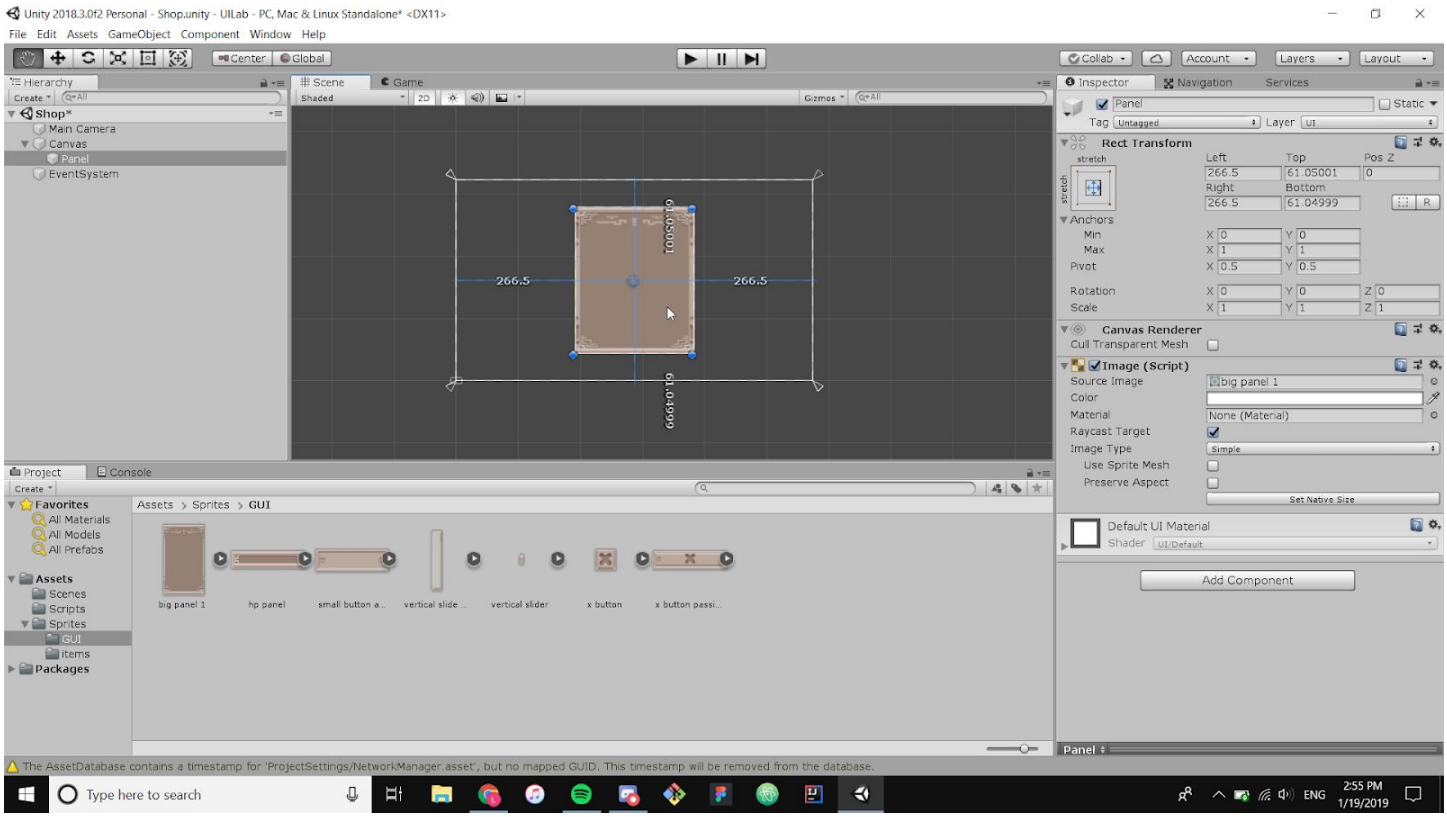


Now that we have something, let's resize the panel so it looks better. When you click on the panel on the scene, you should see 4 blue dots, 4 white arrows, and a blue circle in the middle. The blue middle circle is what marks the center of the panel and the axis of rotation. The 4 blue dots mark the panel boundaries/size scaling the image as you move around the dots. Finally the white triangles are the anchors, which help determine the position of the UI element. It is anchored onto its parent element, which is either the Canvas or another UI element (anything with RectTransform). The anchor can be moved by clicking, dragging or changing its presets in the RectTransform component.

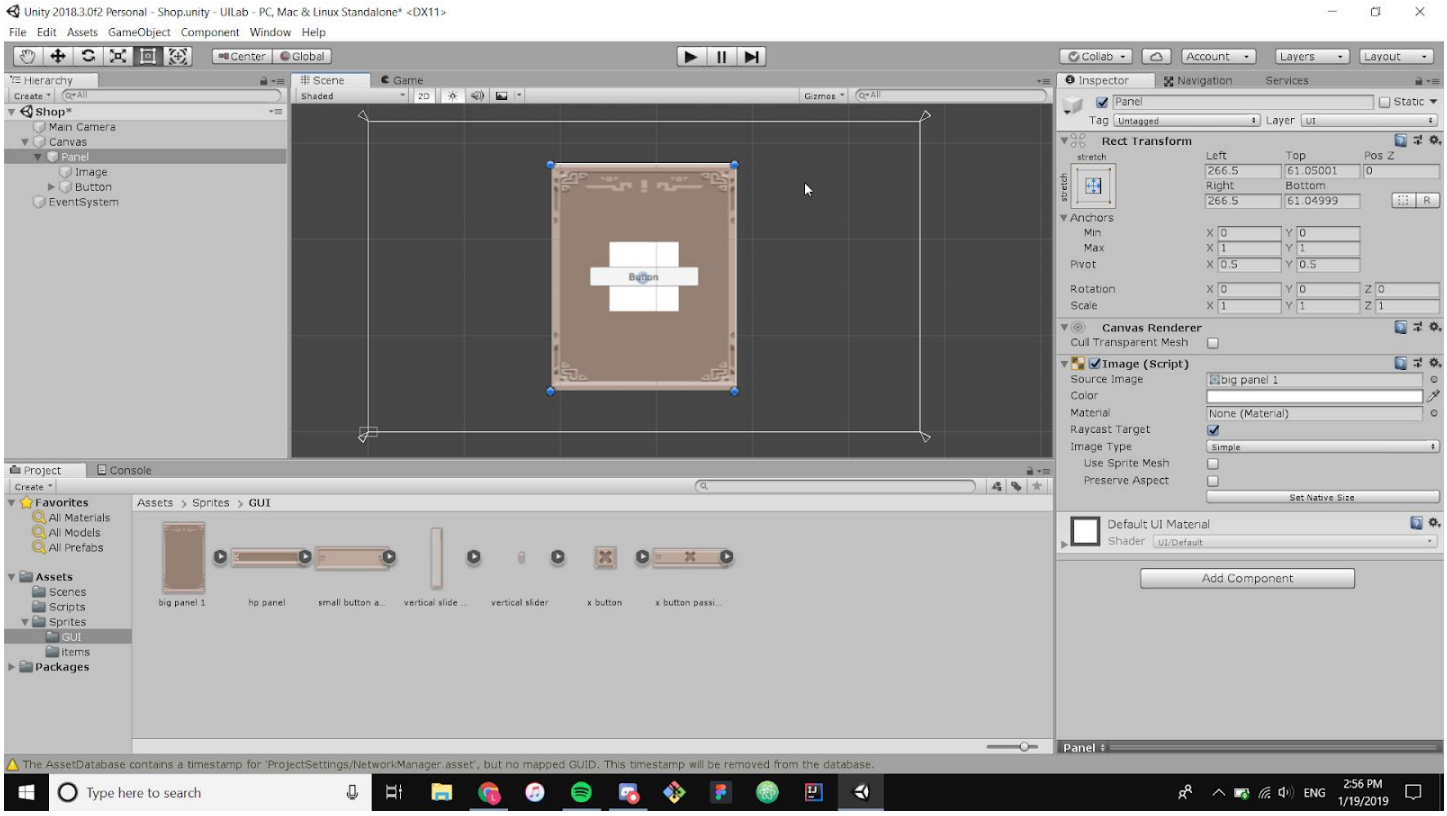
Let's make our shop background look a bit better by shortening the width of the object. First let's make sure that the images scale with the screen size, so that even with varying screen sizes, it will look very similar. Go to the Canvas object and under canvas scaler, change the UI scale mode from Constant Pixel Size to Scale with Screen Size.



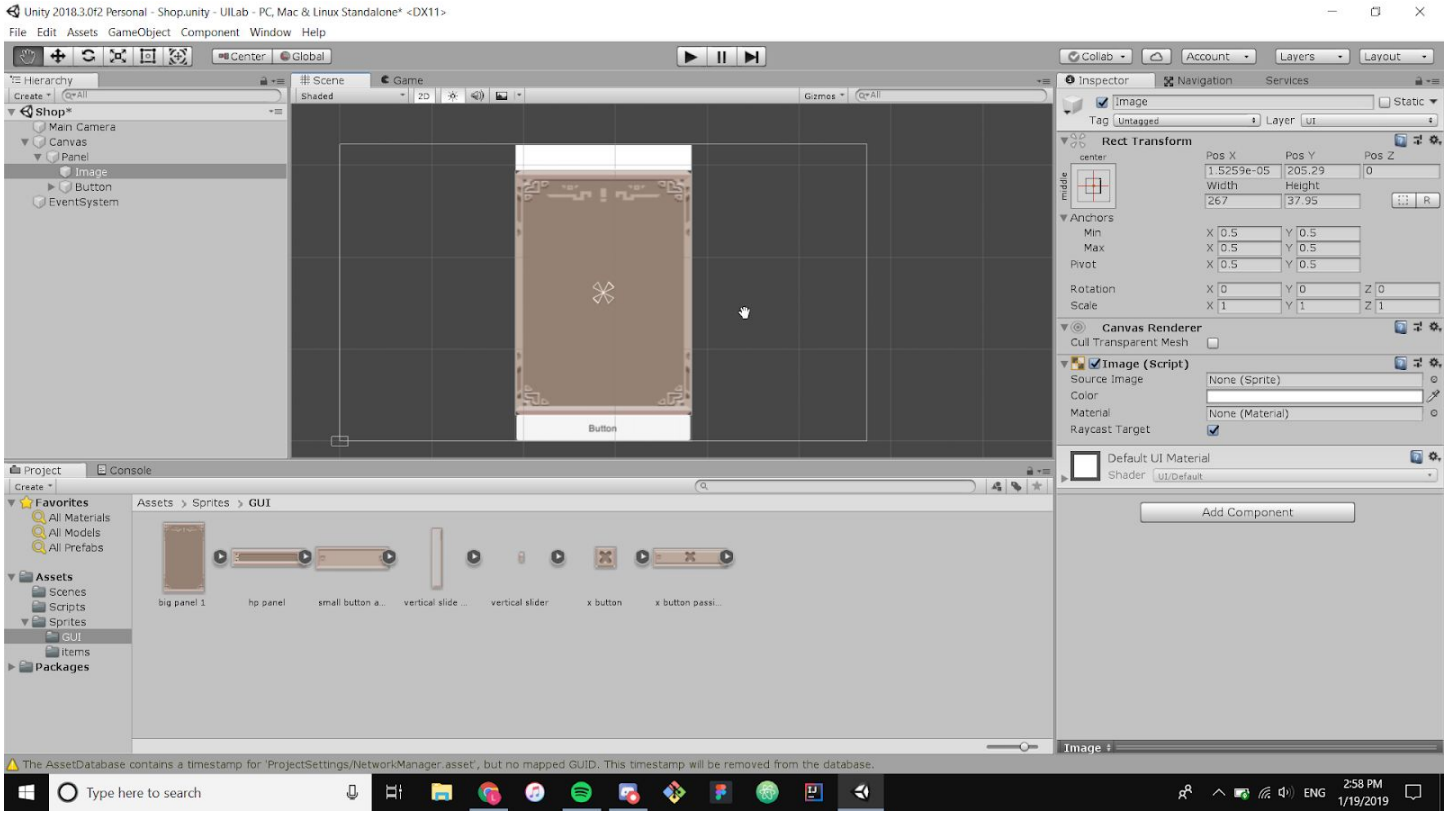
Now change the width and height of the panel to your liking as long as you leave some room at the top and the bottom for the title of the shop and exit button. Once you find a width you like, center it in the middle of the white triangles like so:



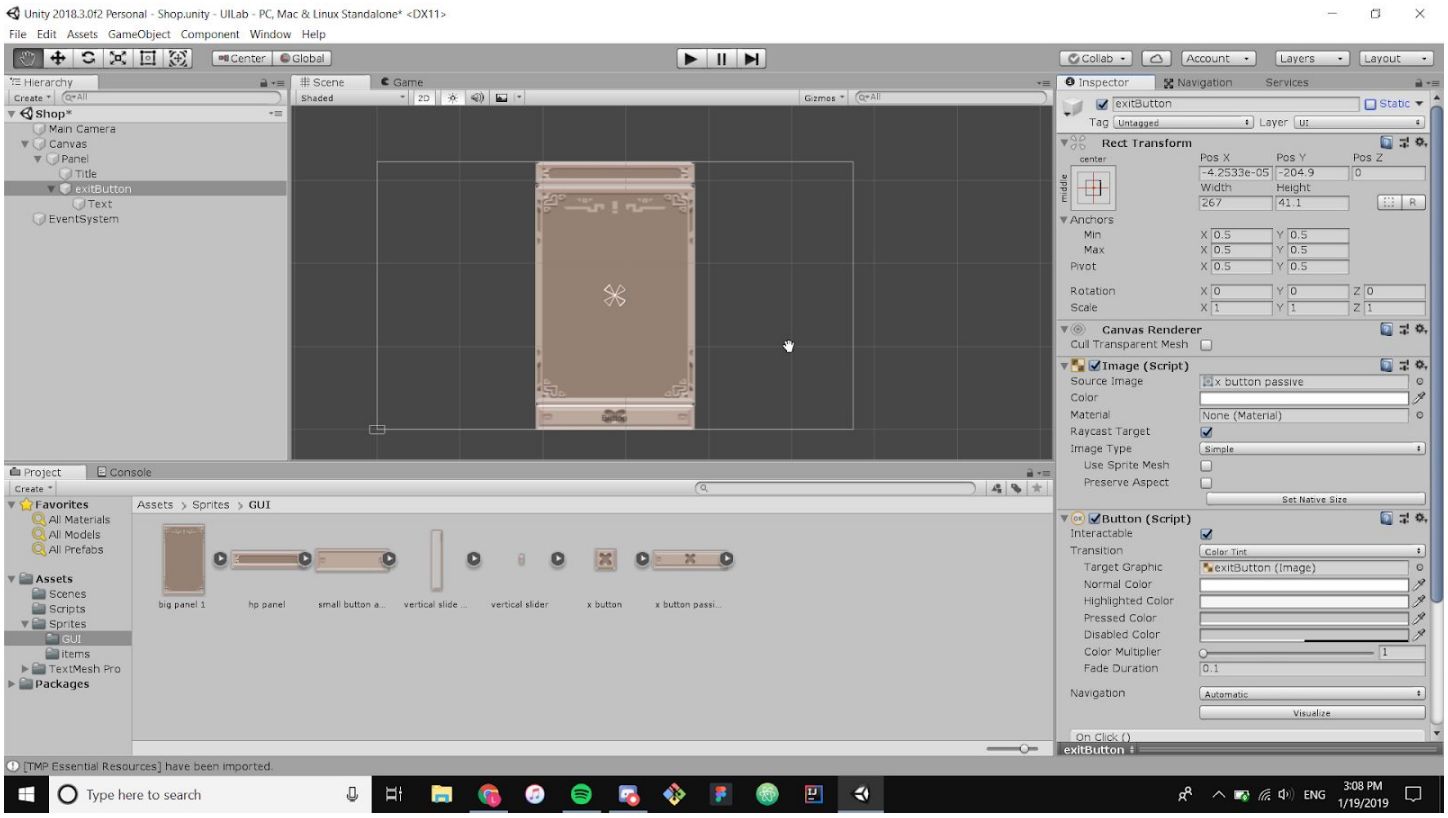
Now, let's add the title and exit button. Creating them as children of the panel, create a new UI image and a UI button. The button will be for the exit, and the image will be for the top shop name.



Drag the button to the bottom and the image to the top and fit it in the sections you left blank, matching the width with the width of the panel.

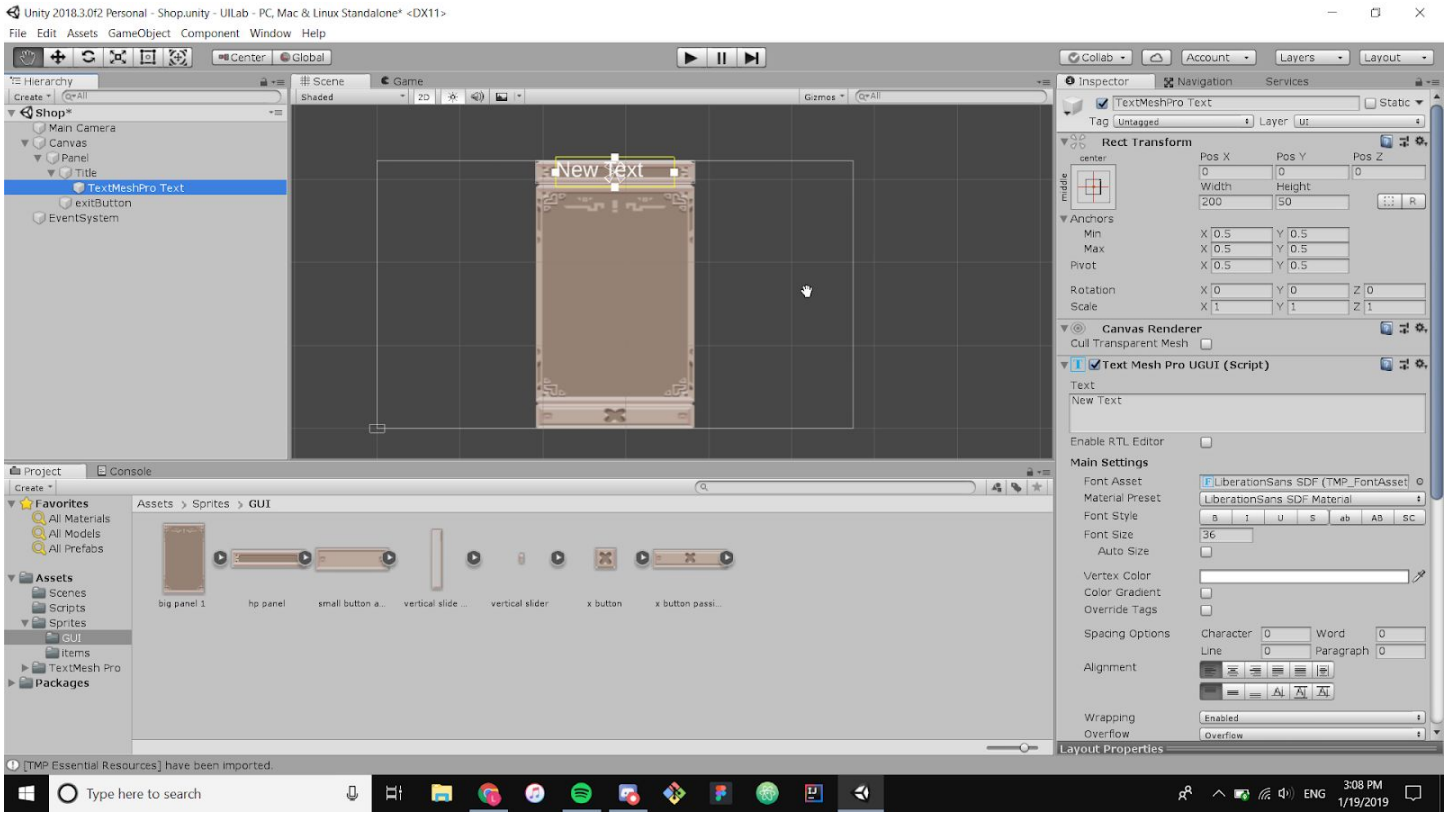


Now let's add the images for the title and exit button. Go back to the GUI folder and put the "hp panel" on image and the "x button passive" on the exit button's Source Image field in the Image(Script) component with the Inspector like you did before with "big panel 1" and the canvas. Let's also name the objects accordingly. You canvas should end up looking similar to this.

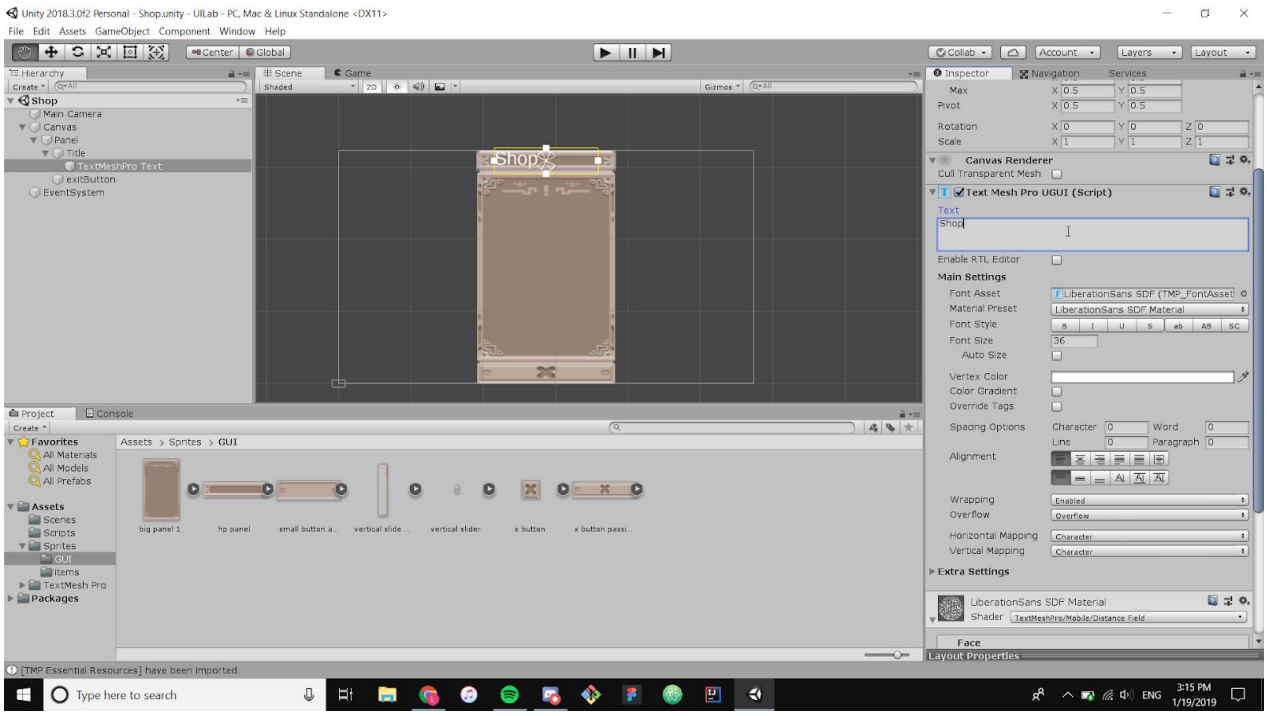


Let's take a minute to refine what we have on the menu. Add a shop name, and expand the button on the hierarchy and right click and delete the child of the button called Text. Right click on Title and add a new UI component called TextMeshPro - Text.

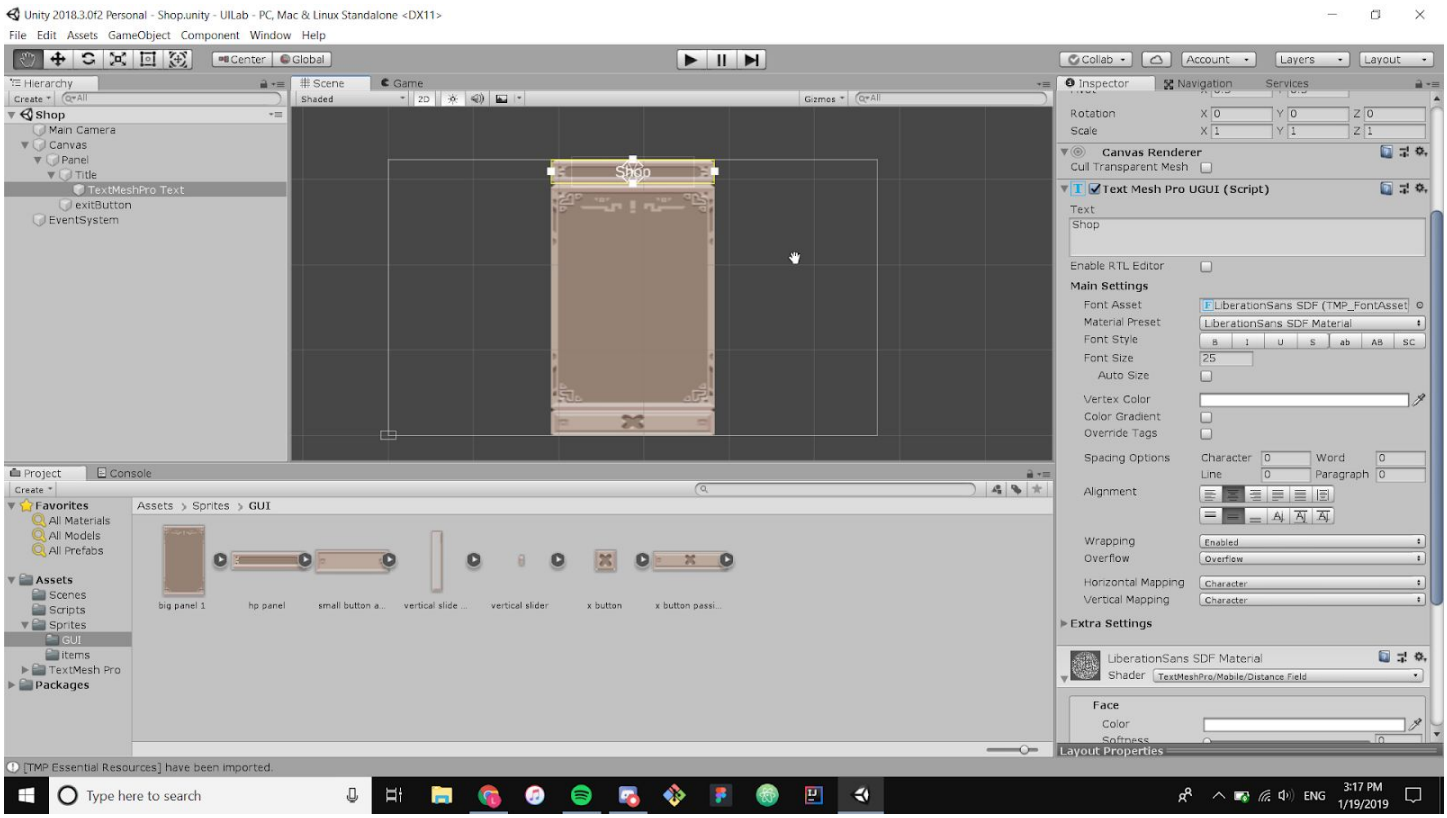
TextMeshPro is a replacement for Unity's default text option. To put it briefly, it solves many of the problems that Unity's default text has and lets you do a lot more. If you're interested, here's a short article about the benefits: https://blogs.unity3d.com/2018/10/16/making-the-most-of-textmesh-pro-in-unity-2018/. If this is your first time using TextMesh Pro, then go to Window > Package Manager and search for "TextMesh Pro" and Install it. Once it's installed, it will appear in your UI options.



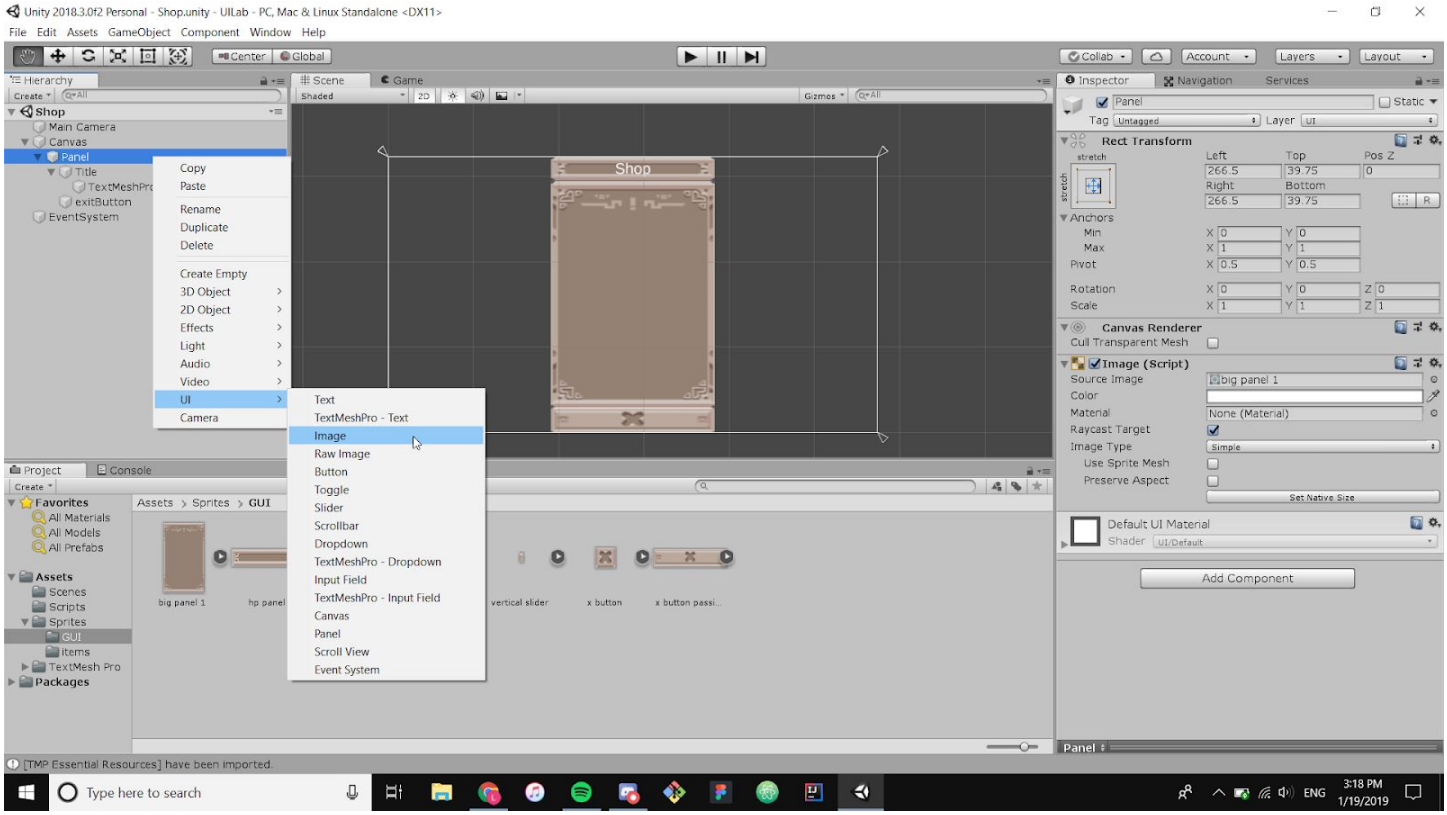
If you select the text, you should see it in the scene. You edit the text through the Inspector. There should be a box in the Text Mesh Pro component that has "New Text" in it. Whatever is typed in there will be reflected in the game. I will just be naming my shop "Shop", however you can name it anything you like.



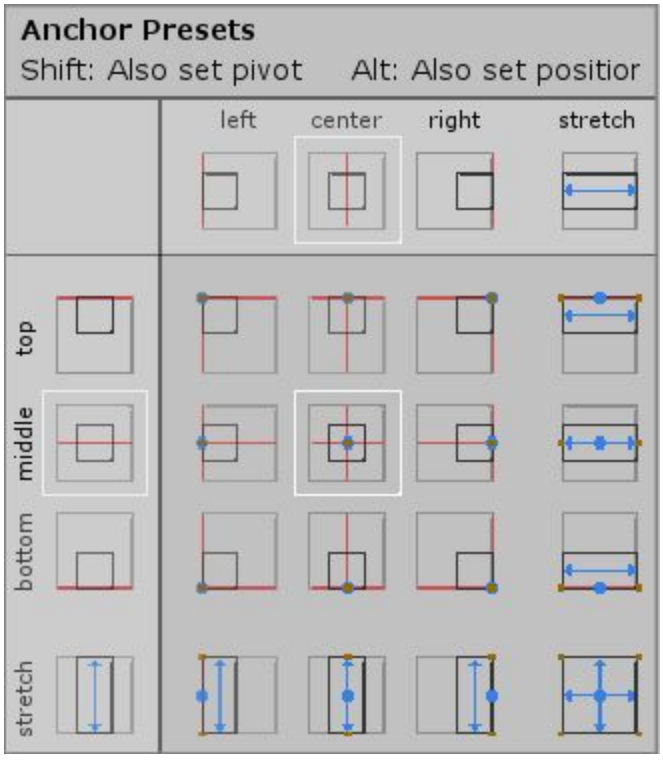
Now let's make it centered and more visible in the game. First, change the dimensions of the text to the inside of the title image. Then look further into the Text component and there should be a section called Alignment. Select the middle alignments; this should center the text in both directions. Now, change the font size to something that looks appropriate. Also, feel free to change the font color to anything you want with the Vertex Color option. Now it should look something like this:



Let’s start setting up the items to be sold in the shop. Make a new image element as a child of panel and name it spacer.

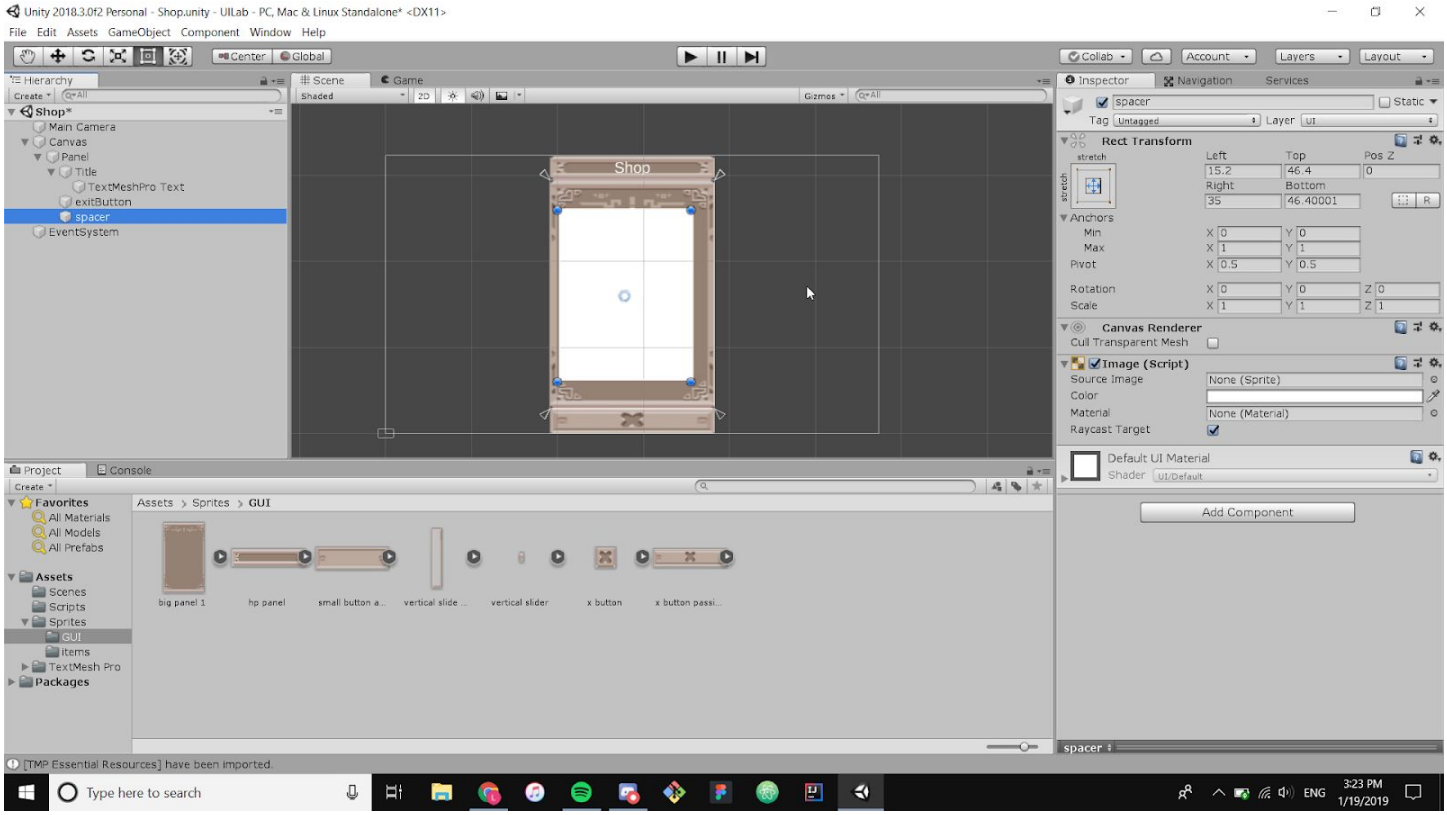


Now let’s change the anchor presets of this image so that it doesn’t extend past the panel itself. Select spacer and look under the Rect Transform component in the inspector and click on the box that looks like  on the top left corner. Hold down Shift and Alt at the same time and select the bottom right configuration (holding down shift also sets the pivot - which point the image rotates on, and holding down alt sets the position of the element).

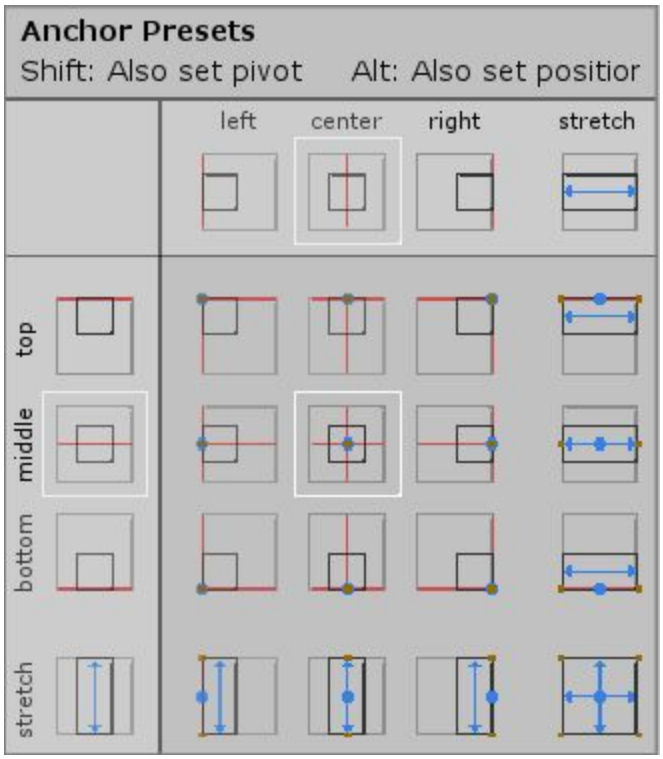


<- this option

This should cause the image to cover the whole panel. Resize it to cover the section of the panel where you want to display the items of the shop. Be sure to leave some room on the right for the scroll bar:

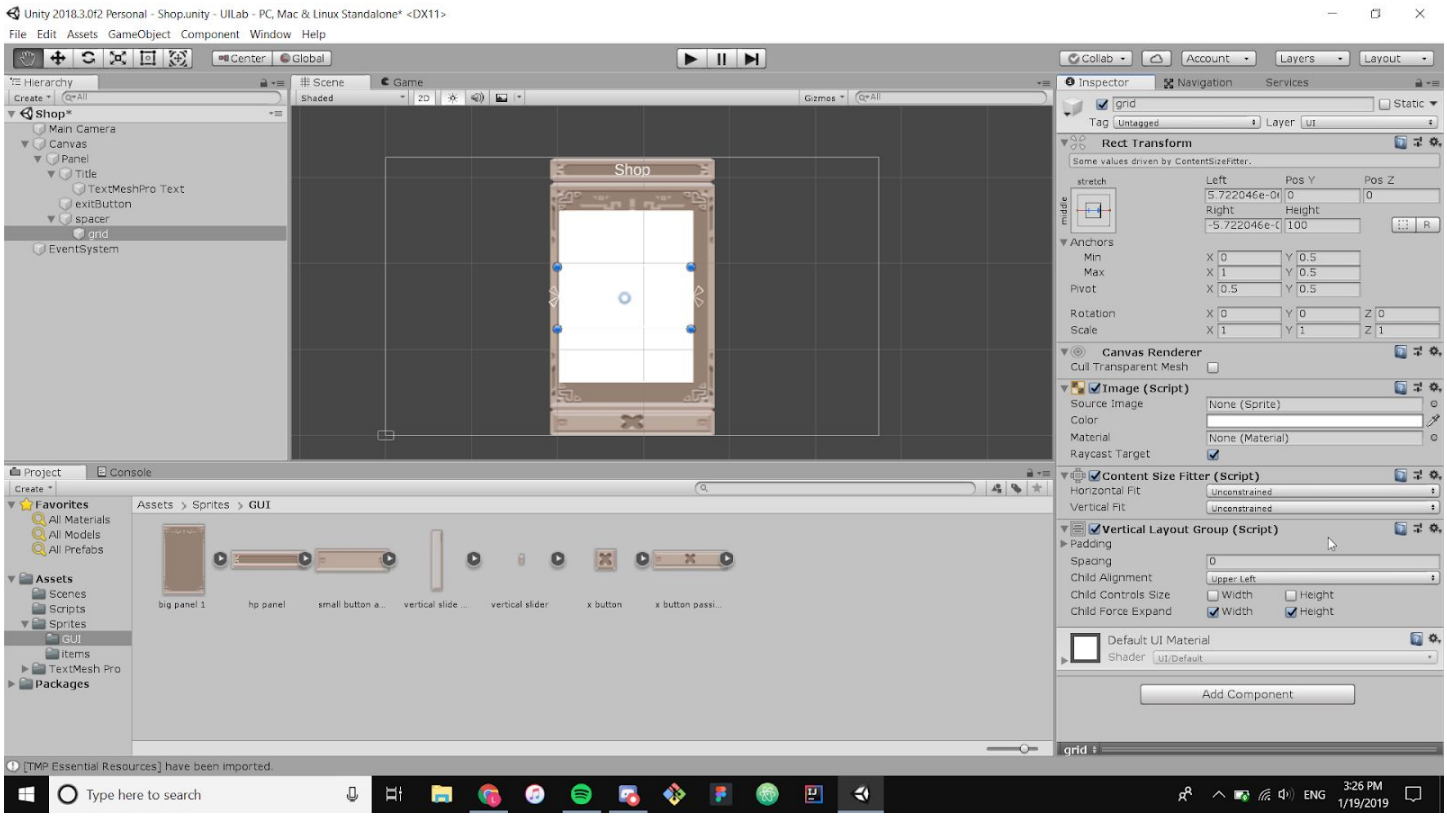


Now, make another Image that is a child of spacer. Name this Grid. This is the grid of all items that are going to be sold. Use rect transform on this image as well except this time choosing this option on the rightmost column (Again don't forget to hold down shift and alt):

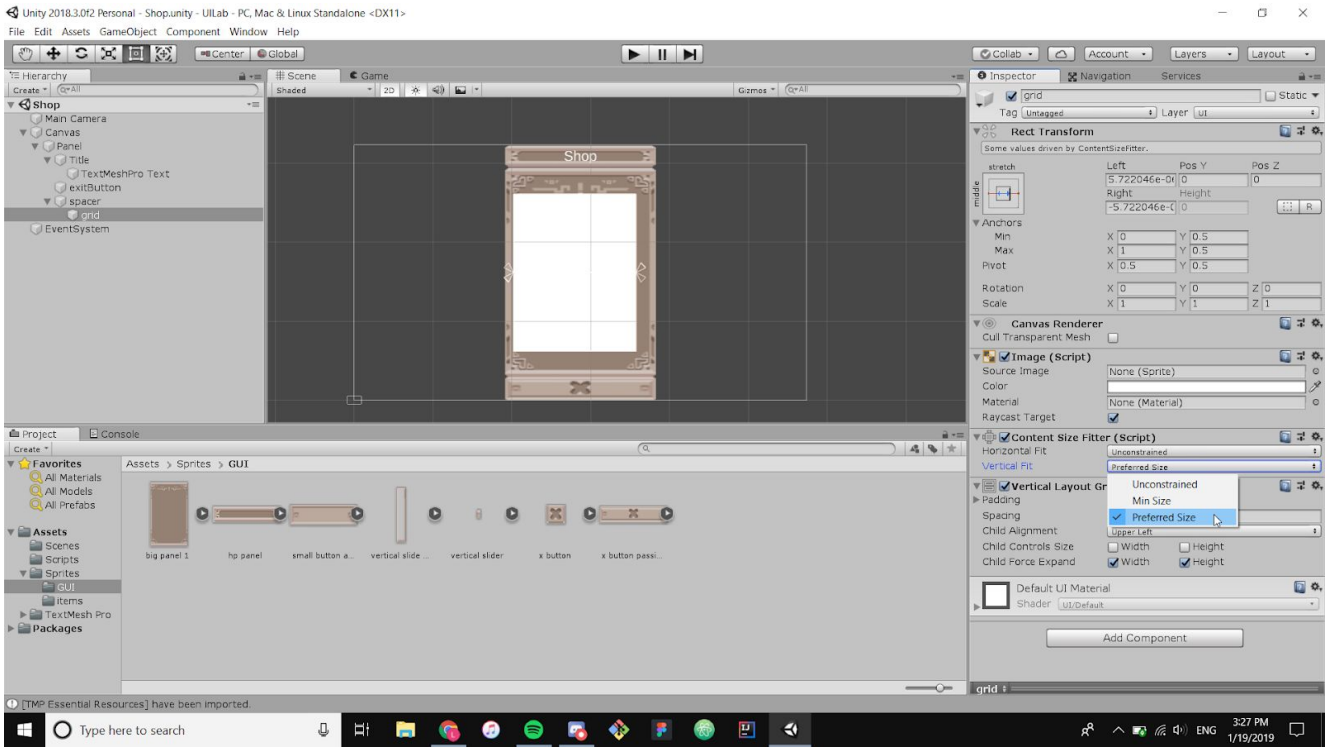


<- this option

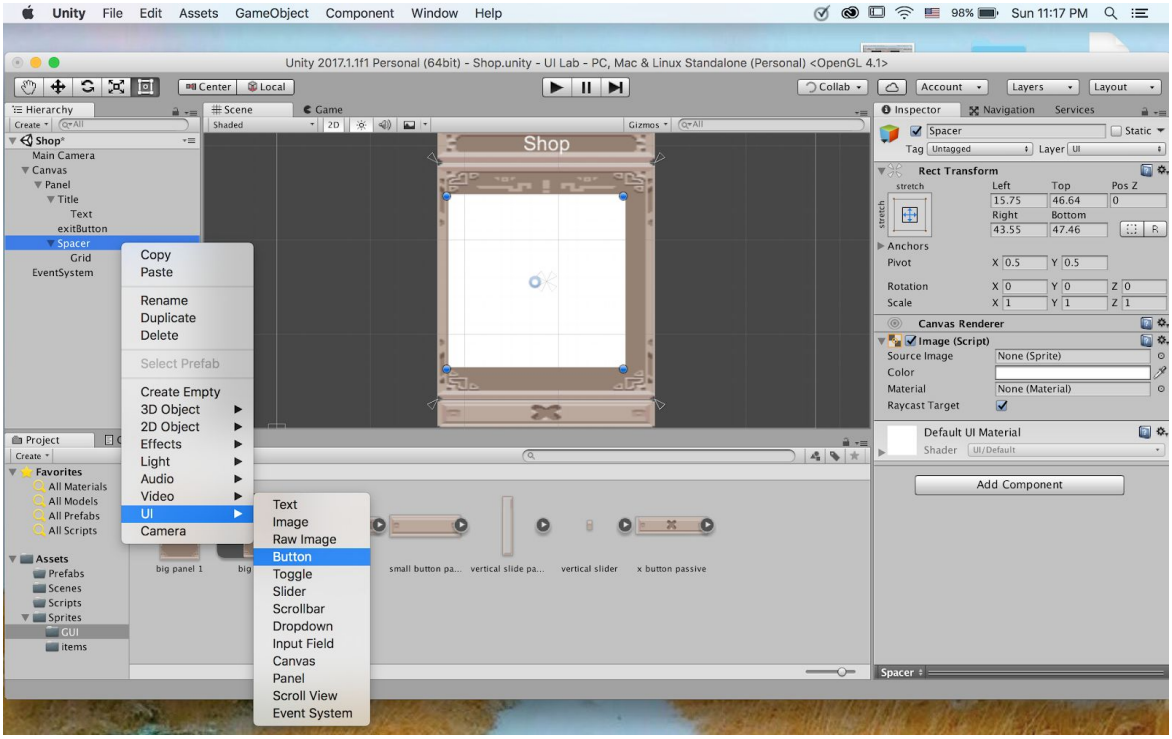
Now add two new components to grid in its inspector: Content Size Fitter and Vertical Layout Group. Both components are used to allow the grid of items to grow as you add more things that can be bought. Content Size Fitter will change the size of the grid as you add things, whereas Vertical Layout Group will help you order things as you add things to the grid.



You don’t have to change anything in Vertical Layout Group, but in Content Size Fitter, change the Vertical Fit to be Preferred Size instead of Unconstrained. This will cause the width of the image to be 0 for now, but as you add things to the grid, it will grow with the items.

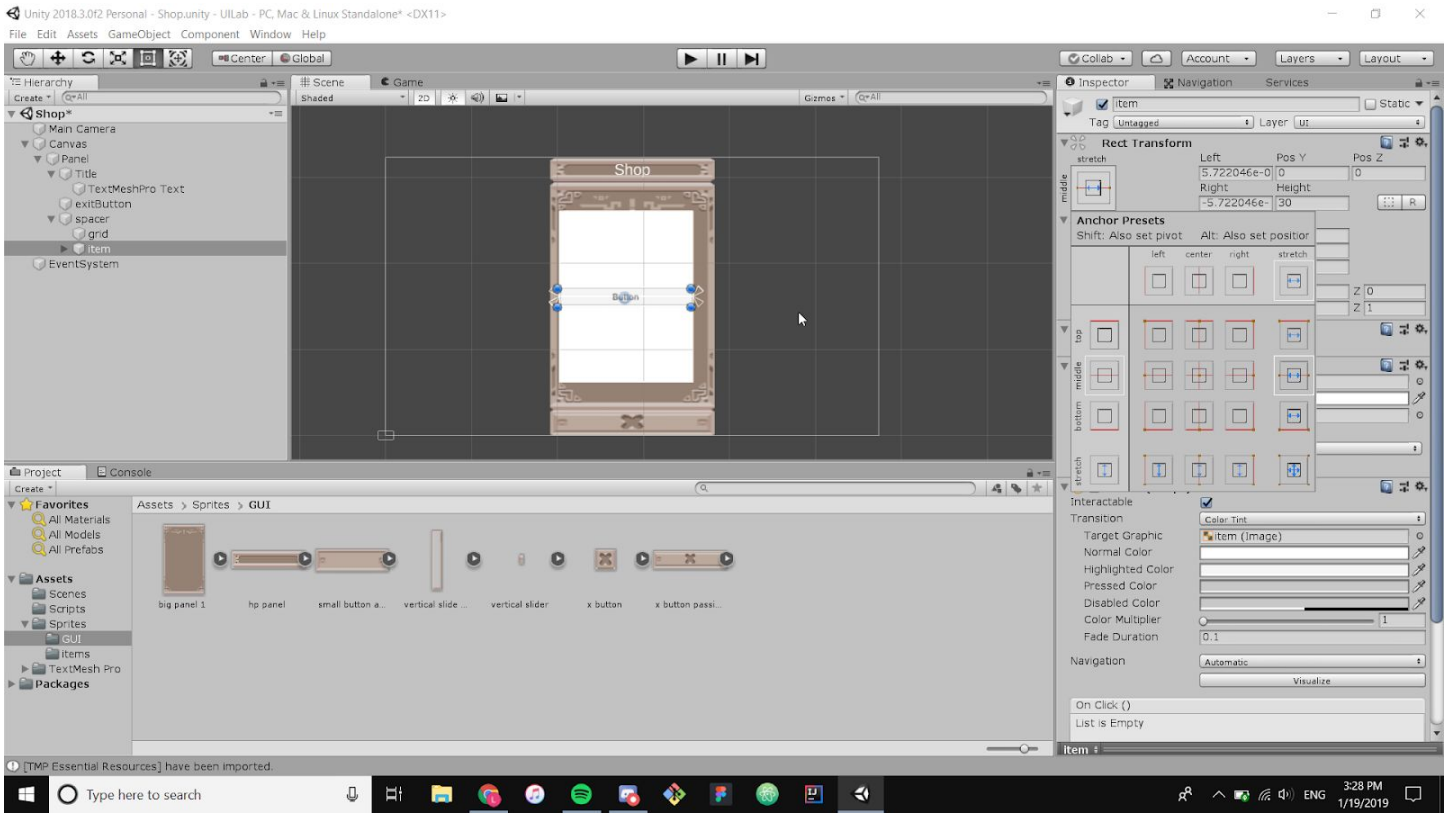


Now in Spacer, add another UI element, but this time a button and name it Item (Make sure you're making it a child of Spacer, not Grid).

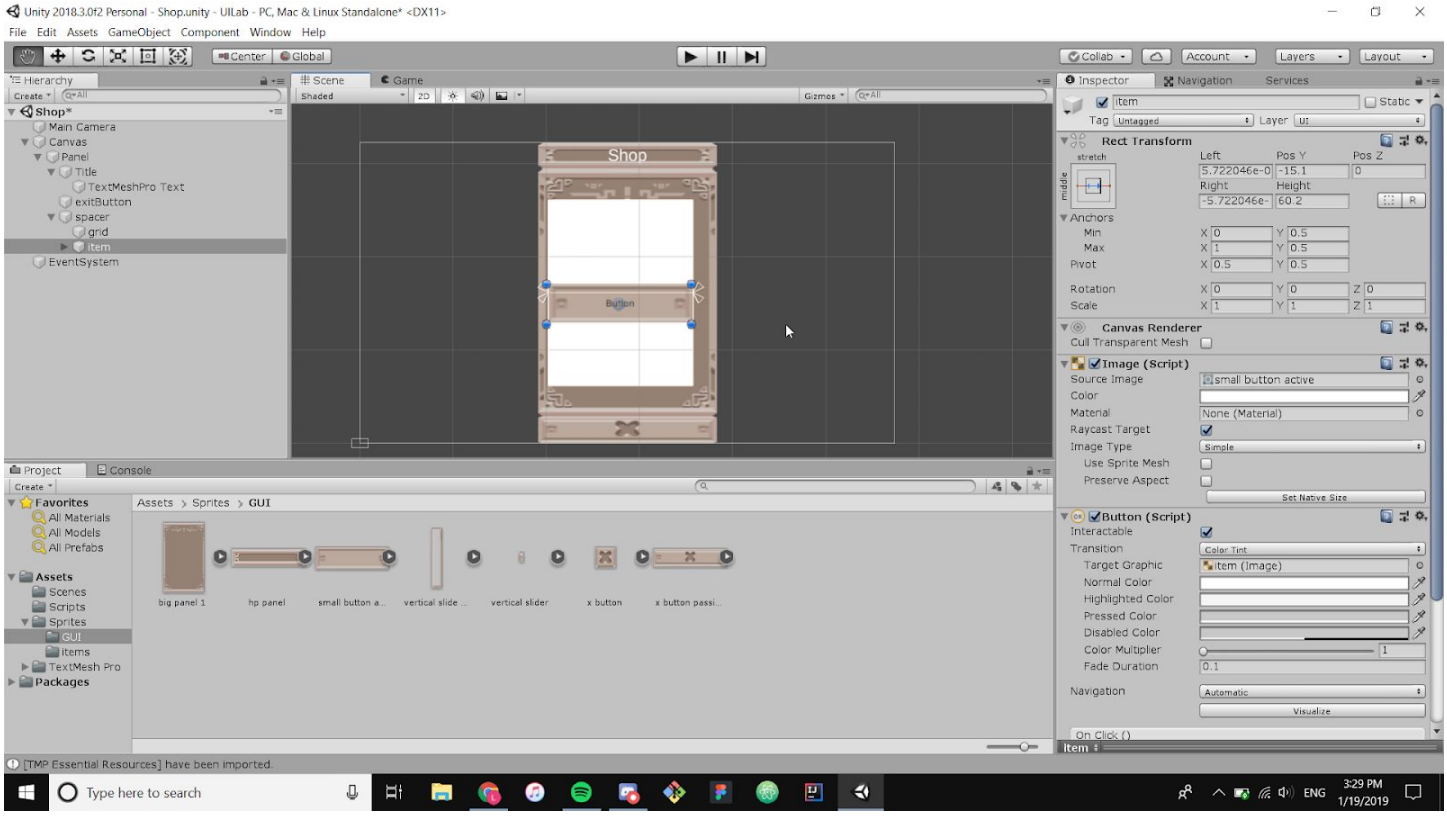


This button will be how you can buy the item, and should show multiple things: a picture of the item, the name of the item, and the price/whether you have bought this item or not. Qe are going to be trying to make a Prefab out of this, so that we can easily add as many as we need.

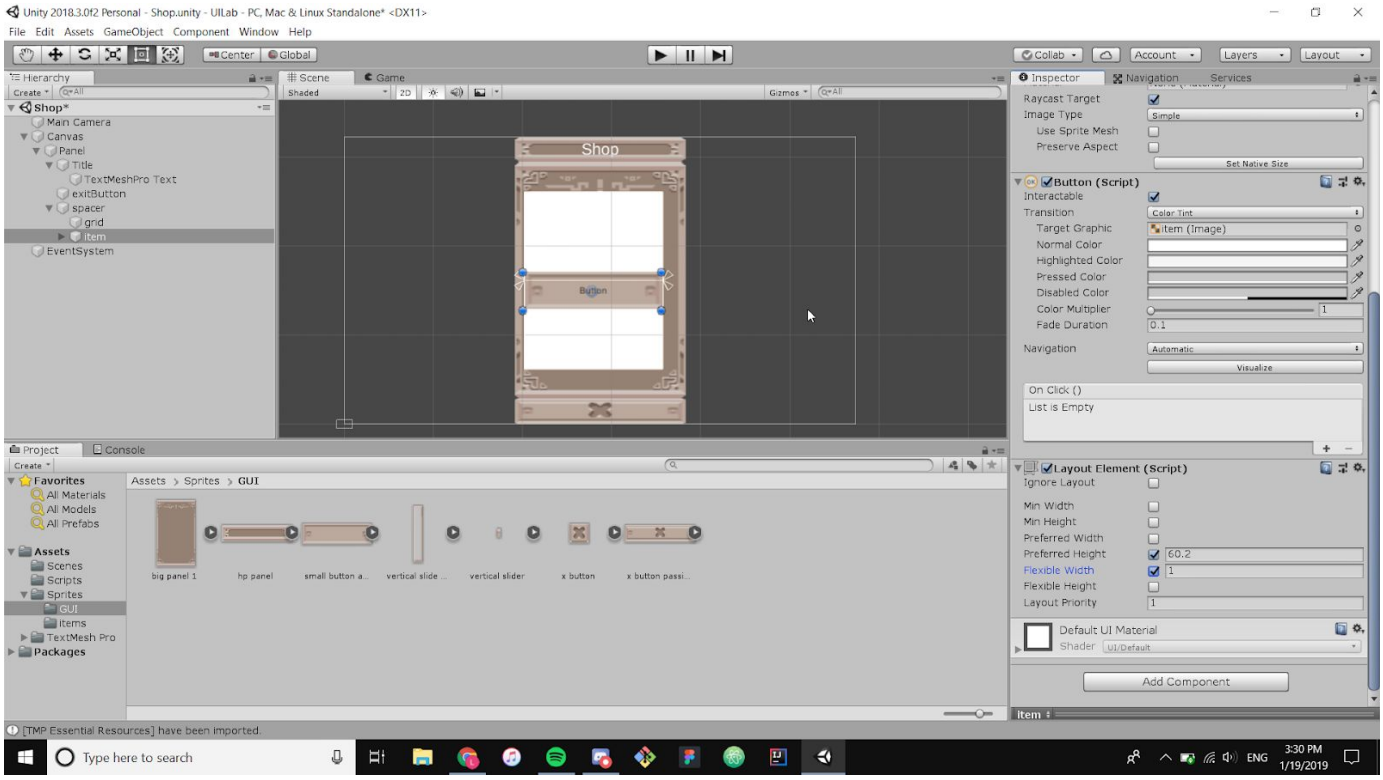
To make it into a Prefab, first let's format the button according to the scene. Use Rect Transform to shift the anchors and pivot point. Select the same option as last time (Don't forget to hold down shift and alt).



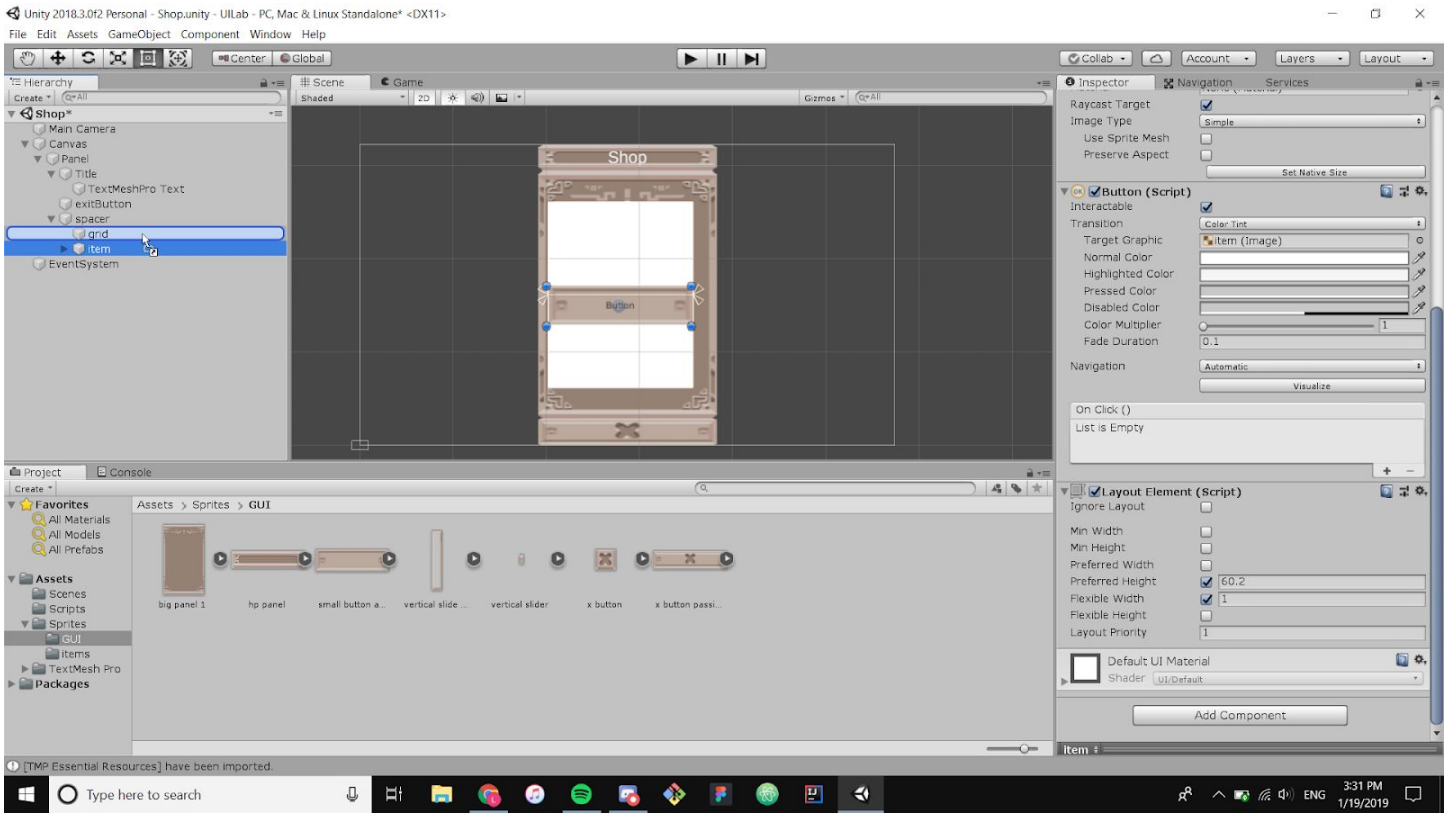
Now add the UI Sprite Small Button to the image component to add an image to Item, and change the height of Item so that it can hold all the things we need.



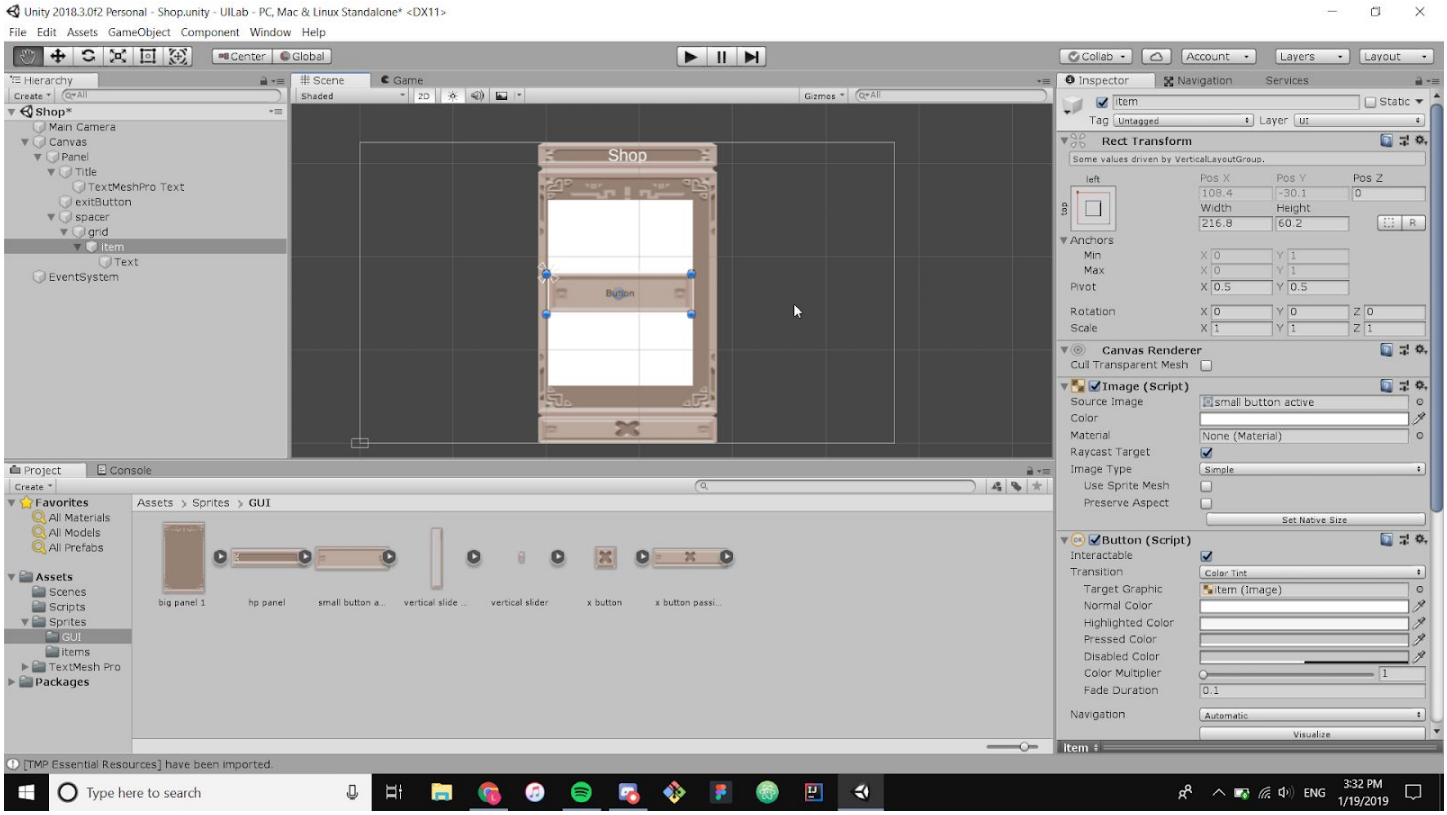
Now let’s add the component that will allow us to layout the buttons nicely. Add the component called Layout Element to Item and check Preferred Height and Flexible Width; preferred height will cause the default height of each button to be the one that we set here, and the flexible width will cause the width of the button to correspond to the grid. Remember the Vertical Layout Group component we added to the grid? This component will correlate with that component allowing the Grid to grow and be laid out correctly.



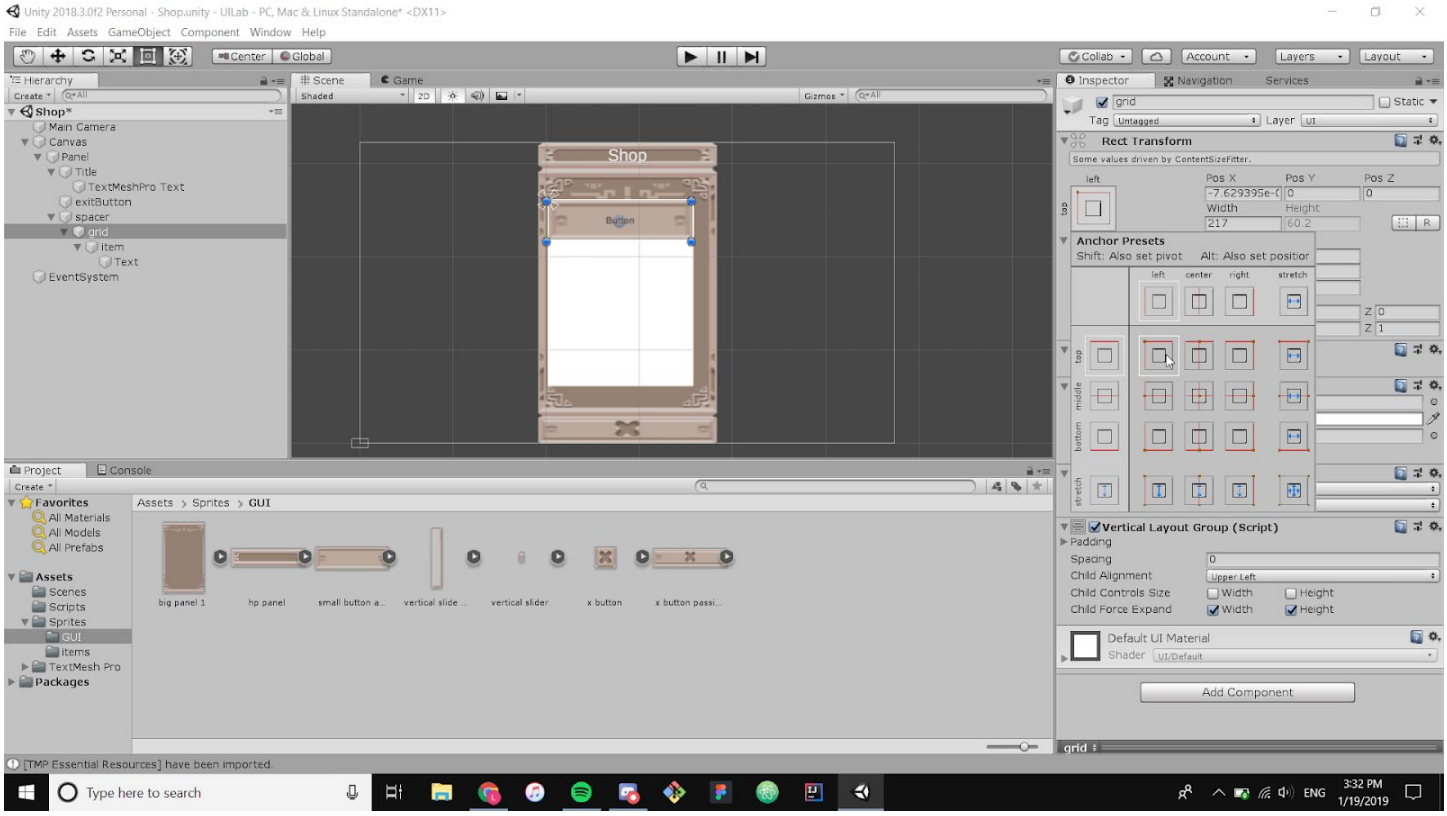
Now move item to be a child of grid:



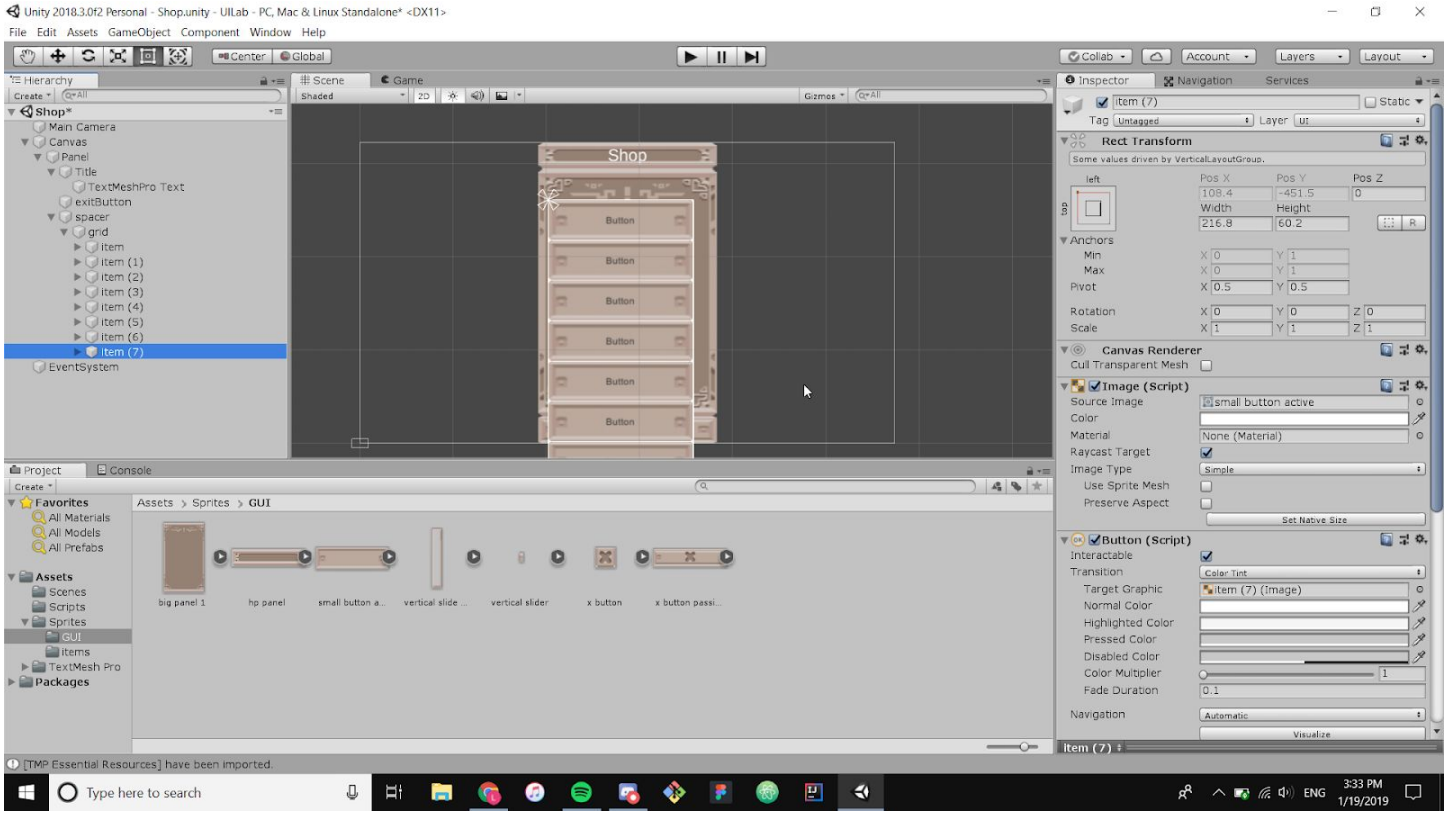
You shouldn't be able to see the button. This is because the width of the button is 0. Click on Item and Set the width to be greater than 0 (which you can do in the rect transform component), and then drag it to the width of the spacer (the white box we made previous)



Now let's set the Grid to the top left corner of Spacer. Select Grid and click on the box in Rect Transform again. This time select this one (remember to be pressing down Shift and Alt while doing this):

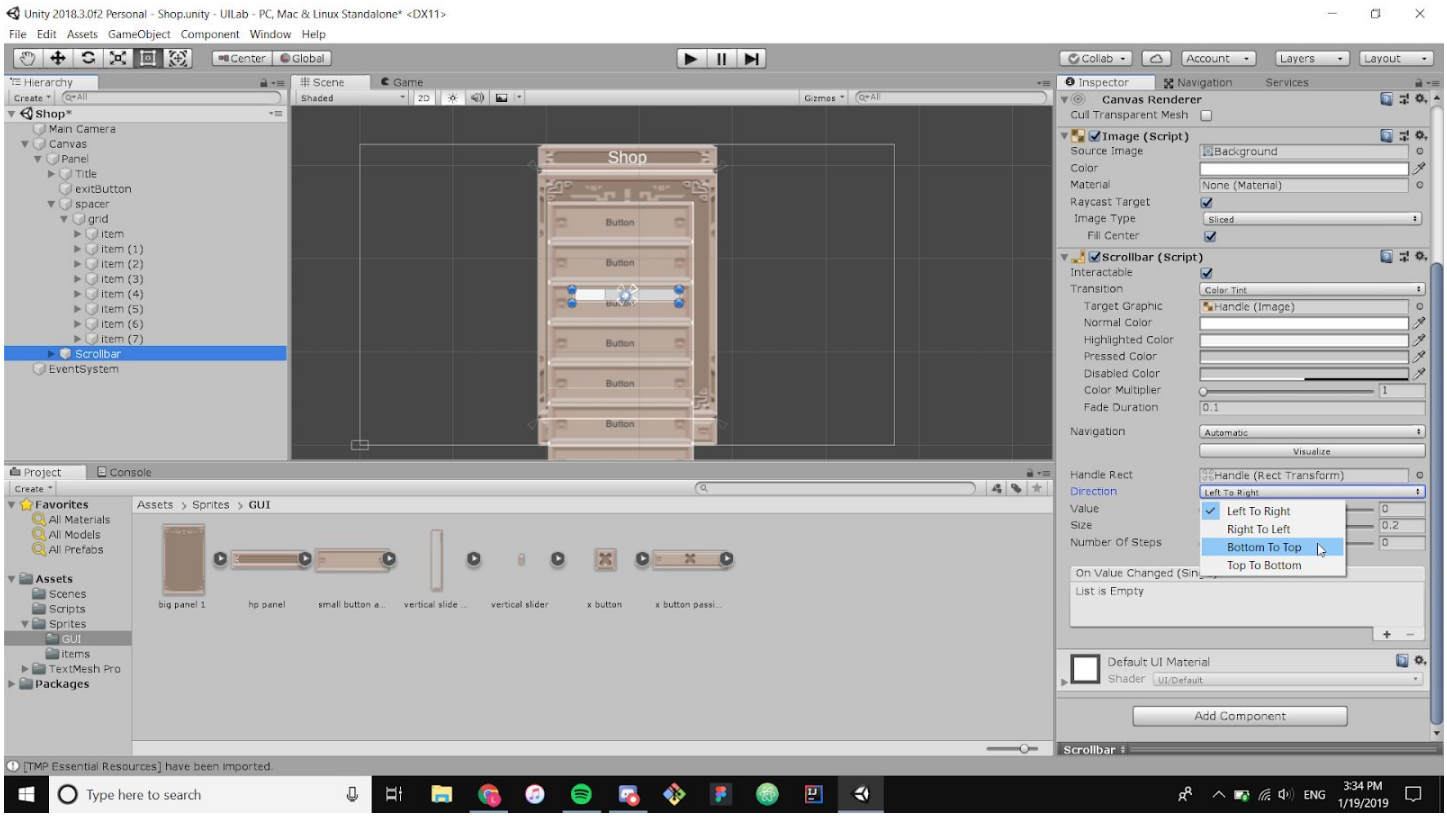


You can check if you did this correctly by selecting Item and copying it and pasting it. As you paste, it should add more buttons below the last.

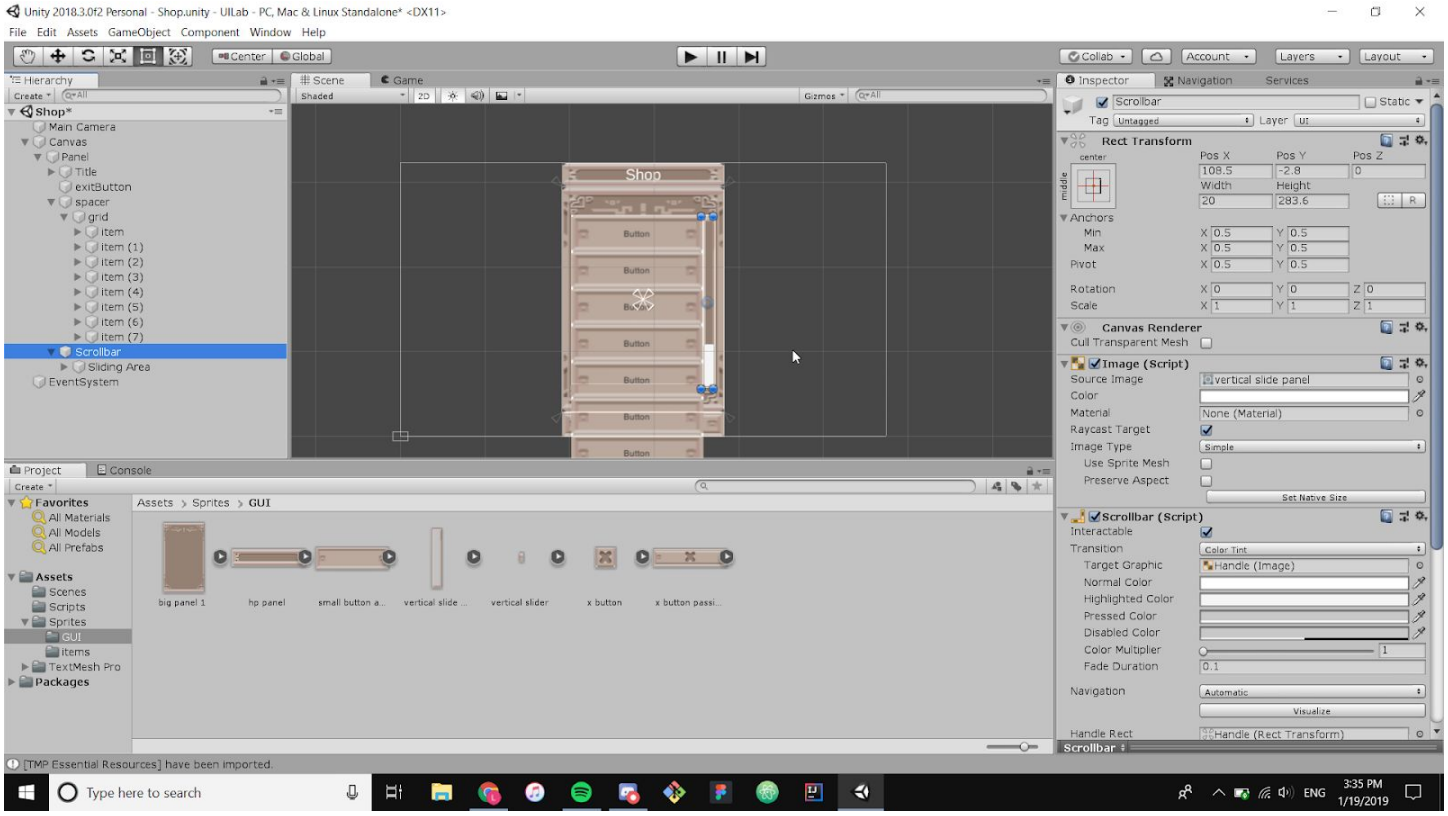


Keep all the extra buttons, as we're gonna add a scrollbar to control them now. Create a new UI element called Scrollbar, make it a child of Panel.

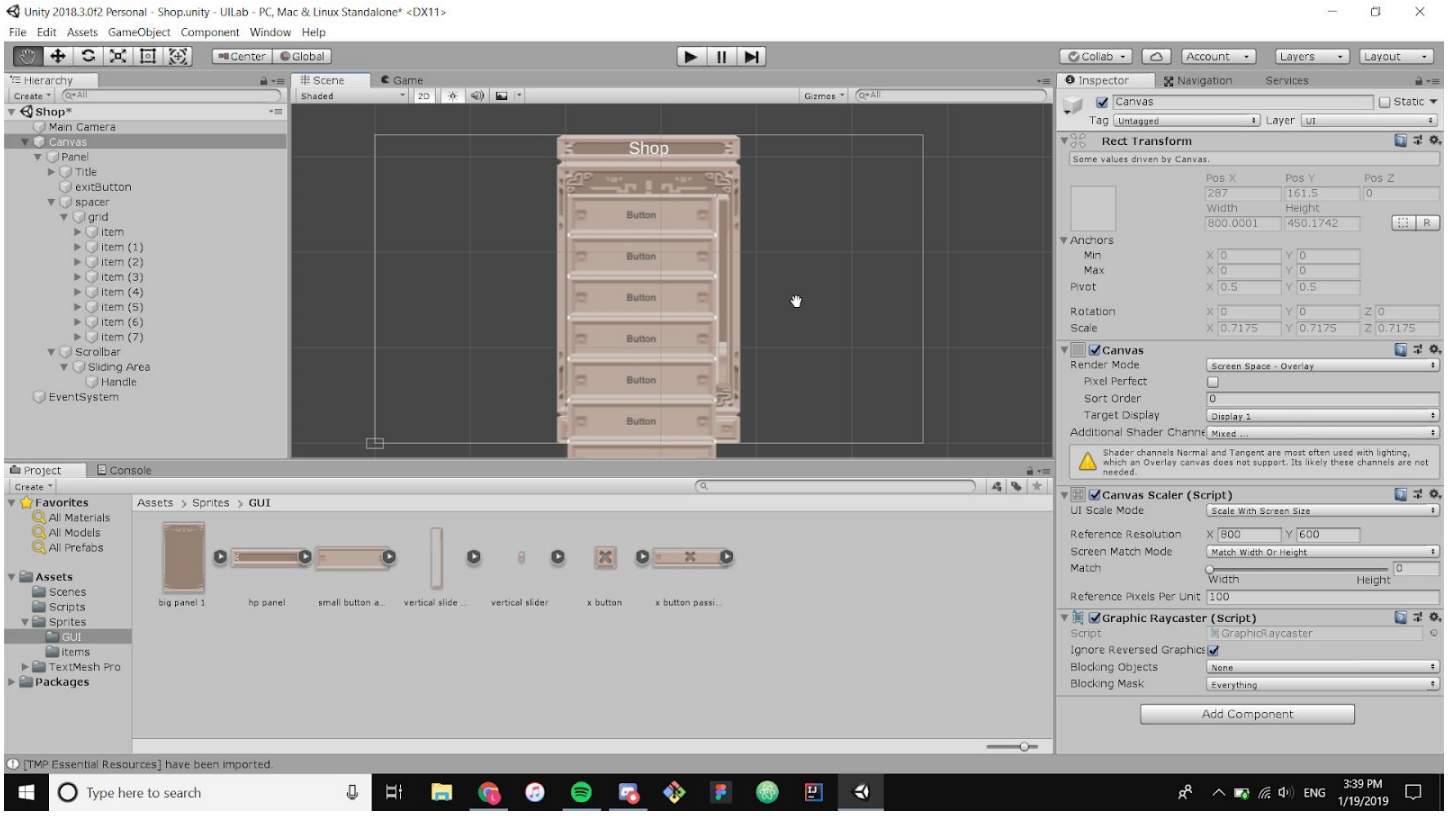
Go under the Scrollbar component and make the Direction go from Bottom to Top as shown in the picture below.



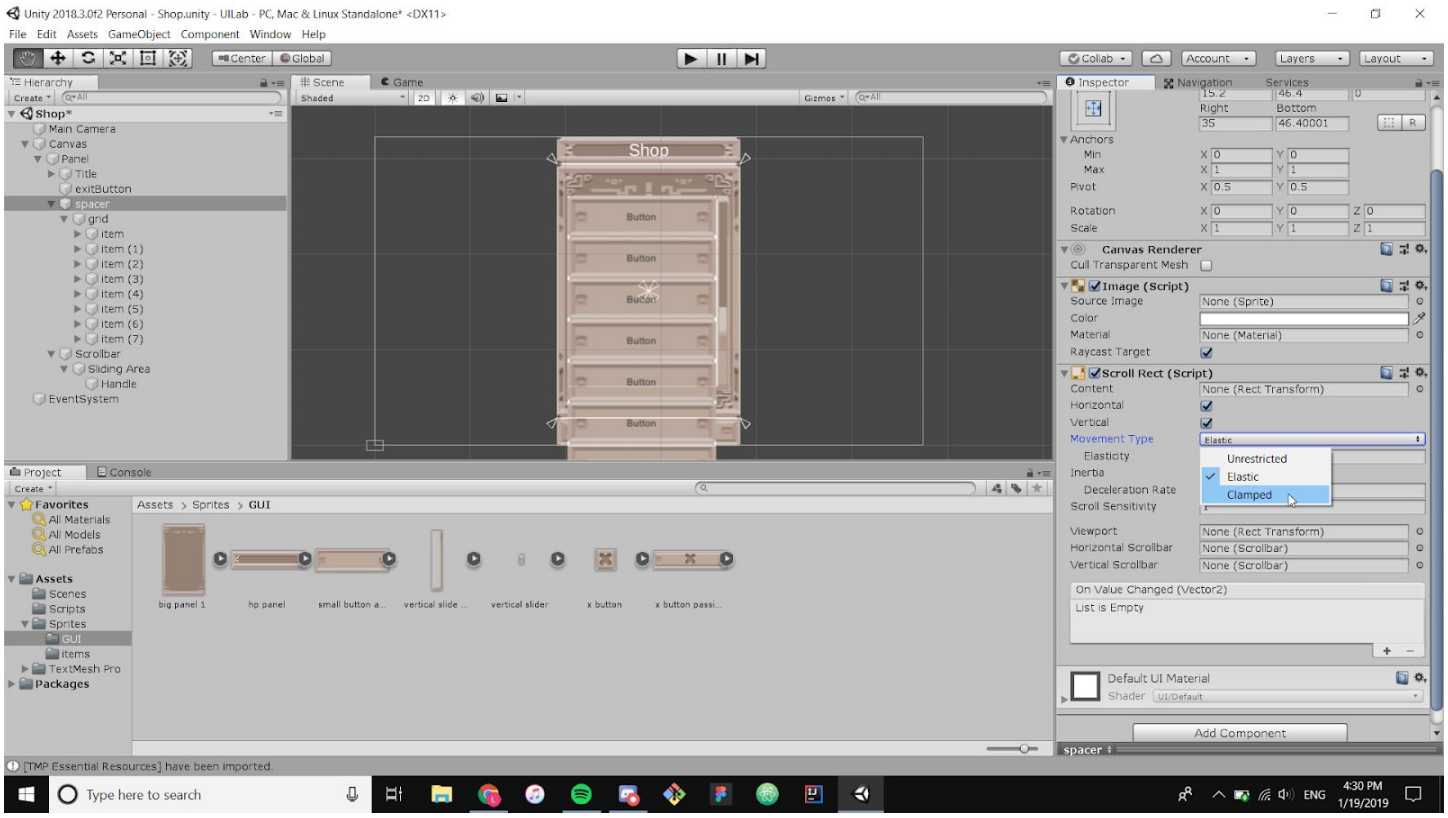
Move the Scrollbar to the right of the items and resize it to however you want. Attach the “vertical slide panel” Sprite to the Image component in the inspector



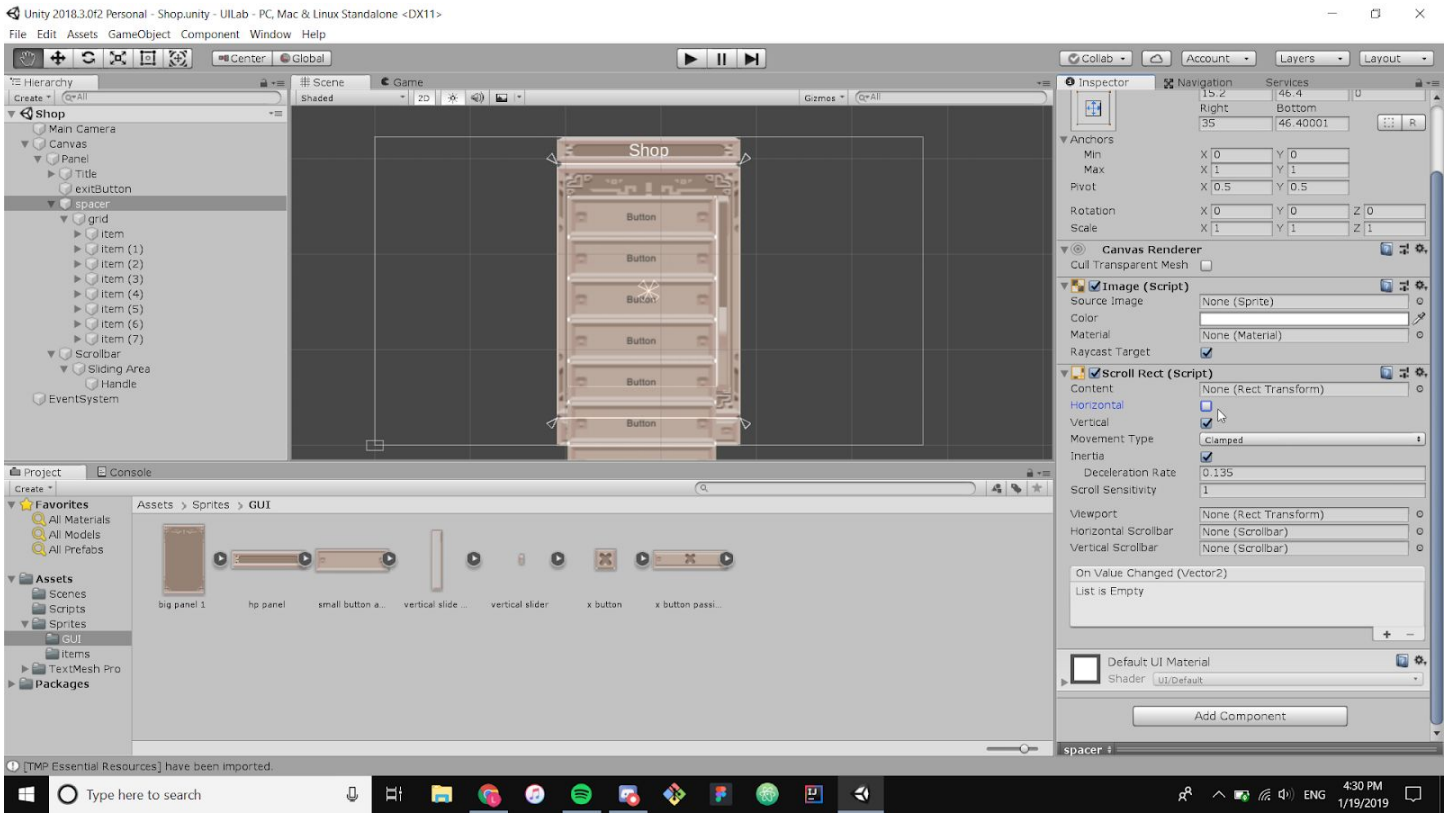
Go to the Handle GameObject, which is a grandchild of Scrollbar, and attach the vertical slider sprite to the image part. Then resize the handle to your discretion.



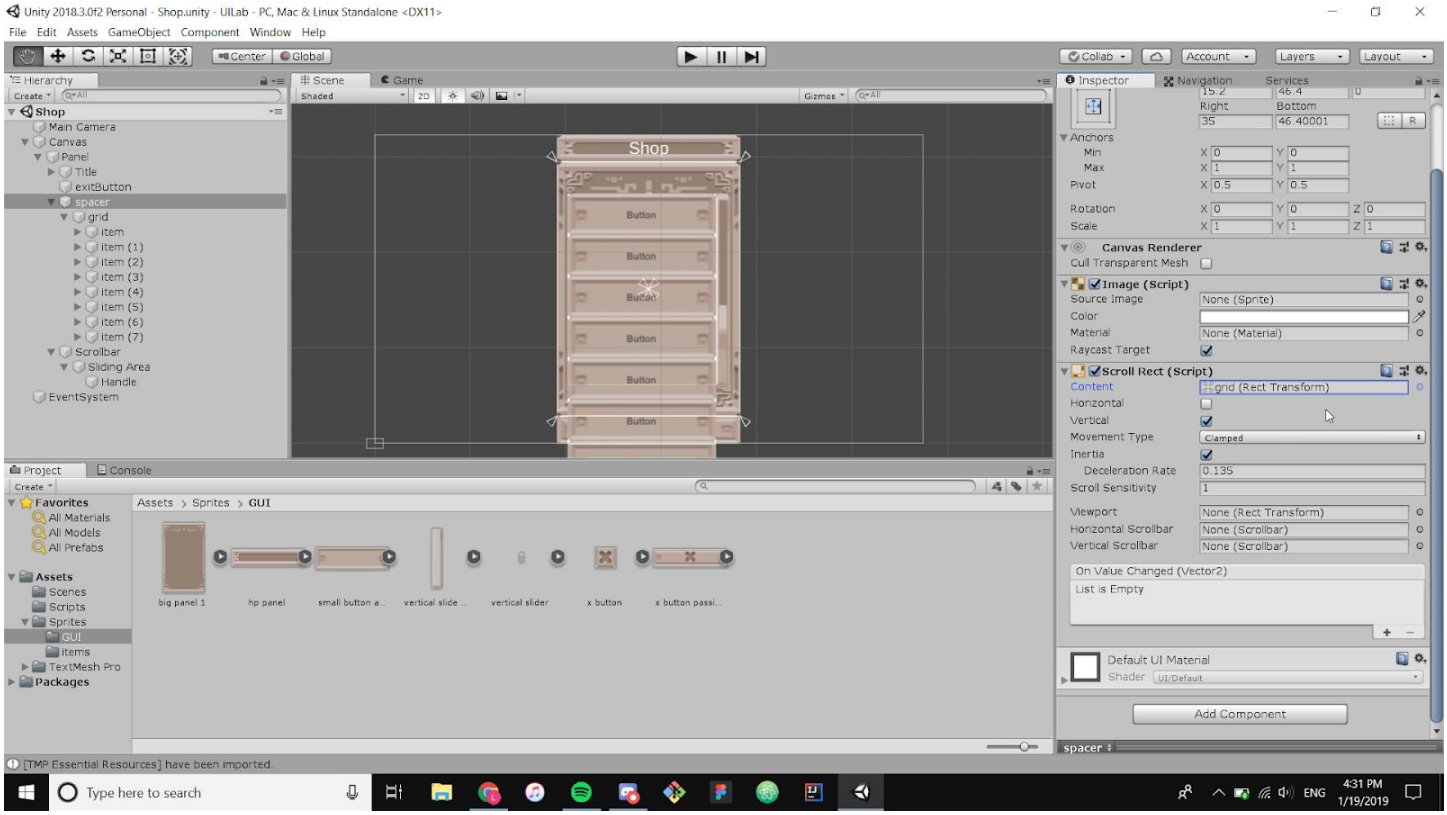
Now, let's attach the scroller to the item list. Select the Spacer in the hierarchy and add a component called Scroll Rect. I prefer my scroll bars to be clamped rather than elastic (e.i. once it reaches the end, it stops moving rather than dipping slightly and snapping back), so I changed the movement type to clamped. Pick whichever one you prefer.



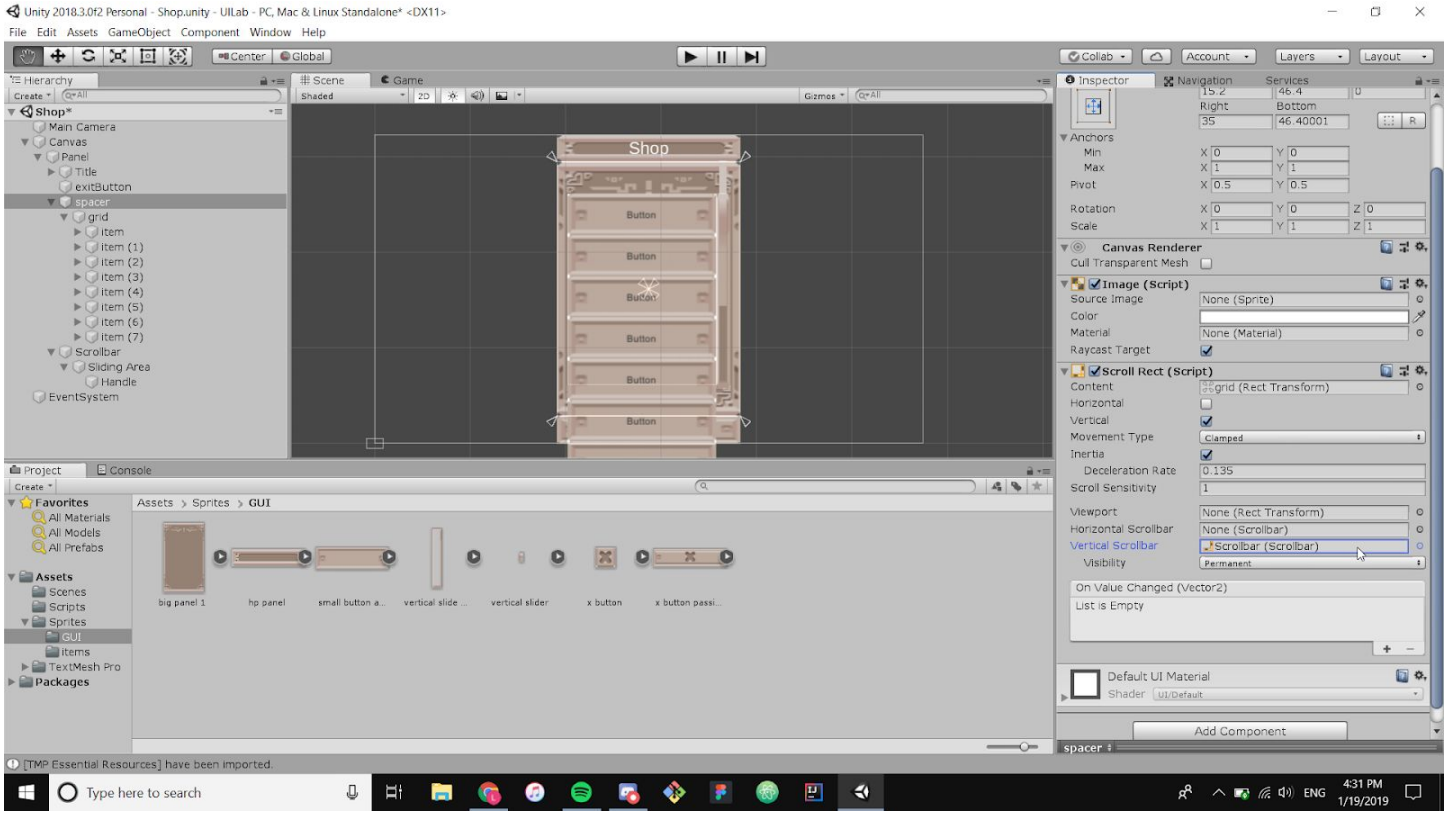
Then, unchecked horizontal so that the items list doesn't ever shift left and right.



Now that you've set up the scroll bar, let's attach it. Within the Scroll Rect (Script) component, drag the grid object into the Content box, since the grid is what we want to shift as we scroll.

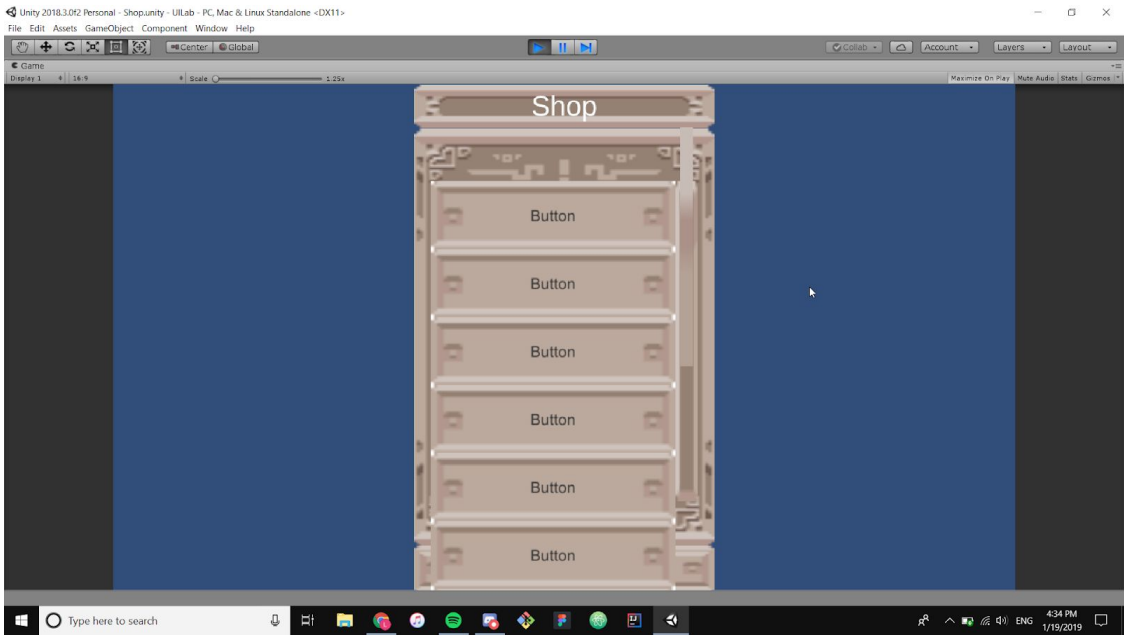


Then under Vertical Scrollbar, select your scrollbar since that's the scroll bar that we want to control the grid

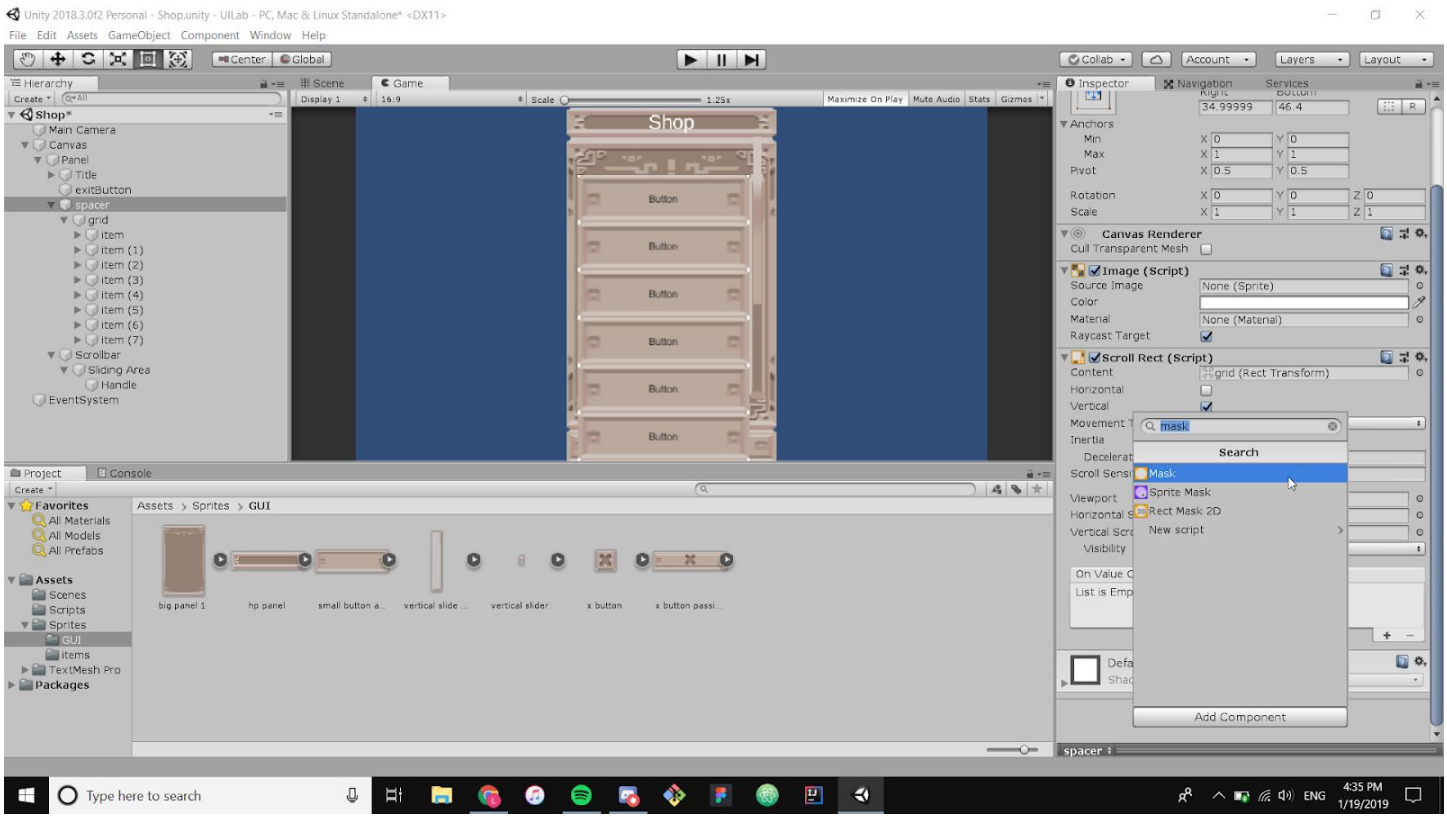


Okay, now the scrollbar should work. If you click the play button, you should be able to scroll through the items! However, if you try using a mouse scroll wheel, you'll notice it's a bit more stiff. To fix this, you can look under the Scroll Rect component in the Spacer and increase the Scroll sensitivity. I found 10 was a good number for me, play around until you're happy with the results.

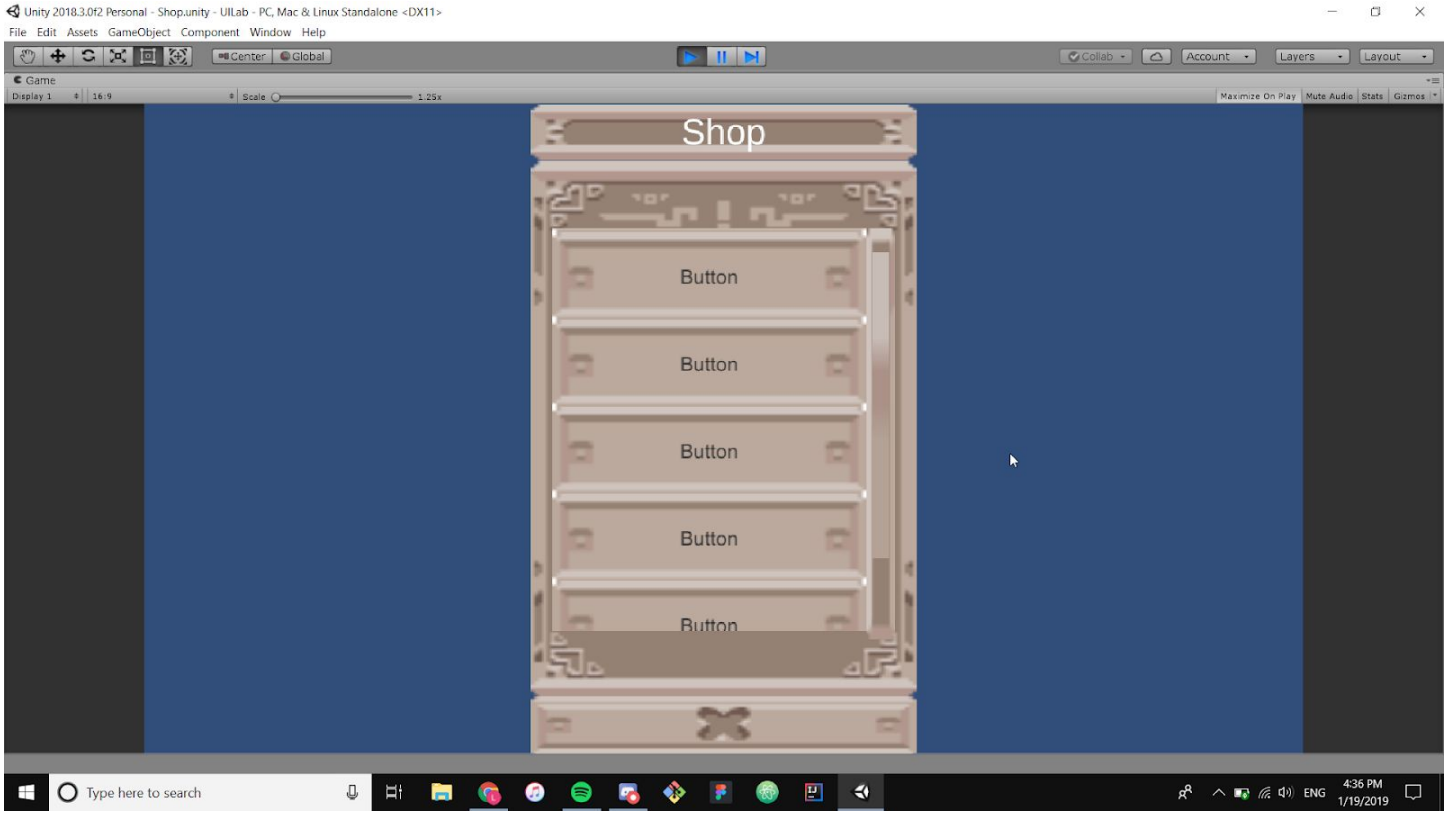
Great! Now we have a good scroller; however we're having the problem that as we scroll through the list, it covers the rest of the UI, like this:



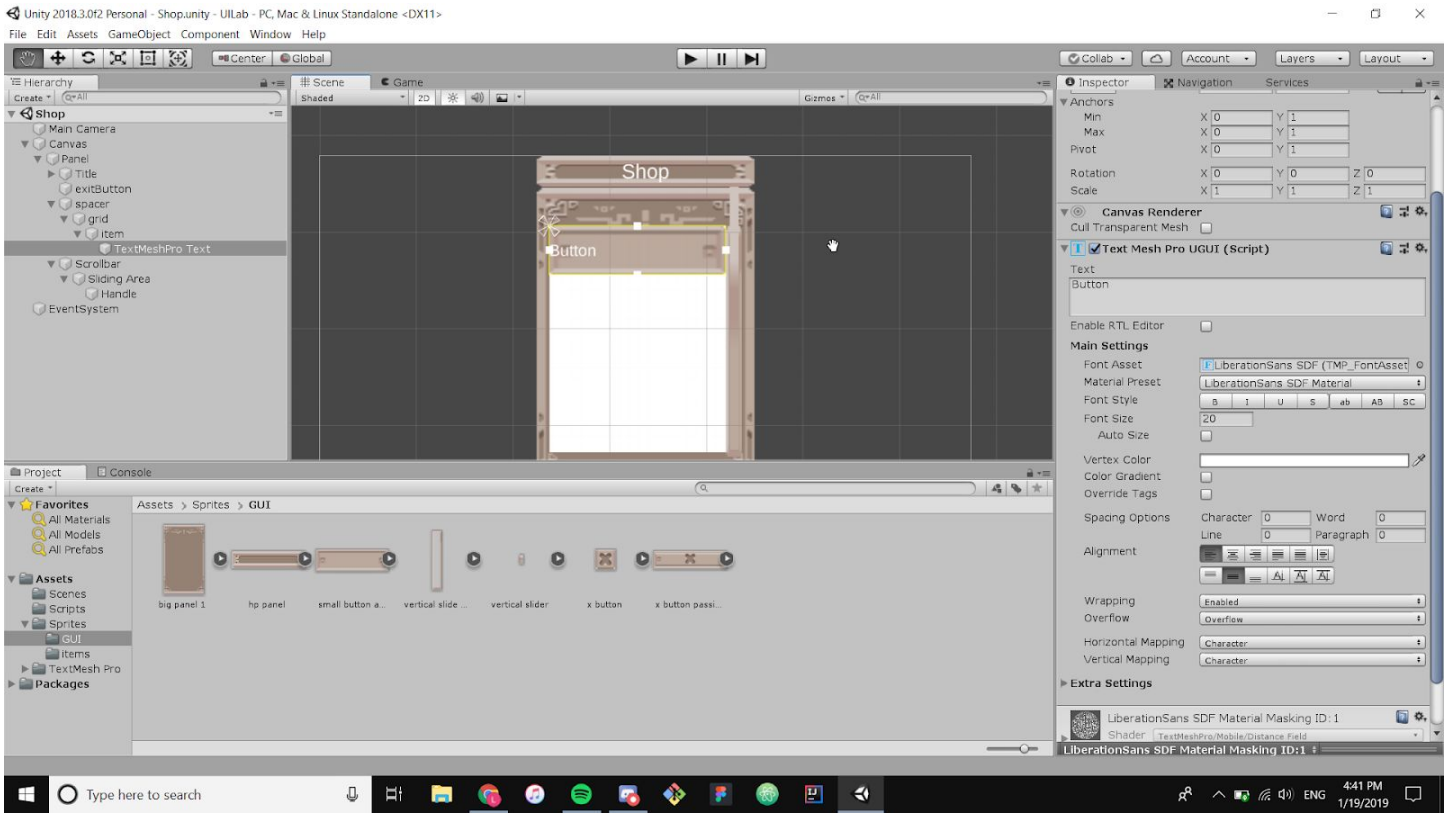
But, this has an easy fix. To fix this, we're going to use a mask which will hide the part of the list that is covering the background panel. Go to the Spacer in your hierarchy and add a component called Mask to it. This will cause everything that is a child of itself to only be viewed in the size of itself, masking everything outside of its area.



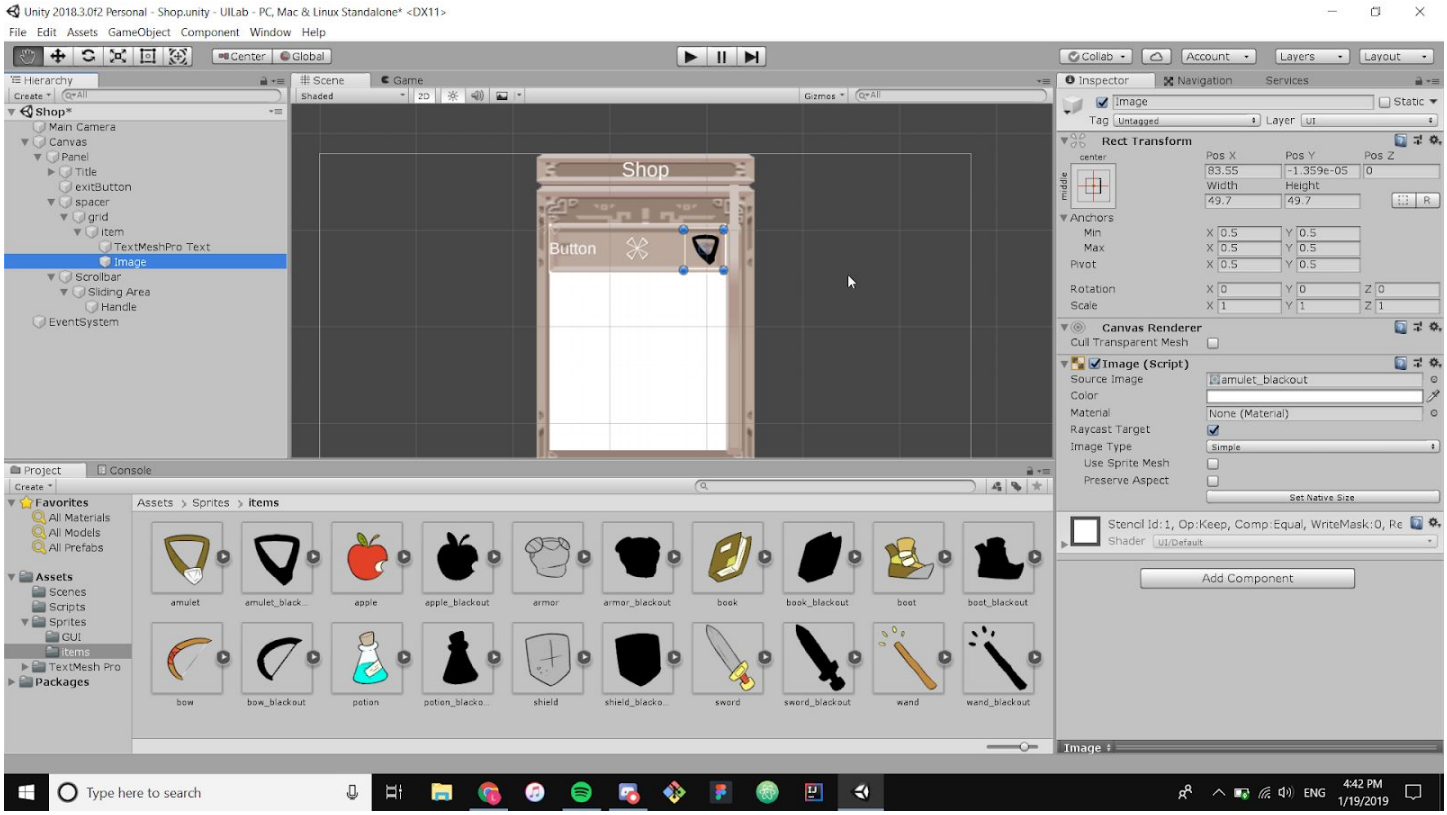
You don’t need to touch anything once adding the component, and now when you play the game, the item list shouldn’t cover any other part of the UI and still scroll. Wonderful.



Now that we have the scrollbar implemented, let’s work on the items. Delete every item button except for the first one. Going to the Item button, replace its default Text child with a TextMesh Pro object. Align the text so that the words appear on the left side of the button.

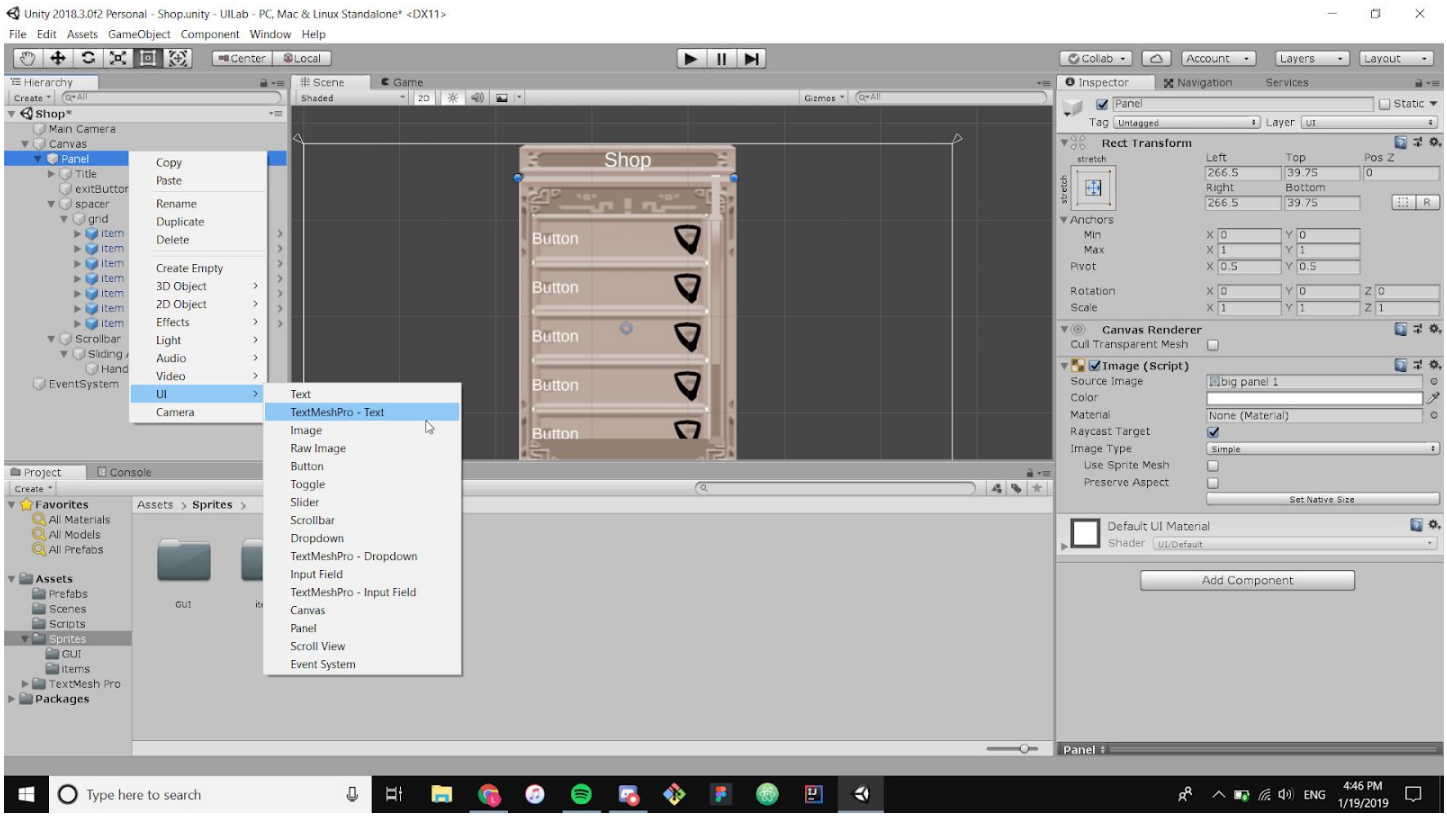


Then, add an image UI element to the button and attach a blacked out sprite (from the Items folder in the sprites folder, such as amulet_blackout) to it (this is because the sprite would no longer be blacked out only once it's bought) and drag it to the right side of the button

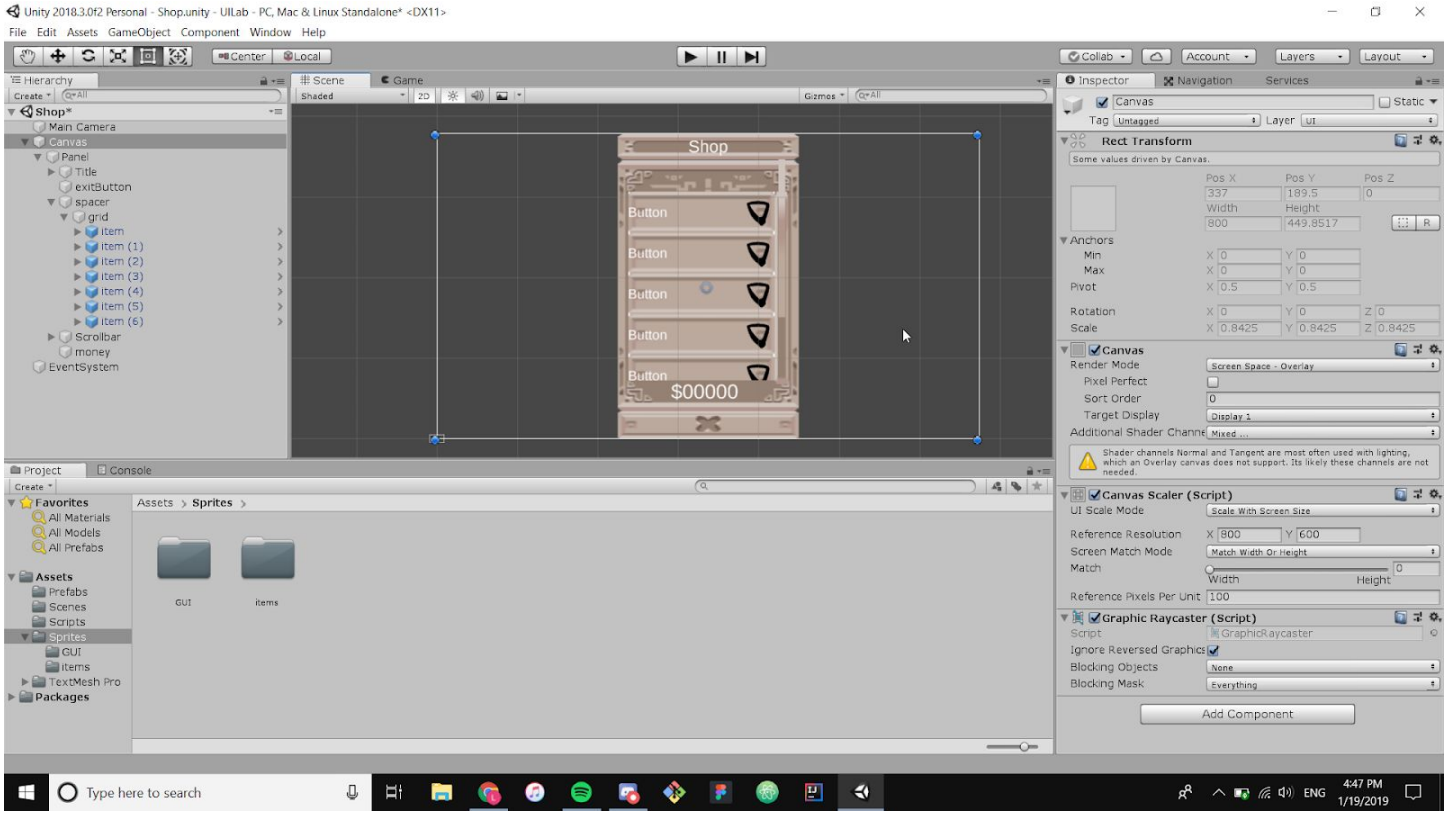


Now, let's make a prefab of the item slot. Create a Prefabs folder inside the Assets folder and drag the Item button into it (making sure not to remove the item from the scene). Once done, you should see a blue cube. Now every time we have to make a new Item slot, we can just drag out an instance of the prefab to the desired spot.

Finally, let's make a text to show how much money the player has left. Create a new text element under the Panel and name it money.



Drag it to the bottom free space and resize to see fit. Mess around with colors, sizes, positions, alignments, everything you would like. I ended up with something like this:



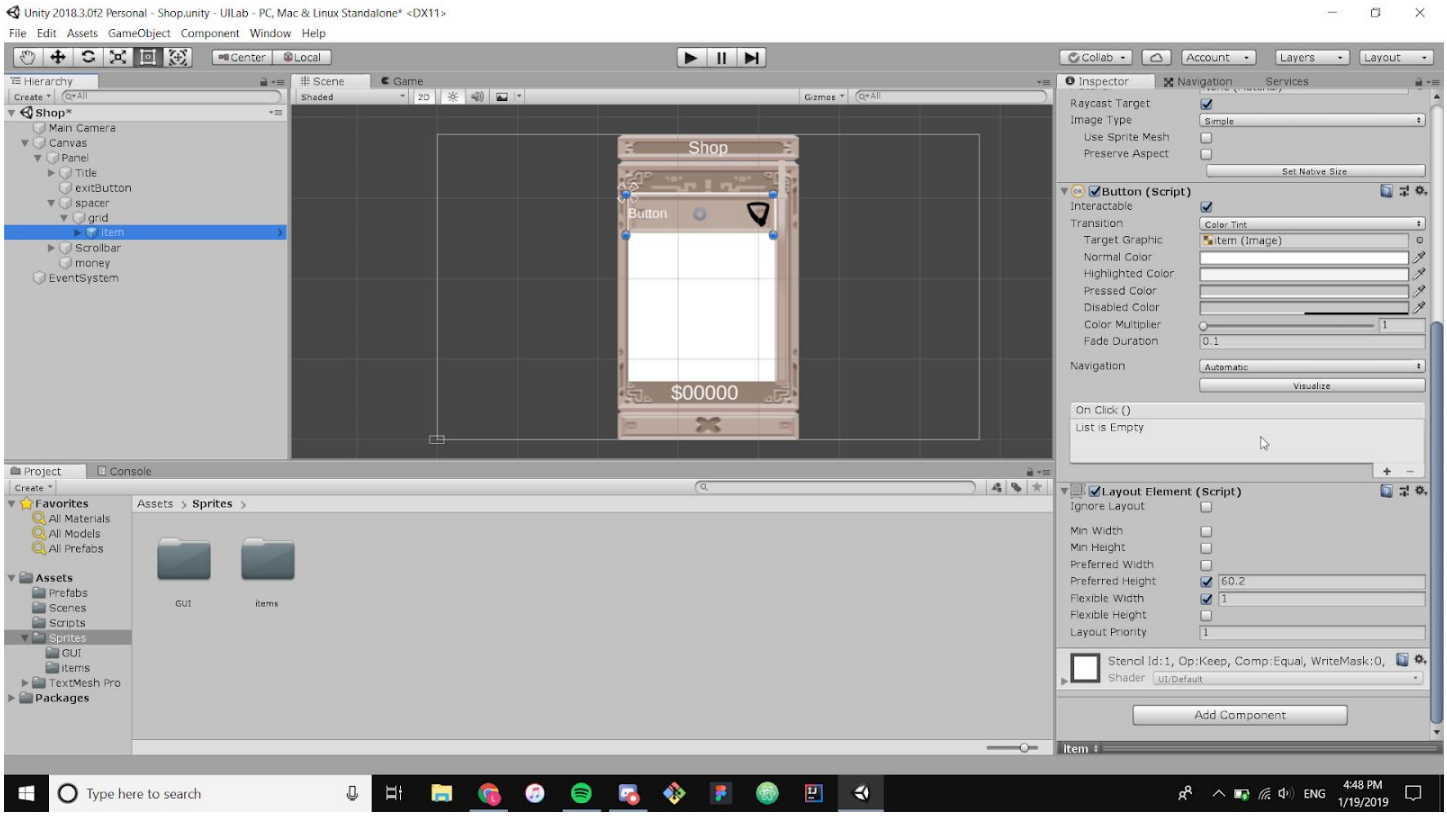
And with this, we are done with the shop! Good job! You have completed the first part of the lab. The rest of the lab will be split between artists and programmers.

Part 2 (Programmers):

There are two sections to this part. The beginning section will teach you how to update things based on the button clicks. The second part will use the scripts already made to automatically fill the list with items and do fancy things, such as check if there is enough money, and not allowing the player to be able to buy the same object, and updating the sprite to be colored and such. The second part is optional, but it is highly recommended that you at least look at the section that explains the scripts and what they do.

Section One:

If you look into Item's inspector, you will see a component that's called Button(Script):



The important thing is the On Click() list in this component. This is what will dictate what happens when the button is pressed.

To add something to the list, click on the plus button on the bottom.
To get rid of something, you need to select the item you want to remove (even if there's only one) and click the minus button.
The bottom field is an object field. Putting a game object in there will allow the events system to find all the functions on that game object that you can use on the button.
The right dropdown will allow you to choose which function you want to select.
The left dropdown is when the button can be triggered.
Off will cause the button to do nothing when pressed. The editor and runtime will allow the button to be triggered while the game is running and while you are editing values in the inspector. Runtime only will allow the button to be triggered while the game is running and not be triggered if you edit values in the inspector.

Try creating a new script in the scripts folder and adding it to a new game object called GameManager in the hierarchy. Then open the script to edit, delete all existing methods and create a new function.

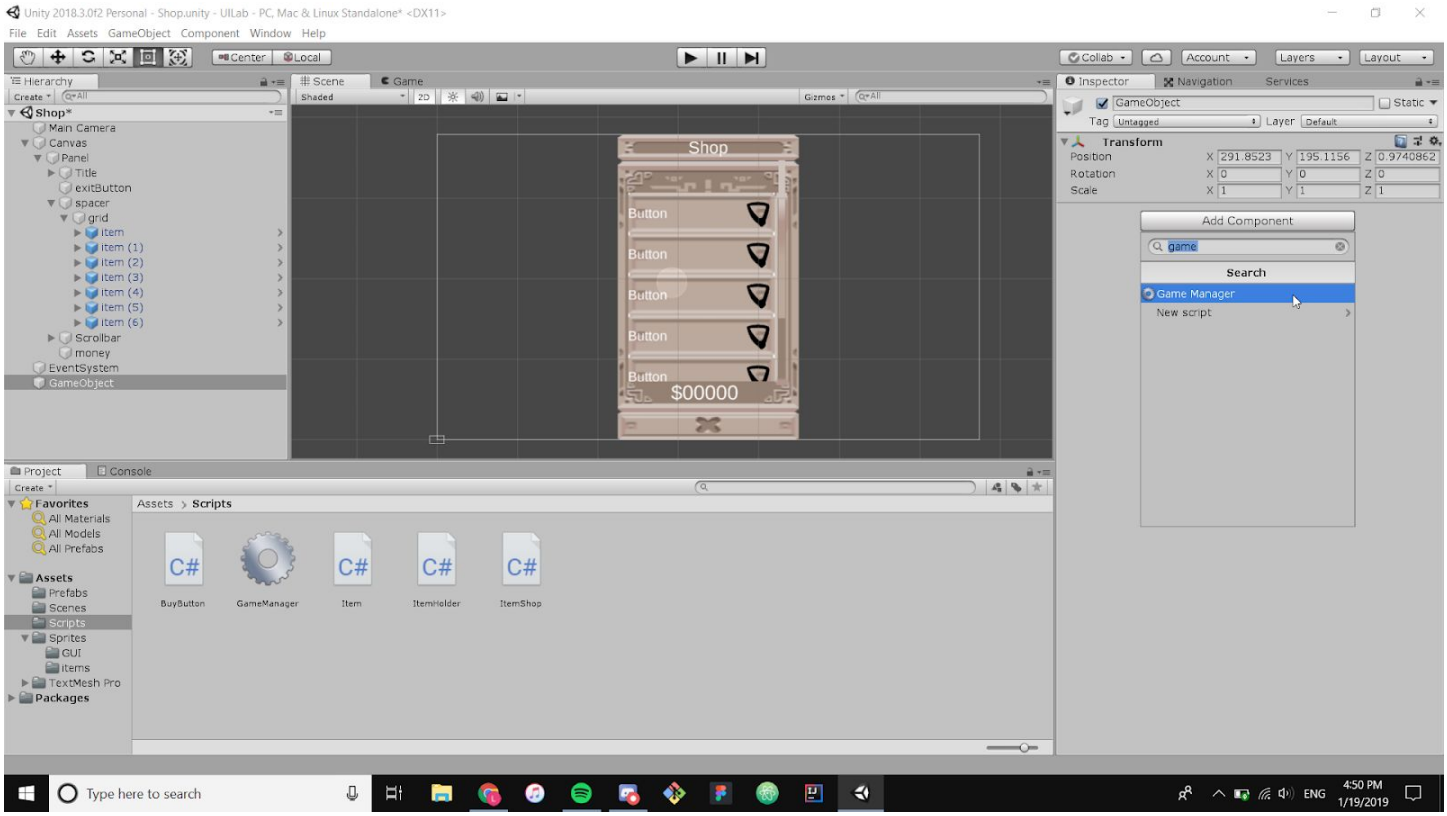
Now you can code whatever you want the button to do once it's clicked. Feel free to go as crazy or simple as you want (it might just be printing a debug.log to the console or trying to implement some of the things the later scripts will do like setting the price text to equal sold out once bought).

Once you've coded what you want to happen, drag that game object into the object field of the On click () parameter of the button and select your function name through the dropdown (you will have to hover over your script and then select the function you created). Now if you play the game and click on the button, it should do as you scripted.

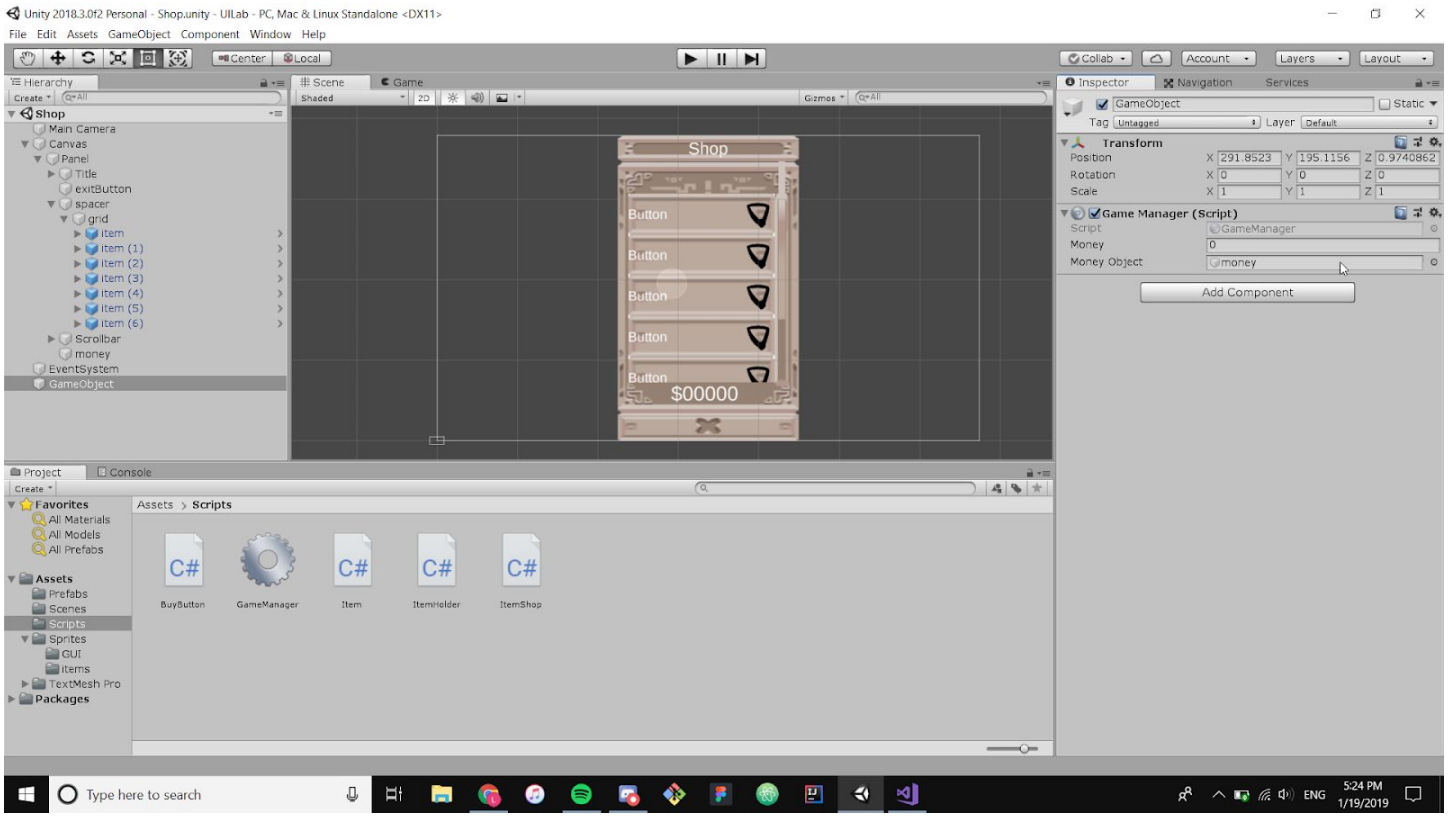
Section Two:

You should see some scripts already within the scripts folder. These are to set up automatic filling of the items for the shop. Since this lab is for UI only, most of the scripting for doing this is out of the scope of the lab, so all the scripting is done for you already. You will just be setting them up so that it works, which we will do now. At the end, all scripts will be briefly explained.

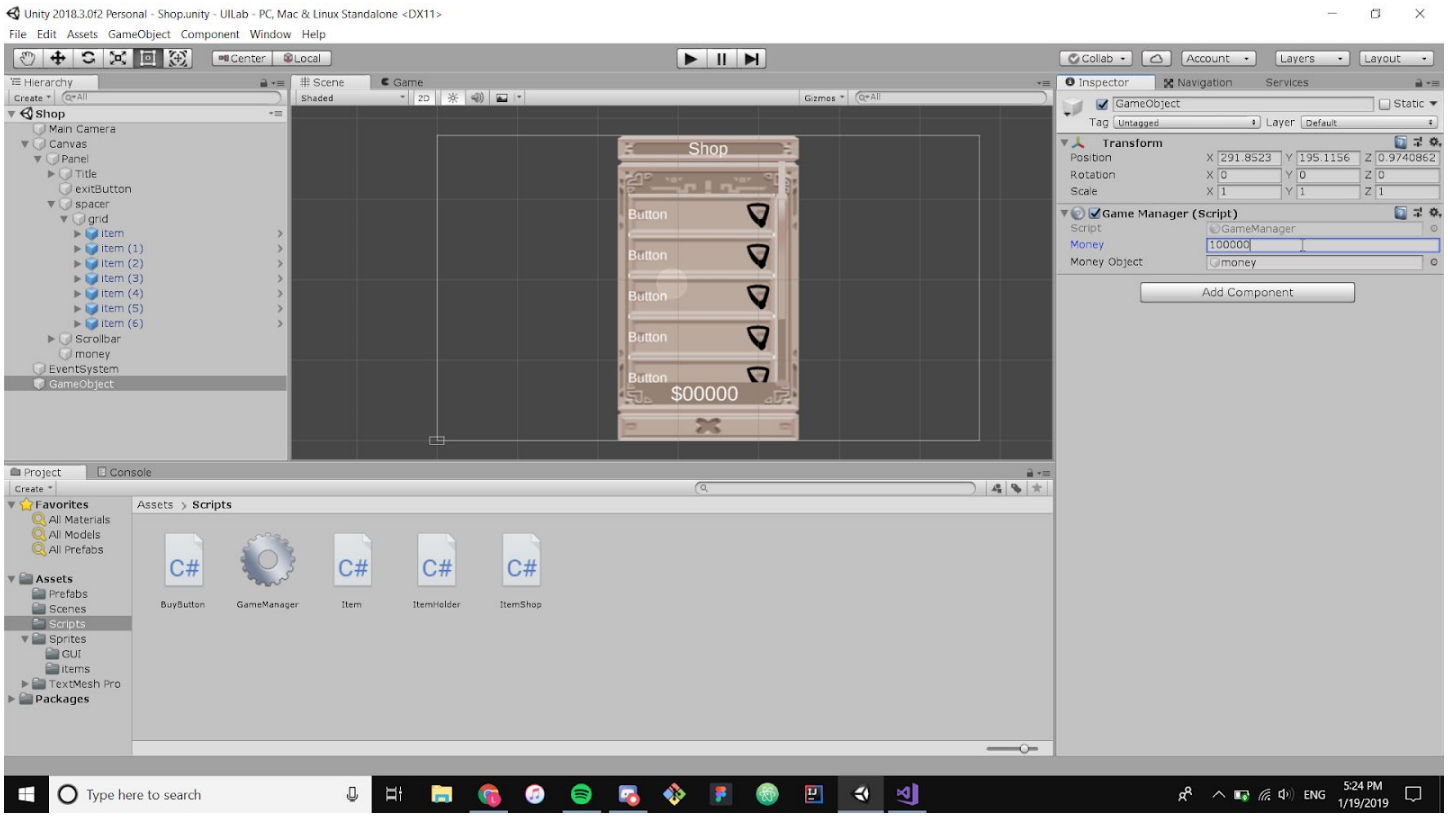
Create a new empty game object outside of the Canvas, this will handle the currency system in the shop. Add the GameManager script in the Scripts folder to that object.



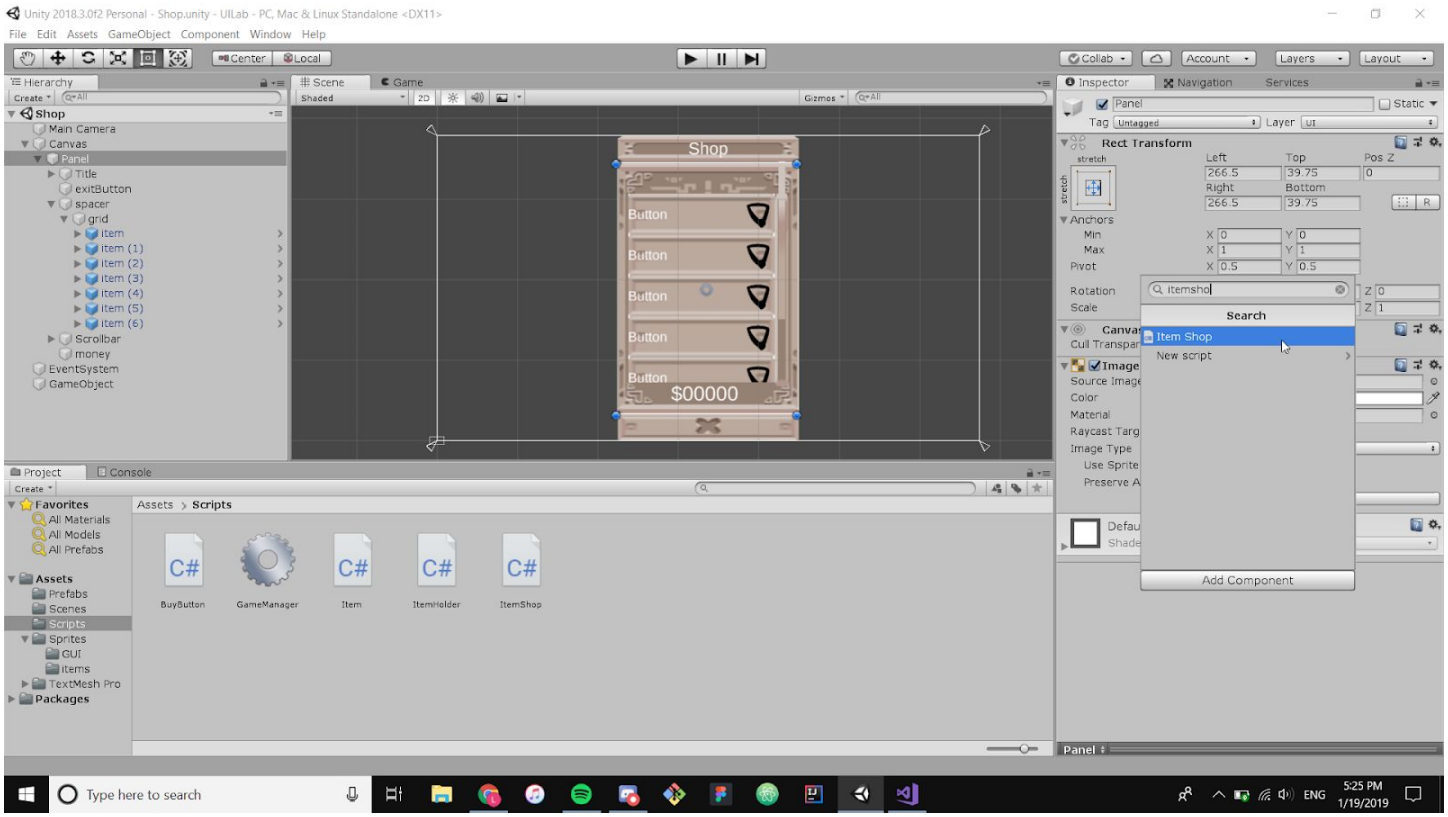
Look under the Canvas and select the Money text object we created and drag it to the Money Object component of the script we just added to the new Game object



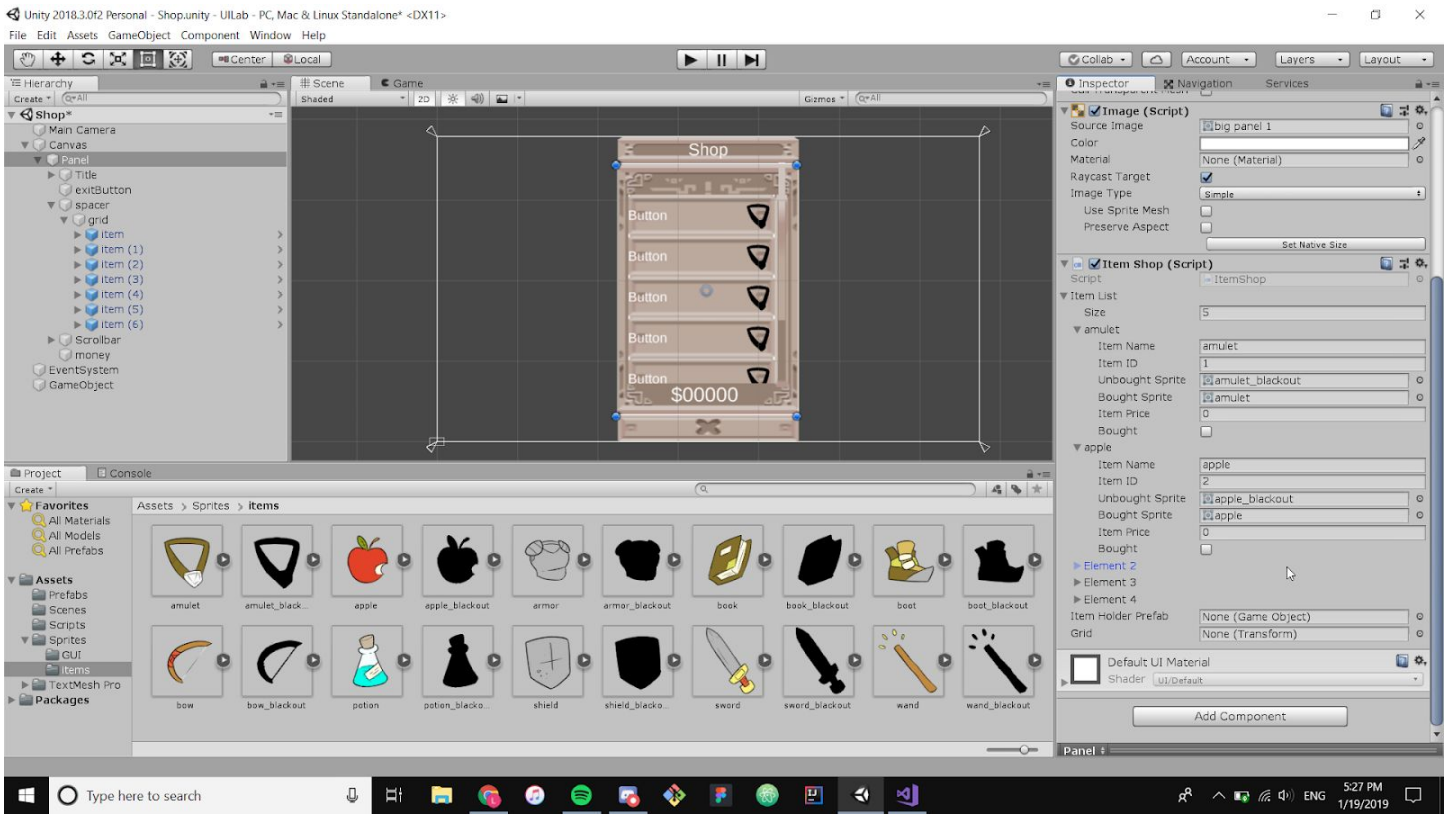
And set a number in the money section. I'm setting it as 100000.



Now go to the Panel component under the Canvas and add the script ItemShop to it

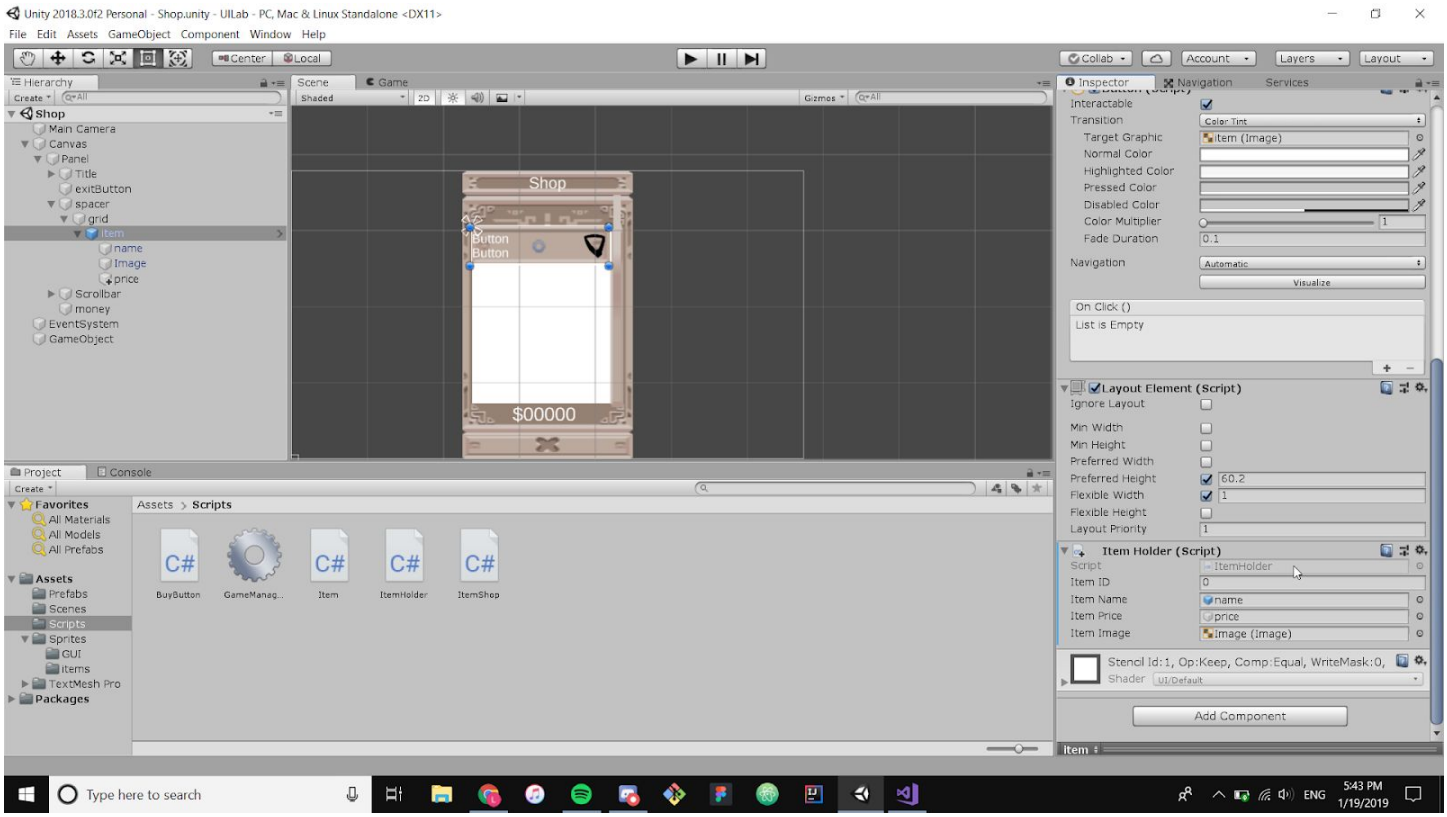


In that script component, there should be an expandable field called Item List. Once expanded, change the Size to 5. Once you do, there should be a list of Elements. For each one, set the item ID's from 1 to 5 (don't index from 0), add the necessary sprites from the Items folder in the Sprites folder, and set the item names and prices.

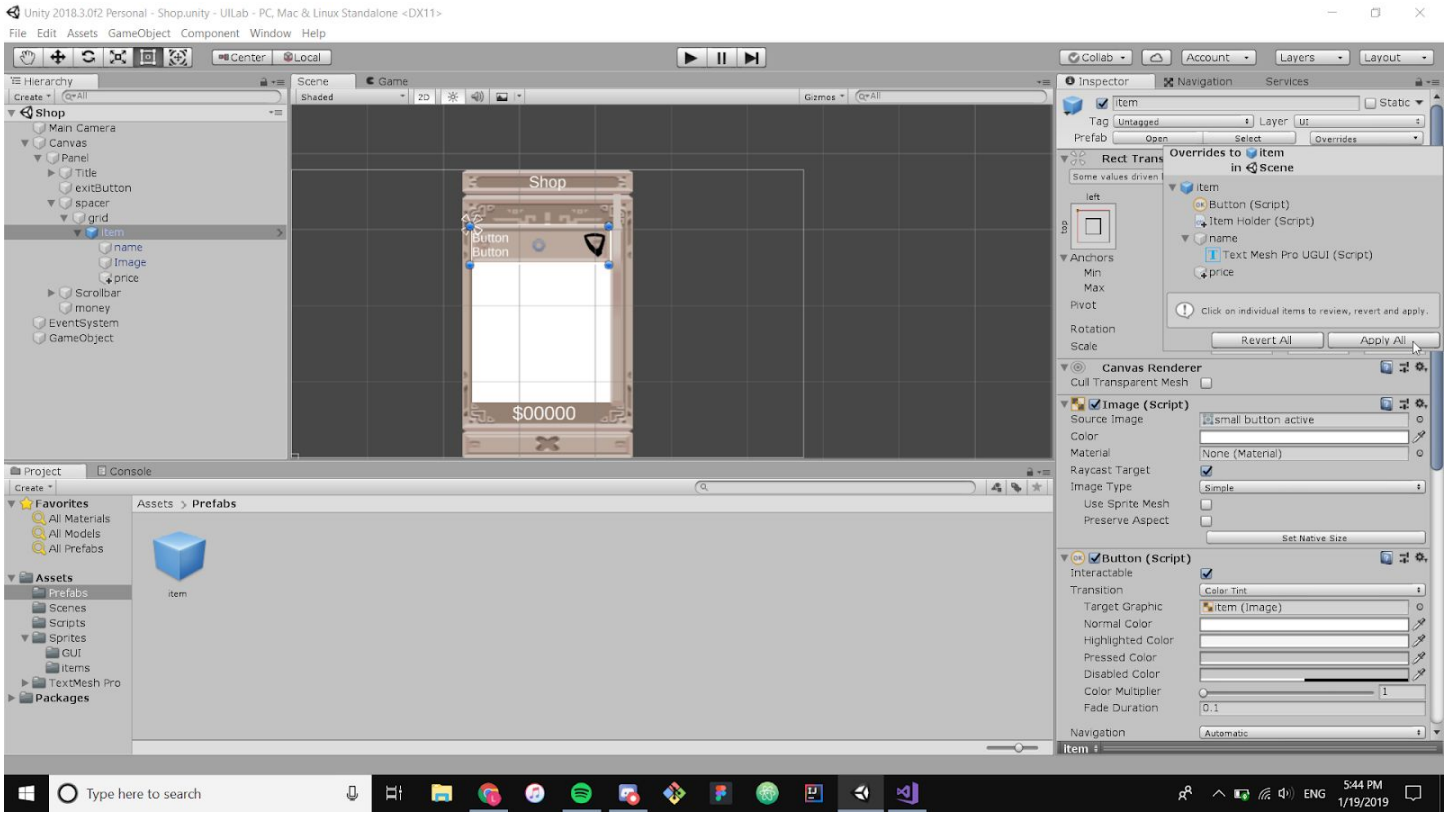


Now make sure you only have one item under grid. And in that one item, add the script, ItemHolder. And in the components, add the necessary UI elements for each item (name goes into Item Name,

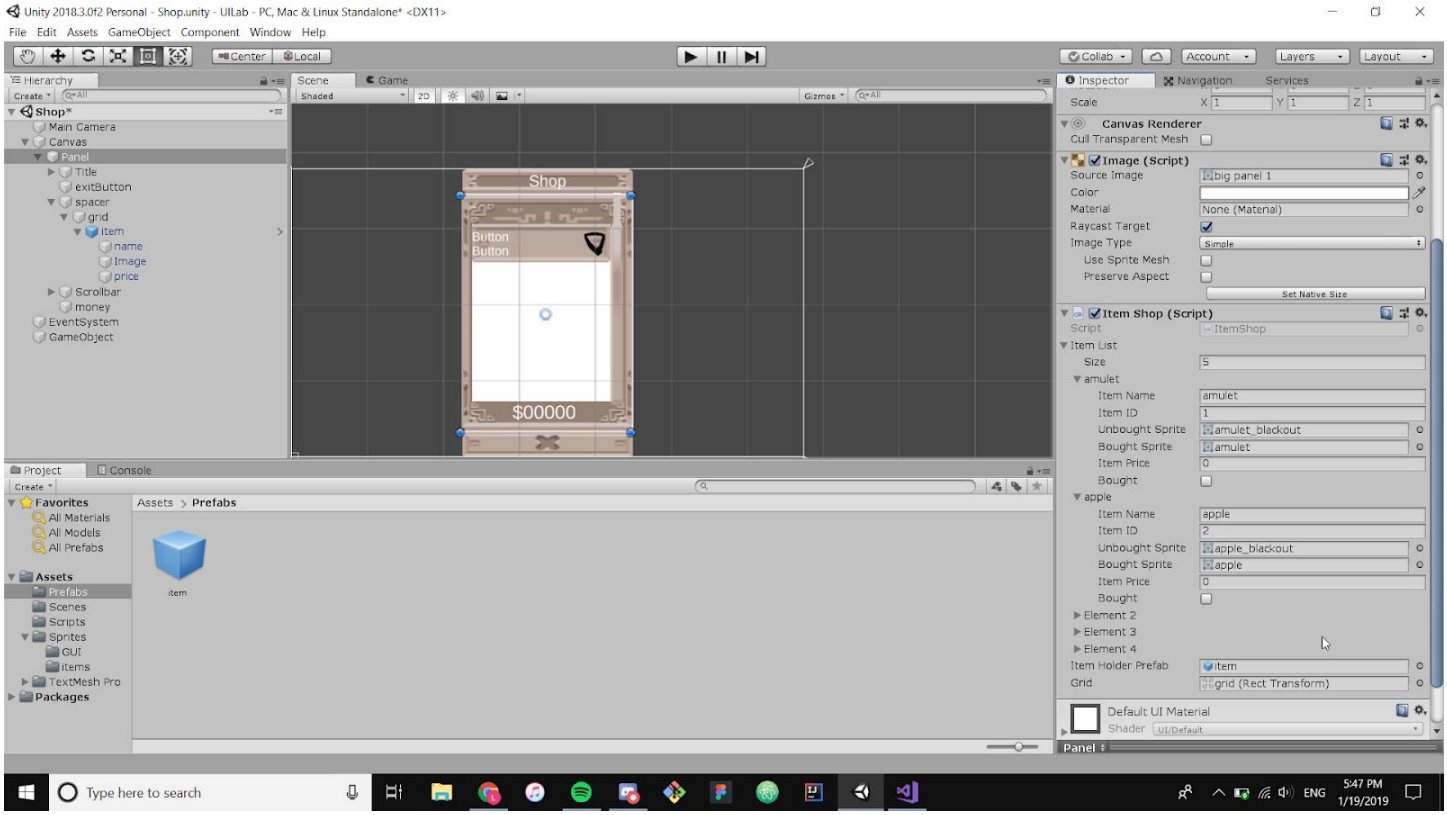
price goes into Item Price, and its sprite goes into Item Image). You will need to add another TextMesh Pro object for the price and align it accordingly.



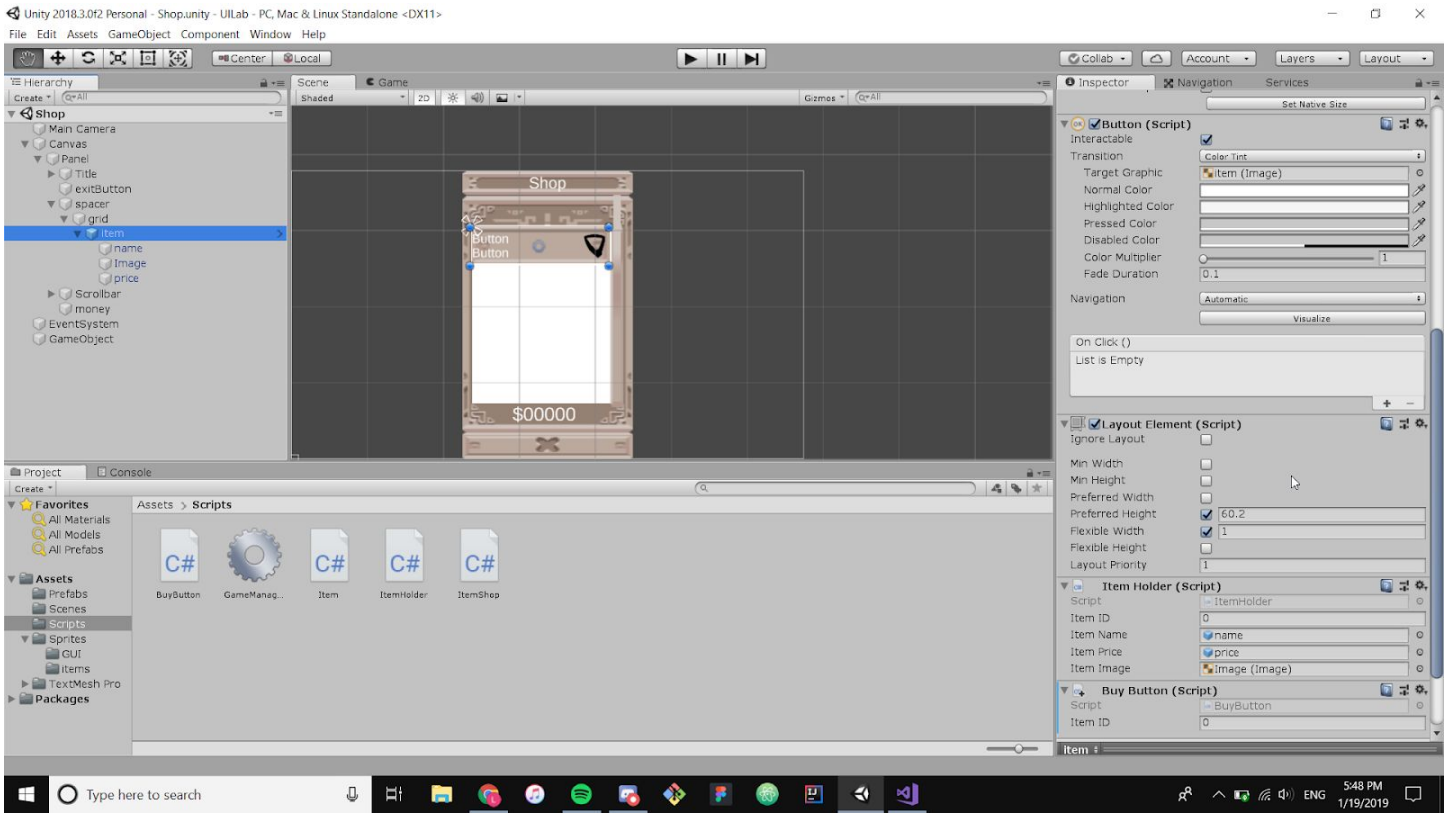
Now, update the prefab by clicking Apply All under the Overrides dropdown like so:



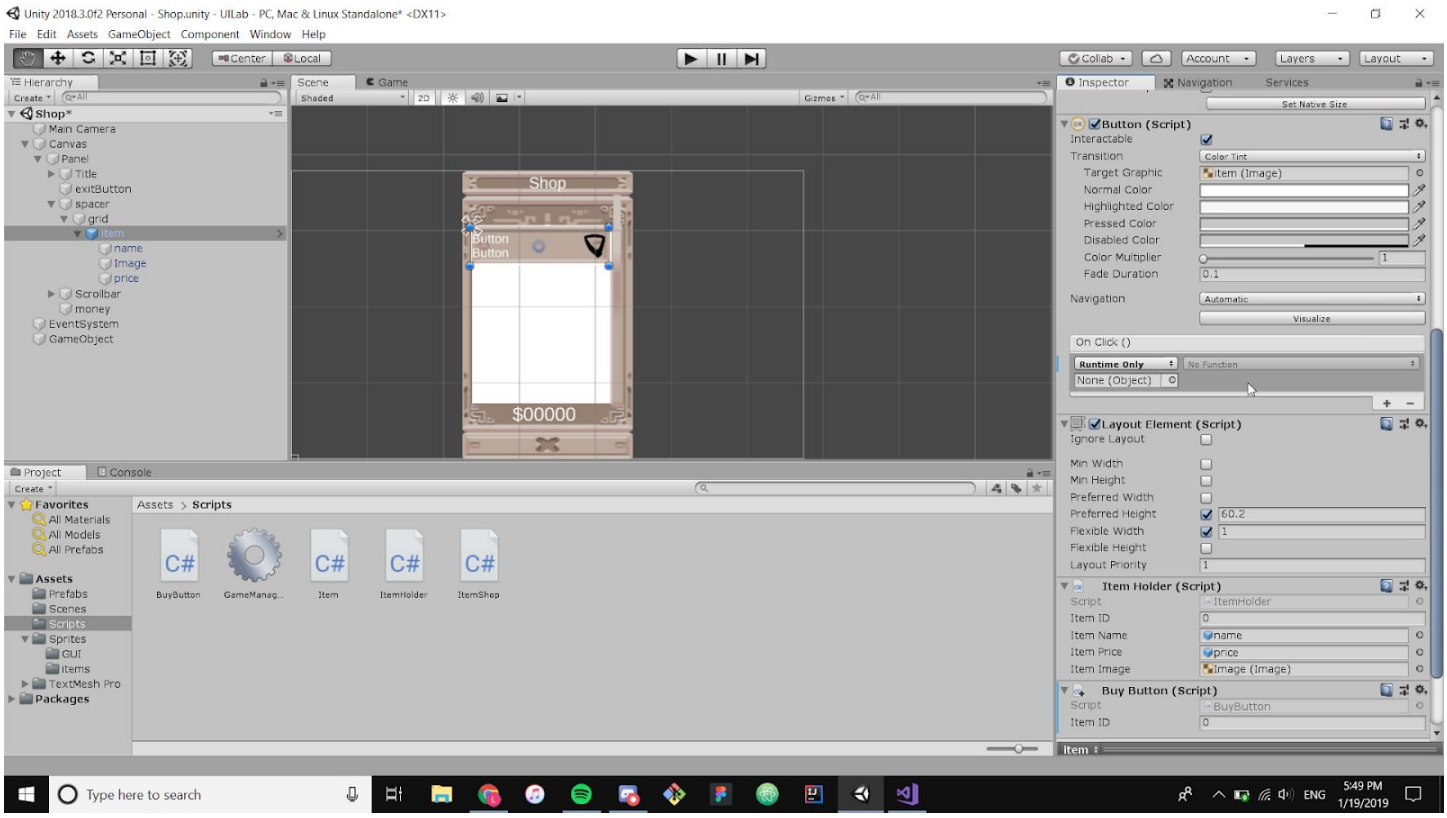
Then select the panel in the hierarchy and add the Item prefab to the Item Holder Prefab section of the Item Shop Script, and in the Grid section, add the Grid game object.



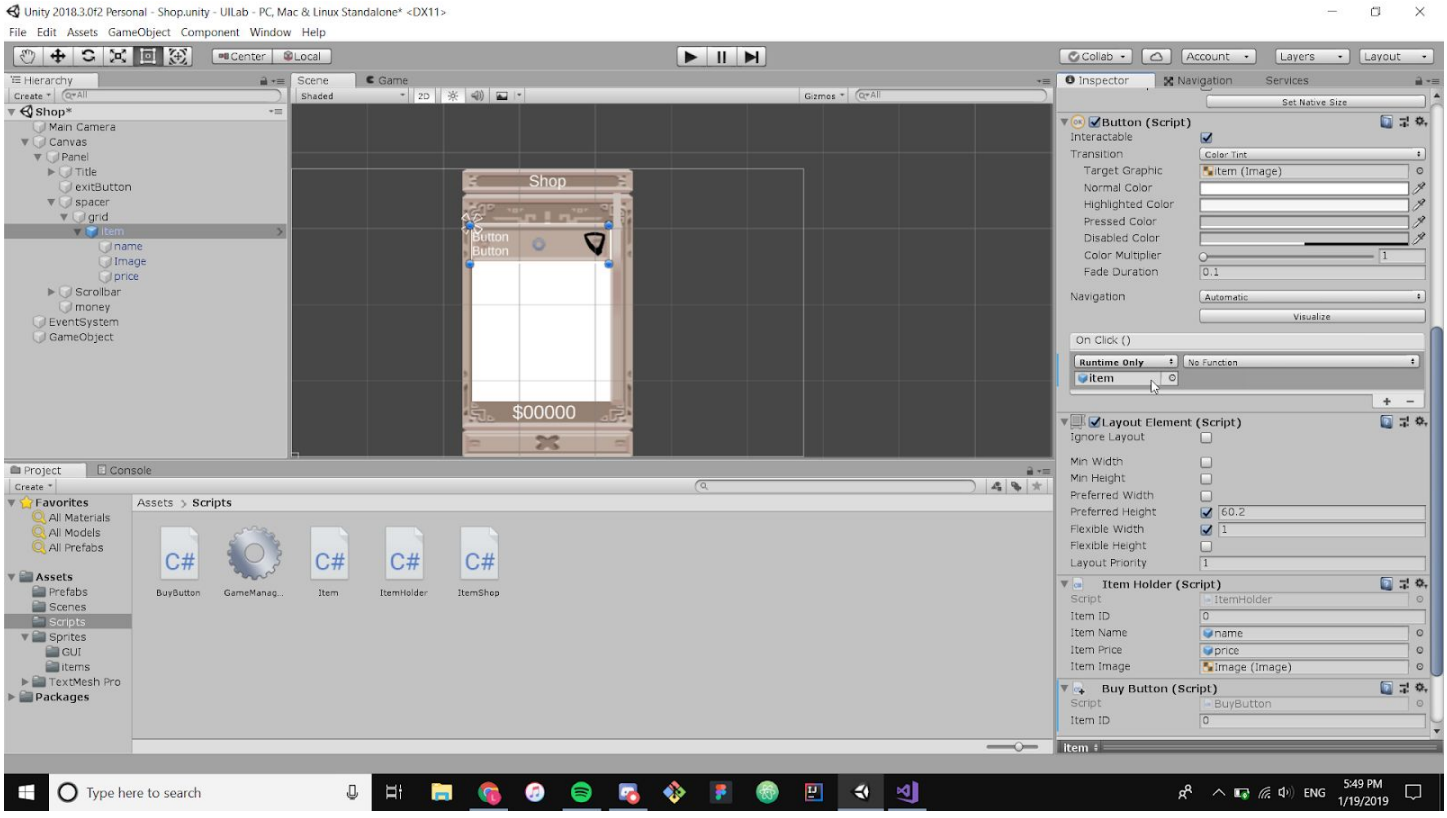
Now let's make the buttons do actions. In the item in the hierarchy, add the script called BuyButton



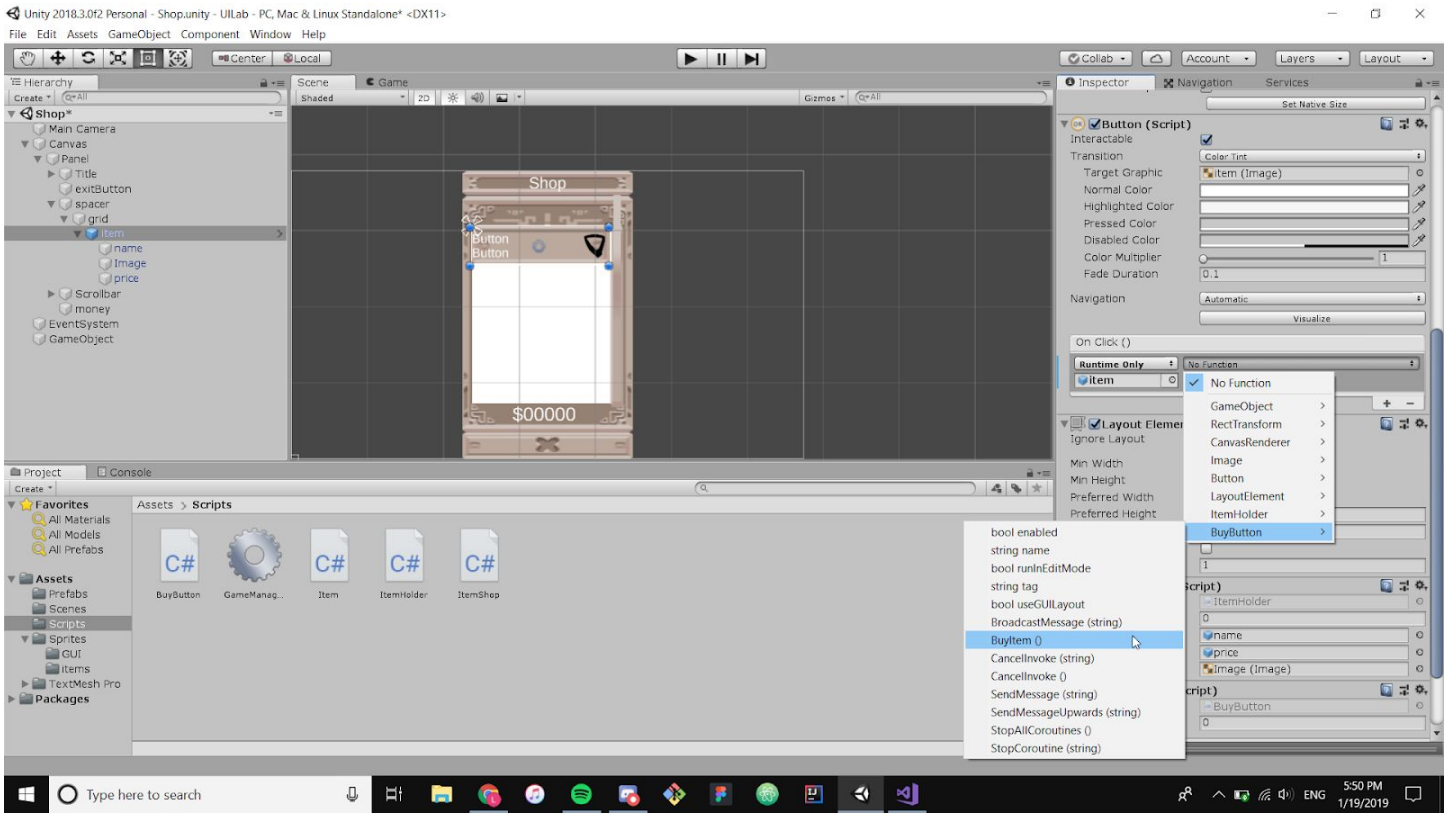
In the Button(Script) component, add a new condition to the On Click() list



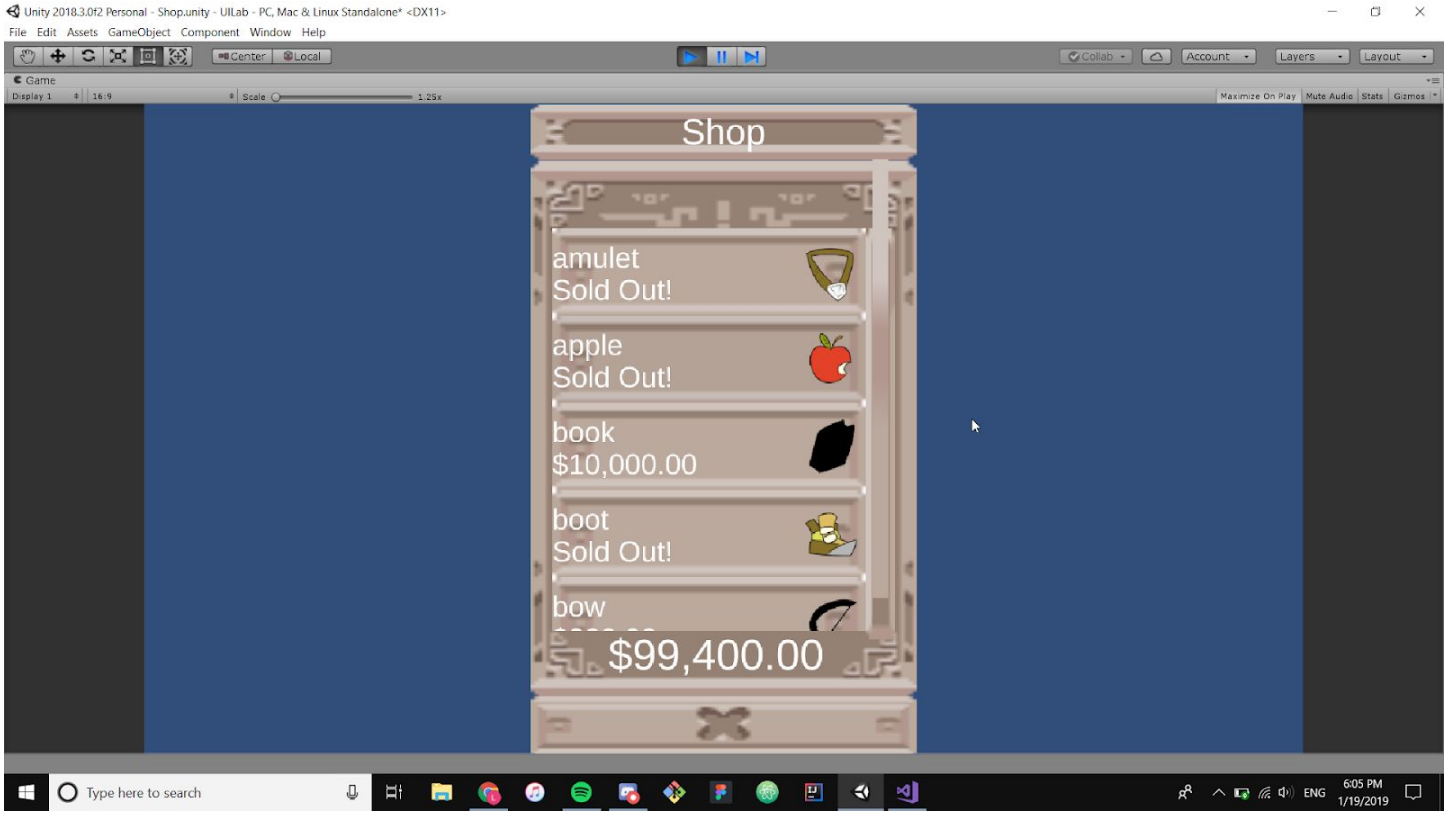
Drag the Item itself into the object portion



Then in the function part, select BuyButton.BuyItem()



Then **MAKE SURE TO APPLY THE PREFAB CHANGES** again. Deselect the item by unchecking the checkbox next to its name at the top. If you play the game, you should be able to click on each item and once you do, it should update the sprite to a colored image and update the amount of money you have, and you shouldn't be able to buy an item you already bought or buy anything that is greater than your current total money.



If you ran into any problems, make sure that your Item prefab changes have always been applied, and make sure that the Item Shop script in the Panel object is holding the updated prefab in Item Holder Prefab.

Now I will explain what each script does. It's highly recommended you look at each script as it's being explained:

GameManager.cs:
This is the Script that handles the overarching monetary system. It keeps track of how much money the player has left and updates it accordingly.

Item.cs:
A class used to hold all the important information that each item has, such as the sprite and name and ID.

ItemHolder.cs:
A class to hold the information of each item in the shop. This is different from Item.cs since it holds only the information visible from the button in the shop whereas Item.cs holds all the information the item has.

ItemShop.cs:
Sets up the shop.

BuyButton.cs:
Checks to see if an item clicked can been bought, and updates accordingly. This is the script used on the button.

Artists Second Section:

Arguably, the most important thing behind UI design, is that less is more - The less things there are for the player to look at, the more the players will play attention to what is there. For example, if there are no colors except for a big red button, most of the attention will go to the button. If there are too many things, the player won't know what to read and will have trouble reading what there is. Cleaner designs for things that don't affect game environments tend to be more lauded.

An example of really cluttered UI:

As you can see, it's hard to read and the player doesn't know where to look or what's happening, it even detracts from the gameplay because it's so loud. So, let's look at a few games and see what they did right.

Half-Life 2

As you can see the HUD (head-up display) is very clean and unobtrusive. It gives the player exactly all the information that they need without being too bright and in the face of the player. Not only that, but Half-life mainly used very clear audio cues to alert the player of things traditional games would use UI for, such as when they picked up ammo or resupplied their health. This game shows how the UI can be very clean and beautiful, and that not every single thing needs to be a visual cue.

Hearthstone

For hearthstone, the UI seems like an integral part of the game. All the important information is easily accessible; the game highlights and brings up important information to the player when they need it, in a very easily digestible form. Things like who can attack, which card you're hovering over, and who adds to spellpower is always clear. This game shows how you can integrate the UI into a part of the game instead of make it an additional layer of clutter so that the player can focus on the gameplay.

Overwatch

In overwatch, all the important information is consolidated. Oliver Janoschek, the senior UI artist at Massive Entertainment, spoke about the Zarya HUD.

“I’m one of the advocates of UI design who is all about bringing the information into a space a player focuses most on, and removing eye travel to fetch information,” Janoschek says.
“Zarya's energy indicator sits comfortably around the actual crosshair, which in return makes the information super easy to access and evaluate. The element itself is subtle enough to not interfere with any of the gameplay, yet its permanent presence makes it so much easier to react to various gameplay situations, which is crucial in such a fast paced FPS game.”

So this game shows how you should put as much important information in a very easy to see play as you can, without making it look cluttered or illegible.

Assassin’s Creed

If you look for the UI or HUD for Assassin’s Creed, you see nothing. The fact that there is nothing makes the game so much more immersive for the player. The purity of the screen helps the player to connect more to the character and make the game feel less like “clusters of code.” However, this is only possible since there are many cues in the game, such as important game locations for the player to orient themselves, that allows for the lack of HUD. So, this game shows us how less can be much more. It also shows how clean game play without a HUD can be the mark of great game design.

Dead Space

Jim Brown, a senior designer at Epic Games says "*Dead Space* is very often the 'go-to' for creative HUDs," Brown says. "Your health meter displays on your back, and the overhead map is a projection that draws on the floor and guides you to your next objective. While this was all very groundbreaking and turns the traditional notion of a HUD completely on its head, this design is often overlooked for the even greater impact it had on the gameplay itself. *Dead Space* is a mixture of the shooter and survival horror genres, and the in-game HUD played very well to the horror aspects of the game: it helped keep the player focused in the moment, rather than pulling them out to a 2D overlay that distracted from the gameplay."
All the necessary information is built into the game rather than making it something that is always a layer on the screen for the player to see. This allows for better immersion, like what Assassin's Creed was able to create, for games that require UI elements. This game shows us how you can integrate the HUD completely into the game.

Art Credits:
All art was taken from free asset stores. These are the specific ones I used for this tutorial:
https://plastic-vibes.itch.io/pocket-gui-for-pixel-mobile-games
https://joszs.itch.io/medieval-item-pack

CHECKOFF: Show an instructor your shop. As long as you can scroll through the items, it's good enough; none of the scripts have to be implemented, but if they are, feel free to show it off!

Extra Challenges:
If you've done the scenes lab (or if you want to look up how to switch between scenes through buttons):

- Try making a main menu scene (using assets given or assets of your own) and make it so that you can go from one scene to the other through buttons.

Art:

- Redesign the Shop with the assets given or create your own GUI assets! Design them and import them if you can. Try to replace all the assets with your own art
- If you can't do the top option, try redesigning the HUD (or maybe lack of HUD)/UI for an existing game or a game that you made up on a piece of paper.

Programmers:

- If you haven't done the second part of the programmer's section, do it
- Try adding even more on top of what is given. Maybe allow the player to buy an item a certain amount of times. Try letting them sell things back to the shop. Be creative
- Redesign the shop. Set up a new layout and update the shop accordingly.

Final notes:
This lab doesn't cover everything, notably things like drop bars, scroll view, and toggles. There are also many properties and options on each UI element that can be used for so much that can't be covered in an hour and a half lecture. So, if you're interested, feel free to ask us about them or, you can use online resources to find out more; a great place for UI resources is here:

https://unity3d.com/learn/tutorials/topics/user-interface-ui

It covers most of the different UI elements in more depth and explains RectTransform way better than I ever could. They also have a tutorial on making tic-tac-toe with just UI elements, which could be good practice with UI. UI is hard to get right and will take a lot of tinkering to get things to work.