Potions \ Crafting Minigame

1. Concept overview

The idea is to create a minigame for brewing potions, that could easily be adapted to crafting other items, and inserted in to a larger game.

This will involve: various GUI elements: a GUI to show what crafting materials you have accumulated in your inventory, and a separate GUI for creating items.

When enough of the correct materials are present in the players inventory, crafting certain items will be possible.

Crafting item inventory tracking will use int values attached to the local player via local scripts stored in the GUI. Sever scripts and remote events will handle cross server \ client interactions, including initializing values.

Other assets needed:

- models for Crafting materials will be collected from the game world.
- decals \ images for UI elements
- Sound effects (optional but nice to have)

2. Setting Up (server script and variables)

First we need to create a script to set up the variables that will track our players crafting inventory.

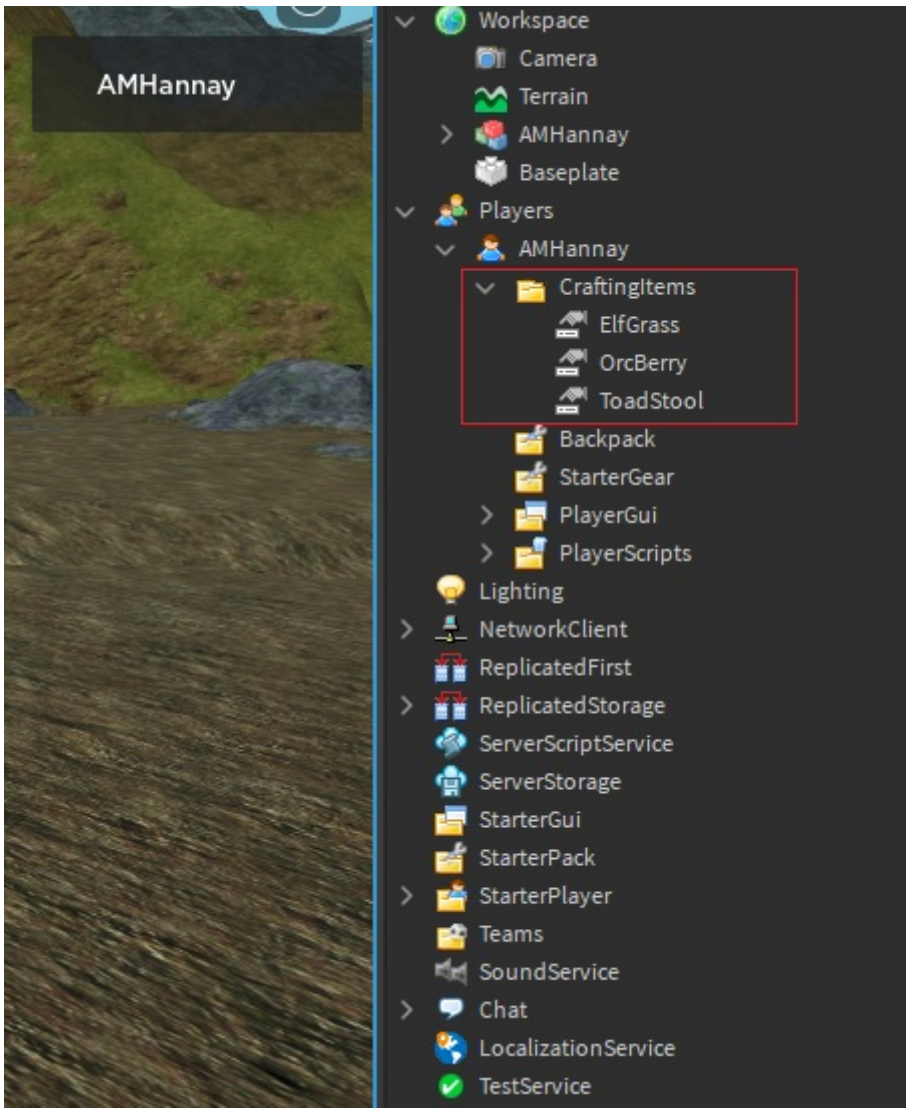
To do this, create a new script under Server Script Service. Name the script something you will remember, like "LevelServerScript"

In this script, we first need to create a function that will run every time a new player is added. This function will then add a folder, to store our inventory info in, and several variables for different inventory items. It will also set the values of all these items to 0 to start with.

LevelServerScript

```
game.Players.PlayerAdded:Connect(function(player)
    --sets up the main inventory folder
    local craftingItems = Instance.new("Folder", player)
    craftingItems.Name ="CraftingItems"

    --initalise the various ietms.
    local elfGrass = Instance.new("IntValue", craftingItems)
    elfGrass.Name = "ElfGrass"
    elfGrass.Value = 0

    local orcBerry = Instance.new("IntValue", craftingItems )
    orcBerry.Name = "OrcBerry"
    orcBerry.Value = 0

    local toadStool = Instance.new("IntValue", craftingItems)
    toadStool.Name = "ToadStool"
    toadStool.Value = 0

end)
```

Now when you run the game, it you look under Players>YourName you should see a folder called CraftingItemts with three int values under it like this:



3. Harvesting Crafting Resources

Now we have our variables set up, next we are going to need an object in our game world that we can harvest for crafting resources .

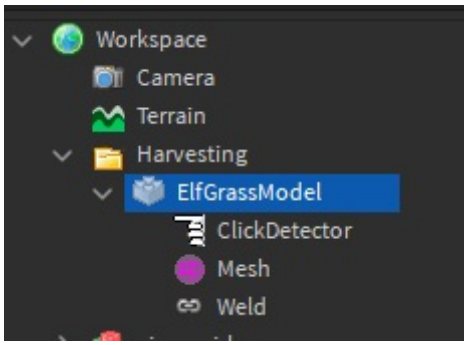
To do this we will need a model in our game world, that will represent our harvestable items. You can create your own model for this, but today we will be using one from the Roblox Toolshop. Be careful when using items other people have made in the toolbox, some can have malicious scripts in them. One way to be safe is to only use Official Roblox created resources. We have created this model here, adapted from a Roblox created model: URL. This will be our elf grass (I know, it is more of a vine than grass, but you get the idea!). To import a model from the tooblox simply click on it and it will appear wherever you have the camera focused.

To import a model from a file Right click Workspace and click "insert from file", then locate the file and insert it into Studio. Alternatively you can drag your file from explorer into the game window.

To keep our workspace organised, we will keep all our harvesting materials in a folder. Add a new folder to your Workspace and rename it "Harvesting". Now, still in the explorer window, click and drag the ElfGrass model into the new Harvesting folder.

Now we will need a way for your players to harvest the item. In this example we will keep it simple, and they will just have to click on it. To do this we need to add a click detector, by clicking the plus next to our model in the explorer.

Now your Elf Grass model should be in it's Harvesting folder and have a click detector like this:



Because we will want multiple Elf Grass models, right click and duplicate it several times. Now move them around the workspace to where you want them.

Now, to make our harvesting work, we need a script. We will need to refer to our newly created values in a local script. There are several different places you can create local scripts and because this script will interact directly with our players GUI, we will create it under our StarterGUI. Create a script under StarterGUI can rename it to HarvestingScript.

First, we nee to set up our player variables, by finding the local player, and accessing the various item int values we already created:

```
--player variables.
local player = game.Players.LocalPlayer
local craftingItems = player:WaitForChild("CraftingItems")
local elfGrass = craftingItems:WaitForChild("ElfGrass")
local orcBerry = craftingItems:WaitForChild("OrcBerry")
local toadStool = craftingItems:WaitforChild("ToadStool")
```

Next we need a variable to relate to our elf grass model, and a function that will add 1 to the value of elf grass when we click it.

```
--environment Variables
local harvesting = workspace:WaitForChild("Harvesting")


--harvesting through click

for _, child in pairs(harvesting:GetChildren()) do
    if child.Name == "ElfGrassModel" then
        child.ClickDetector.MouseClick:Connect(function()
            elfGrass.Value += 1
            child:Destroy()
        end)
    end
end
```

Also, in future, we will track our item values in the GUI, but for now, we will test it by printing the value to the output:

```
--output to debug our harvesting
elfGrass.Changed:Connect(function(Val)
    print("elf grass qty now "..elfGrass.Value)
end)
```

Now, when you run your game and click on them elf grass models in the game, you should see an output saying your quantity of elf grass has gone up. Also the model should disappear.

Now, we have a working harvestable ElfGrass, and the associated variables to track it in our inventory. We need to repeat this process for our Orc Berry and ToadStool. As a reminder, the steps to do this are:

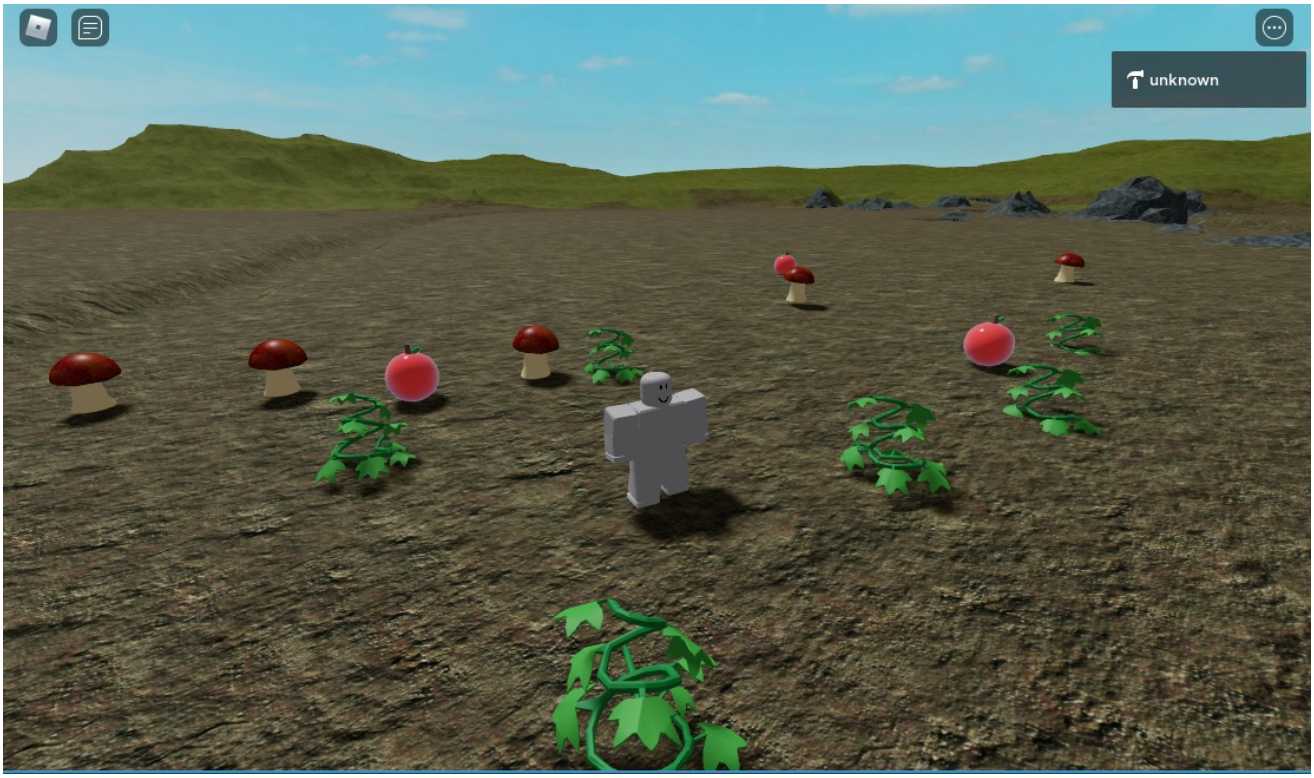
1. Find a suitable model (either in the toolbox, or create one yourself).
2. Add a click detector to the model.
3. Rename it appropriately (OrcBerryModel and ToadStoolModel).

4. Move the model into the "harvesting" folder.
5. Duplicate the model as many times as you want and move it around the workspace.
6. Alter your existing HarvestingScript to include the other harvesting items

When adding to your HarvestingScript, rather than repeat the whole process of looping through the harvesting folder, we can add an elseif line, after our current if statement, that checks for our other items, and adds to the appropriate inventory variable:

```
--harvesting through click

for _, child in pairs(harvesting:GetChildren()) do
    if child.Name == "ElfGrassModel" then
        child.ClickDetector.MouseClick:Connect(function()
            elfGrass.Value += 1
            child:Destroy()
        end)
    elseif child.Name == "OrcBerryModel" then
        child.ClickDetector.MouseClick:Connect(function()
            orcBerry.Value += 1
            child:Destroy()
        end)
    elseif child.Name == "ToadStoolModel" then
        child.ClickDetector.MouseClick:Connect(function()
            toadStool.Value += 1
            child:Destroy()
        end)

    end
end

--output to debug our harvesting
elfGrass.Changed:Connect(function(Val)
    print("elf grass qty now  "..elfGrass.Value)
end)

orcBerry.Changed:Connect(function(Val)
    print("orc Berry qty now  "..orcBerry.Value)
end)

toadStool.Changed:Connect(function(Val)
    print("toadStool qty now  "..toadStool.Value)
end)
```

You should now have a game world littered with harvestable crafting items:



4. Setting up our GUI.

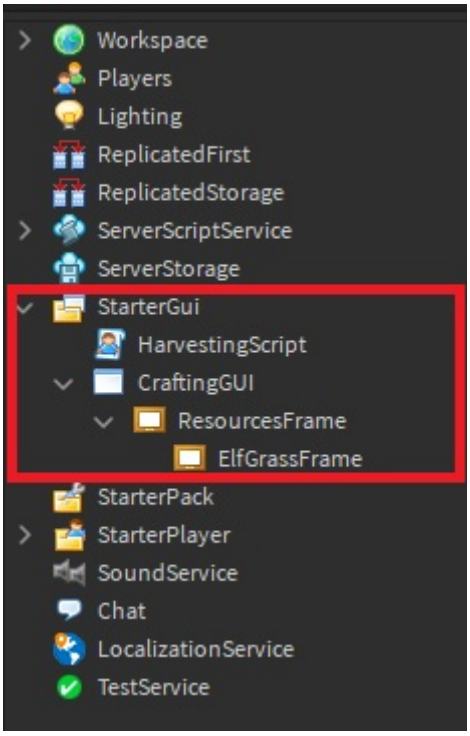
Next we will need to create a GUI (Graphical User Interface) that our players can interact with. We will need several GUI elements to do several things including:

1. Track the number of gathered crafting resources in the players inventory
2. Craft new items from gathered resources
3. Track and use the newly crafted items
4. Display the crafting recipes

4.1 Gathered resources GUI

First we will create the labels that will track the gathered crafting resources. First, create a new ScreenGui under StarterGUI and call it "CraftingGUI".

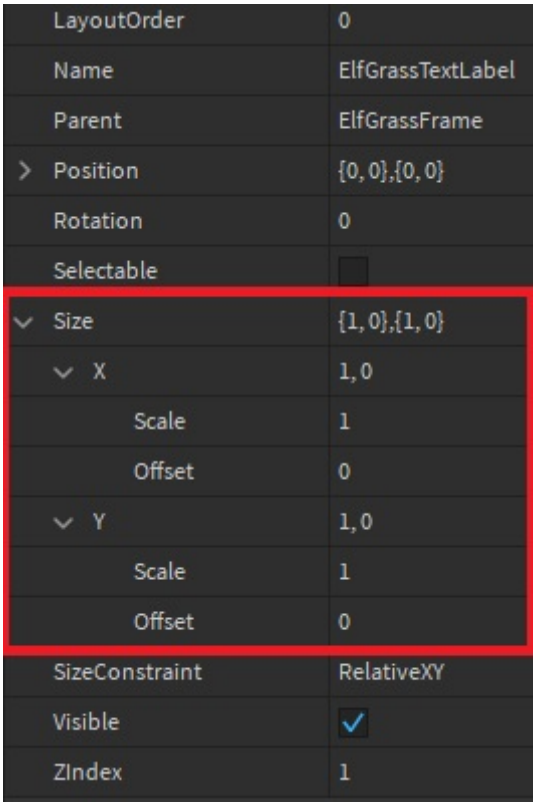
Now under the CraftingGUI create a Frame, and rename it ResourcesFrame. Again, under ResourcesFrame create a new Frame and call it, ElfGrassFrame. Your new GUI hierarchy should look like this:



Now, reposition your Crafting GUI frame on screen, remembering that the LabelsFrame is a child of it, and will moe around with it. Resize the CraftingGUI frame so that it will be big enough for 3 text labels, to track our three resources.

Now, in the properties window of both frames, change the back ground Transparency to 1 and border pixel size to 0. The frames will disappear, but don't worry about that for now.

Now under the LabelsFrame add a new text label. Rename it to "ElfGrassTextLabel" and Change the properties under size to scale 1 and offset 0, for both X and Y like this:



Now, Right click and duplicate your ElfGrassTextLabel Twice, so you have three text labels. Rename two of them to OrcBerryLabel and ToadStoolLabel respectively. Reposition them on scree or using the properties to how you want them to look. Now, change the transparency to 1 and border pixel size to 0.

Now, in the L:abel properties, change the font, color, size, text and other properties to just how you want the label to look. Mind ended up looking like this:

4.2 Gathered Resources GUI Script

Now, finally, we will set up the script that will make the labels reflect the number of items in our inventory

Now, in our existing HarvestingScript, somewhere near the top, with our other variables, add the following:

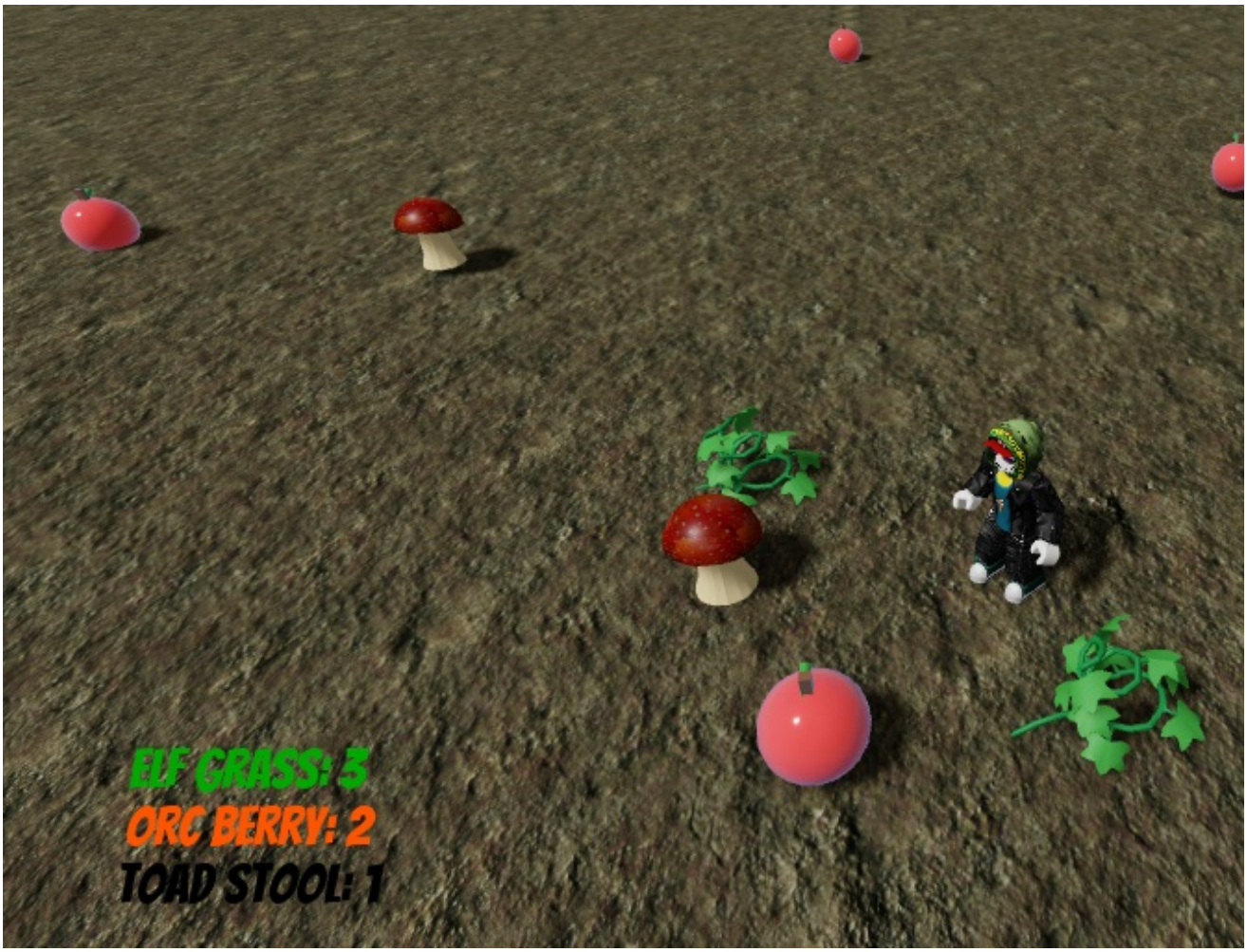
```
--GUI Variables:
local craftingGUI = script.Parent:WaitForChild("CraftingGUI")
local resourcesFrame = craftingGUI:WaitForChild("ResourcesFrame")
local lablesFrame = resourcesFrame:WaitForChild("LablesFrame")
local elfGrassTextLabel = lablesFrame:WaitForChild("ElfGrassTextLabel")
local orcBerryTextLabel = lablesFrame:WaitForChild("OrcBerryTextLabel")
local toadStoolTextLabel =
lablesFrame:WaitForChild("ToadStoolTextLabel")
```

Now, find the section we created before under --output to debug our harvesting. We are going to delete this, and instead of outputting changes to console, change the GUI. To do this enter the following:

```
--output inventory changes to Gui

elfGrass.Changed:Connect(function(val)
    elfGrassTextLabel.Text = "Elf Grass: "..elfGrass.Value
end)

orcBerry.Changed:Connect(function(val)
    orcBerryTextLabel.Text = "Orc Berry: "..orcBerry.Value
```

```
end)

toadStool.Changed:Connect(function(val)
    toadStoolTextLabel.Text = "Toad Stool: "..toadStool.Value
end)
```

Now when you test your game, your GUI should update a you collect items! cool!



5 Craft New Items

5.1 Setting Up Crafting Variables

So, now we have our resource gathering working, we need to craft something form them! in this example we will be crafting potions. First, we will need to set up the variables that will track our potions. So, in the exiting sever script (LevelSeverScript) inside our existing function, after the inventory variables, but before the end) add the following:

```
--sets up the potions folder
    local potions =Instance.new("Folder", player)

    --initialise the potion itmes

    local speedPotion = Instance.new("IntValue", potions)
    speedPotion.Name = "SpeedPotion"
    speedPotion.Value = 0

    local bounceBrew = Instance.new("IntValue", potions)
    bounceBrew.Name = "BounceBrew"
    bounceBrew.Value = 0
```

Now, in our HavestingScript, at the top with the other player variables, add the following:

```
local potions = player:WaitForChild("Potions")
local speedPotion = potions:WaitForChild("SpeedPotion")
local bounceBrew = potions:WaitForChild("BounceBrew")
```

Now we will create a boolean (true or false) to track weather we have the right combination of resources to craft a potion. In the first example, to craft the Speed Potion, you need 2 elf grass and 1 toad stool. To track this, we need to set up our bool value.

yet again, in the exiting sever script (LevelSeverScript) inside our existing function, after the inventory variables, but before the end) add the following:

```
--crafing boolvalues

    local canCraftSpeedPotion = Instance.new("BoolValue", potions)
    canCraftSpeedPotion.Name = "CanCraftSpeedPotion"
    canCraftSpeedPotion.Value = false
```

Now back in our HavestingScript, at the top, add the following:

```
-- crafting bools
local canCraftSpeedPotion = potions:WaitForChild("CanCraftSpeedPotion")
```

And, inside our existing functions that track when our resources change, we need to check if we have the correct amount to create the potion. We need to check both resources in both functions, as we don't know what order the player will pick them up in. Also, we also need to check if they have less than the required amount, as the player will use up resources over time. So, in the HavestingScript, replace the existing functions under *--output inventory changes to Gui*, with this:

```
--output inventory changes to Gui

elfGrass.Changed:Connect(function(val)
    elfGrassTextLabel.Text = "Elf Grass: "..elfGrass.Value

    --checking crafting boolean condition
    if elfGrass.Value >= 2 and toadStool.Value >=1 then
        canCraftSpeedPotion.Value = true
    elseif elfGrass.Value <= 2 and toadStool.Value <=1 then
        canCraftSpeedPotion.Value = false
    end
end)

orcBerry.Changed:Connect(function(val)
    orcBerryTextLabel.Text = "Orc Berry: "..orcBerry.Value
    --checking crafing variables


end)

toadStool.Changed:Connect(function(val)
    toadStoolTextLabel.Text = "Toad Stool: "..toadStool.Value

    --checking crafting boolean condition
    if elfGrass.Value >= 2 and toadStool.Value >=1 then
        canCraftSpeedPotion.Value = true
    elseif elfGrass.Value <= 2 and toadStool.Value <=1 then
        canCraftSpeedPotion.Value = false
    end

end)


--crafting debug

canCraftSpeedPotion.Changed:Connect(function(NewValue)
    print("can craft speed potion is now..."..tostring(NewValue))

end)
```

Note Well: this might seem like a complicated way to track this, why track the state of the canCraft Boolean every time the resources change rather than just when we want to craft?? We need to because we will use this Boolean to enable the buttons to craft the potion. We will do this next.

However, instead of rewriting this group of code over and over again, we can create a function, that contains the check for the correct recourse amounts, and call this function when needed. The function looks like this:

```
--checks the resources amounts for the speedpotion recipie and assigns
the boolvalue appropreatly.
function speedcraftCheck()
    if elfGrass.Value >= 2 and toadStool.Value >=1 then
        canCraftSpeedPotion.Value = true
    elseif elfGrass.Value <= 2 and toadStool.Value <=1 then
        canCraftSpeedPotion.Value = false
    end

end
```

Now, every time we call our function, simply by typing speedcraftCheck(), it will perform this set of instructions.

So, we will replace this where we have used it previously with this function call. Now this section of our Harvesting Script will look like this:

```
--output inventory changes to Gui

elfGrass.Changed:Connect(function(val)
    elfGrassTextLabel.Text = "Elf Grass: "..elfGrass.Value

    --checking crafting boolean condition
    speedcraftCheck()
end)

orcBerry.Changed:Connect(function(val)
    orcBerryTextLabel.Text = "Orc Berry: "..orcBerry.Value
    --checking crafing variables


end)

toadStool.Changed:Connect(function(val)
    toadStoolTextLabel.Text = "Toad Stool: "..toadStool.Value

    --checking crafting boolean condition
    speedcraftCheck()

end)

--checks the resources amounts for the speedpotion recipe and assigns
the boolvalue appropreatly.
function speedcraftCheck()
    if elfGrass.Value >= 2 and toadStool.Value >=1 then
```

```
            canCraftSpeedPotion.Value = true
        elseif elfGrass.Value <= 2 and toadStool.Value <=1 then
            canCraftSpeedPotion.Value = false
        end

    end
```

5.2 Crafting Potions via GUI

Finally, we will add the GUI elements to allow our player to craft potions, if they have the right ingredients.

We will have the following:

- a button to bring up or close our Crafting Menu.
- a button to bring up or close our Recipe Menu.
- a crafting menu with buttons for each potion *(these will be enabled \ disable and change colour depending on if we have the right ingredients)*
- a menu displaying our potion recipes

But now, you should have some idea about make, sizing and styling GUI elements, so I won't be covering each one in detail again. if you have any issues, refer here:

https://developer.roblox.com/en-us/api-reference/class/UIScale

https://developer.roblox.com/en-us/api-reference/class/GuiObject

So, we will add The following elements under CraftingGUI:

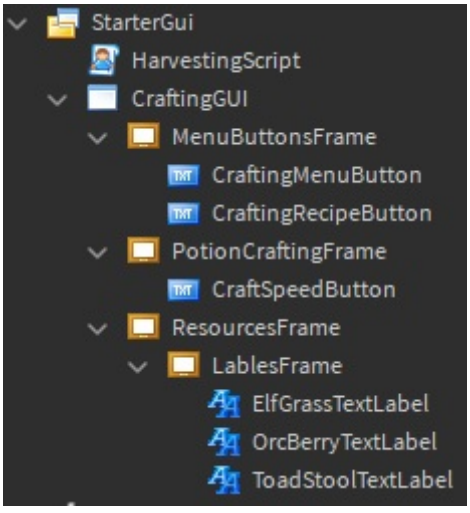
Menu Buttons Frame

- Crafting Menu Button
- Recipe Menu Button

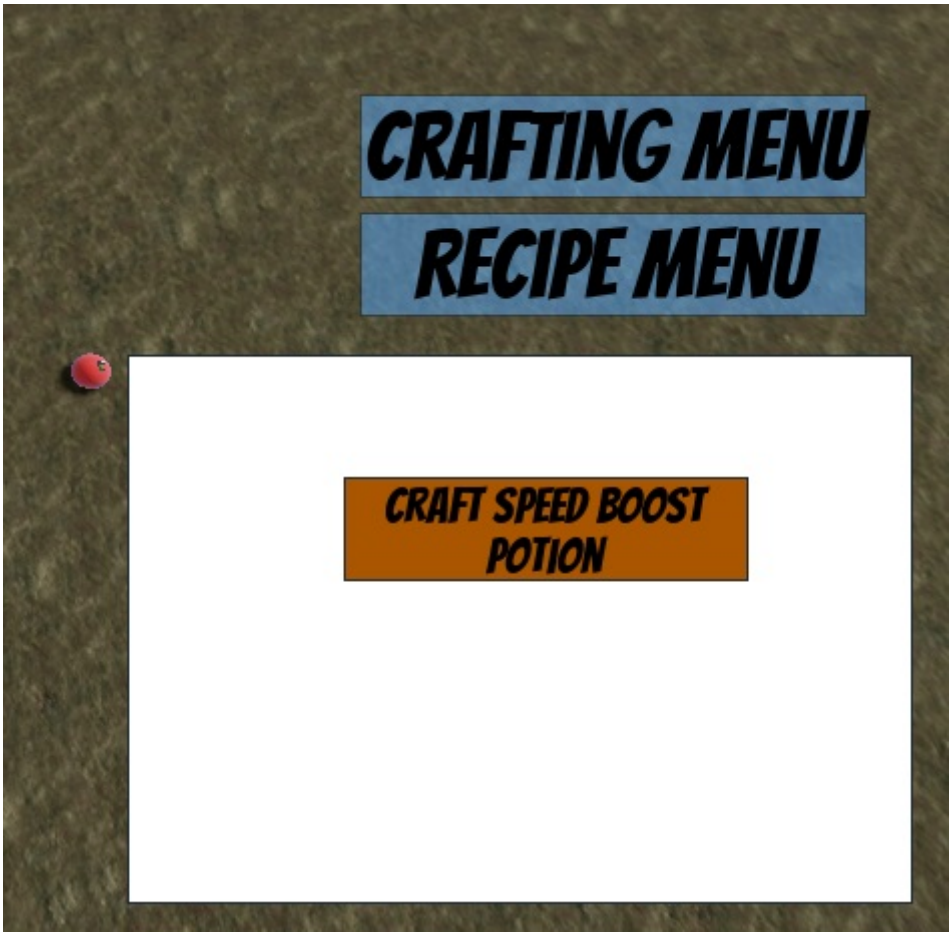
PotionCraftingFrame

- CraftSpeedButton

The GUI Hierarchy should now look like this:



And, my Menu currently looks like this, yours will vary depending on how you positioned and styled it:



Our Two Menu Buttons will bring up our different Menus, our craft speed boost will be enabled \ disabled and change colour based on our already Created BoolValue.

So, to make this work, we first need to add all these elements to our Harvesting Script, under the existing GUI variables, add the following:

```
local menuButtonFrame = craftingGUI:WaitForChild("MenuButtonsFrame")
local craftingMenuButton =
menuButtonFrame:WaitForChild("CraftingMenuButton")
local recipeButton =
menuButtonFrame:WaitForChild("CraftingRecipeButton")

local potionCraftingFrame =
 craftingGUI:WaitForChild("PotionCraftingFrame")
local craftSpeedButton =
potionCraftingFrame:WaitForChild("CraftSpeedButton")
```

Now, we will add a brief script, that will mean that the crafting menu button will make our crafting menu visible \ invisible. In the harvesting script add the following

```
--menu button actions
local function craftButtonActivated()
    potionCraftingFrame.Visible = not potionCraftingFrame.Visible
end

craftingMenuButton.Activated:Connect(craftButtonActivated)
```

Now, in the properties window, uncheck the "Visible" property of the potioncraftingFrame. Now, you can use the menu button to bring the crafting menu up or down. In future, you can use the tween service to create a smooth transition animation, but for now we will just plain open and close it.

Now, we actually want to be able to craft some items, if we have the right ingredients. First, we will use colour feedback to tell our player if they can craft it or not. We do this using the following script in our HarvestingScript:

```
--changing crafting button colours

canCraftSpeedPotion.Changed:Connect(function(NewValue)
    print("can craft speed potion is now..."..tostring(NewValue))
    if canCraftSpeedPotion.Value == true then
        craftSpeedButton.BackgroundColor3 = Color3.new(0, 1, 0)
    elseif canCraftSpeedPotion.Value == false then
        craftSpeedButton.BackgroundColor3 =Color3.new(0.666667,
0.333333, 0)
    end
end)
```

Finally, if the player has the right ingredients and clicks the craft potion button, we want to add the potion to their inventory, and subtract the used ingredients:

```
--crafting speed potion

craftSpeedButton.Activated:Connect(function()

    if canCraftSpeedPotion.Value == true then
        speedPotion.Value += 1
        elfGrass.Value -= 2
        toadStool.Value-=1
        --output for debugging
        print("Speed Potion Crafted")
        print(speedPotion.Value)
    end
```

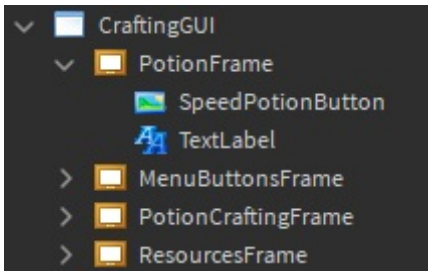
```
end)
```

Next to add andrew:

6. Using Crafted Items (potions)

So, now we need to allow our player to actually use the crafted items!

First we will add a few final UI elements. Add a new frame, and under it, an image button and text label. Your UI hierarchy will now look like this:



Add an image to your image button by searching in the toolkit, right clicking the desired image, and copying it's asset ID. Now in the properties, paste this into the "Image" property. Again, you can style these elements however you want, this is how mine ended up looking:

Now we need to make it work, in your HarvestingScript, first add the following variables:

```
local potionFrame = craftingGUI:WaitForChild("PotionFrame")
local speedPotionButton = potionFrame:WaitForChild("SpeedPotionButton")
local spTextLabel = potionFrame:WaitForChild("TextLabel")
```

Now find the section that starts with the comment *--output inventory changes to Gui* and the the bottom add:

```
speedPotion.Changed:Connect(function(val)
    spTextLabel.Text = "Speed Potion x "..speedPotion.Value
end)
```

Now, at the very bottom of the script add:

```
--using speed potion
local cantrigger = true


local cooldown= 10
local boostTime= 9

speedPotionButton.Activated:Connect(function()

    if speedPotion.Value >0 and cantrigger then
        cantrigger = false
        speedPotion.Value -= 1
        game.Players.LocalPlayer.Character.Humanoid.WalkSpeed += 100
        print(game.Players.LocalPlayer.Character.Humanoid.WalkSpeed)
        spTextLabel.BackgroundColor3 = Color3.new(255,0,0)
        wait(boostTime)
        game.Players.LocalPlayer.Character.Humanoid.WalkSpeed =16
        print(game.Players.LocalPlayer.Character.Humanoid.WalkSpeed)
        wait(cooldown)
        spTextLabel.BackgroundColor3= Color3.new(0,85,0)

        cantrigger = true
    end
end)
```