CODE NINJAS®

# Silver Belt Ninja Guide

# Activity 14: Scavenger Hunt Deluxe

# Conversational UI in Games

Often in games, the player is conversing with NPCs (Non-Playable Characters) to progress the main story, questlines, or to trade items. These interactions often use custom-built UI meant for character dialogue to display the information in an organized manner on screen. It depends on the game, but dialogue boxes typically show:

1. **Who** is speaking (through an icon or text)
2. **What** they are saying
3. An option to **exit or progress** the conversation


CONVERSATIONAL UI IN STARDEW VALLEY

Some may even have **response options** which can be seen frequently in Legend of Zelda: Breath of the Wild and sometimes in Undertale:



RESPONSE OPTIONS SHOWN IN LEGEND OF ZELDA: BREATH OF THE WILD (LEFT) AND UNDERTALE (RIGHT)

# Unique Names

**Unique Names** allow nodes to be referenced from scripts by prefixing their name with a percentage symbol %.

For example, instead of writing this:

```
# ---------------------------------------------------------------------
@onready var player: Node2D = get_tree().current_scene.get_node("Player")  #
# ---------------------------------------------------------------------
```

You can assign the Player node its own **Unique Name** then replace the above code with this:

```
# ---------------------------------------------------------------------
@onready var player: Node2D = %Player                                    #
# ---------------------------------------------------------------------
```

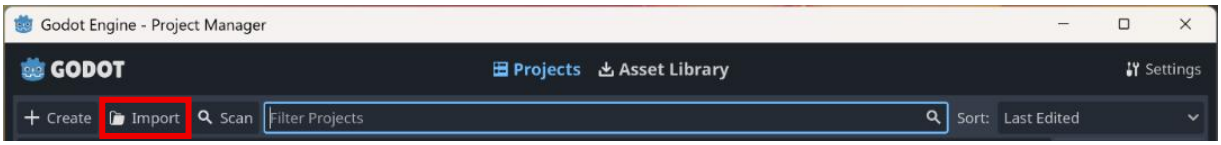
Unique Names can be set in the Scene panel by right clicking a node and selecting **% Access as Unique Name**.

# Activity 14: Scavenger Hunt Deluxe

Your mission: Design a simple dialogue system for the player to communicate with an NPC (non-playable character) in the Scavenger Hunt environment! This NPC, Jack, will ask the player to find a specific item. Then, the player will have to explore the environment and bring back the desired item to win!

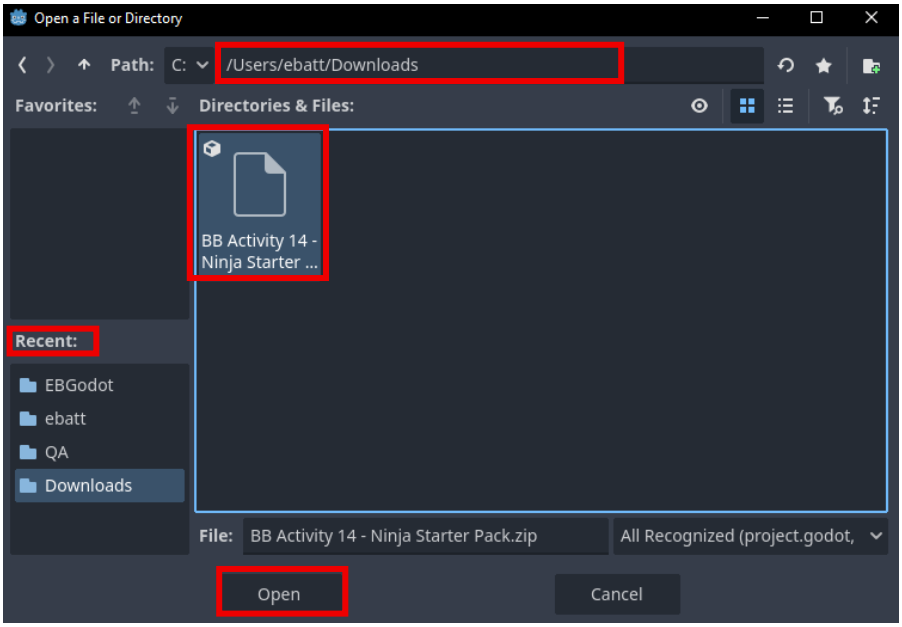
In the starter project, Jack doesn't talk with the player. Jack needs a dialogue box!

1 Open Godot and click **Import.**



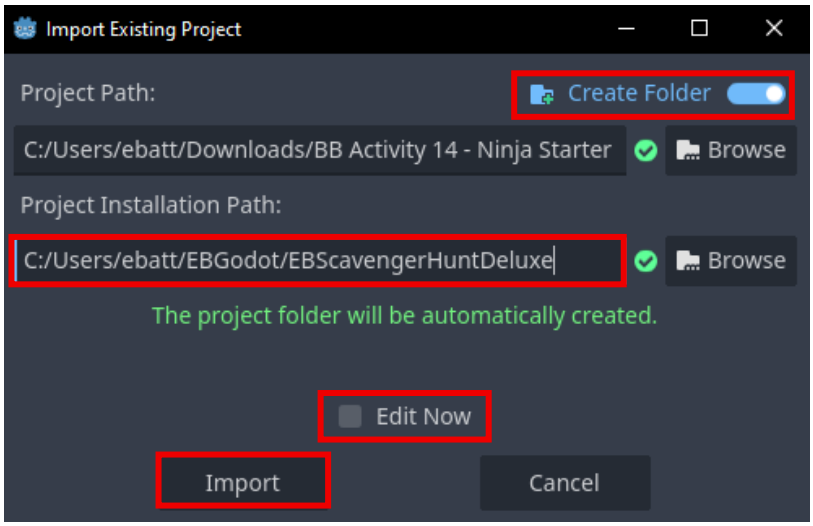

2 In the File Directory, navigate to the correct file path.

Select **SB Activity 14 – Ninja Starter Pack.zip** and click **Open**.




3 Update the **Project Installation Path** and make sure **Create Folder** is enabled.

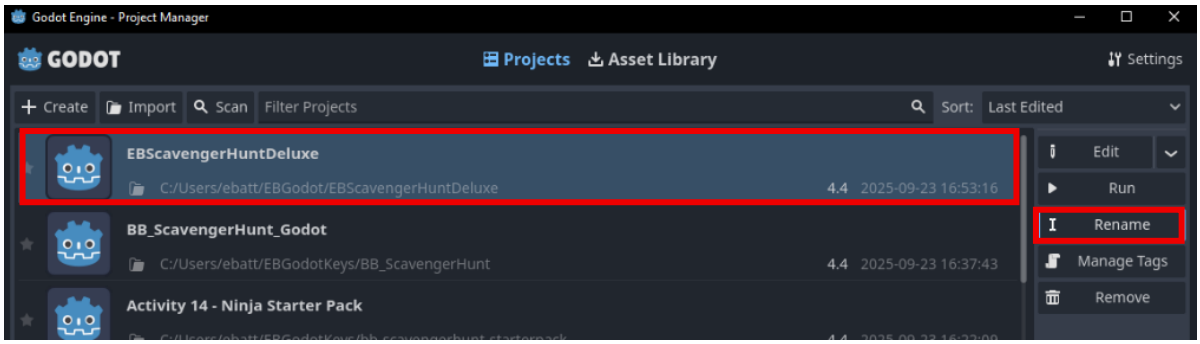
**Uncheck Edit Now** and click **Import**.



**4** The project will appear at the top. Click on the project and select **Rename** on the right.

Update the Project Name to **[YourInitials]ScavengerHuntDeluxe**.

Once renamed, select the project and click **Edit** to open the starter code.

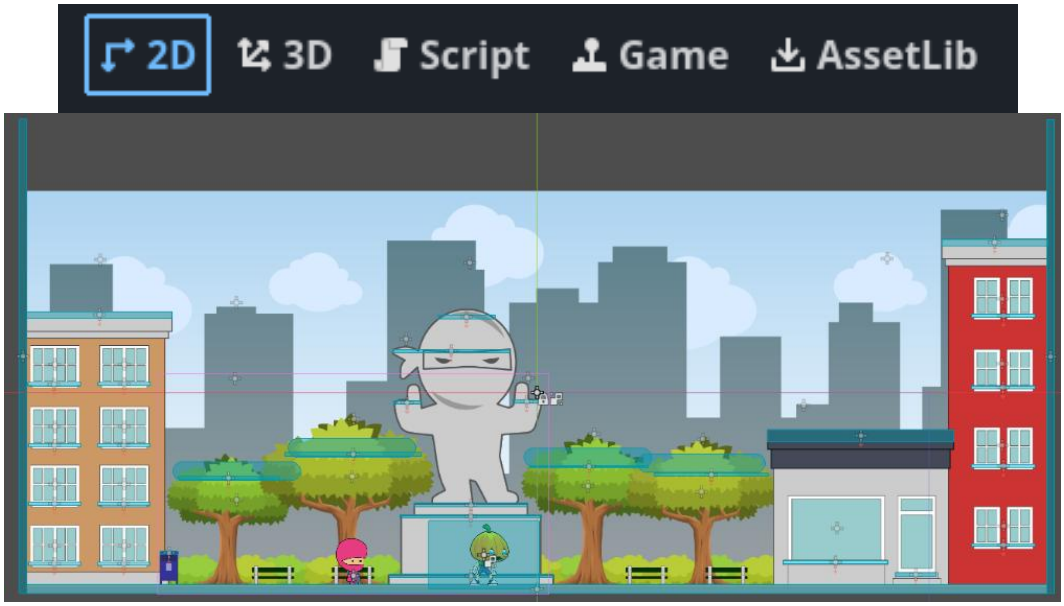



**5** Navigate to the 2D workspace.

Notice that the scene already has many nodes, including:

- Scavenger Hunt assets and platforms from BB Activity 02
- Area2Ds that toggle leaf particles to appear on the trees
- Spawn points littered across the scene to spawn collectibles
- Jack, who is located under the statue...!

Jack will ask the player to find a specific collectible. Then, the player must return it to Jack.



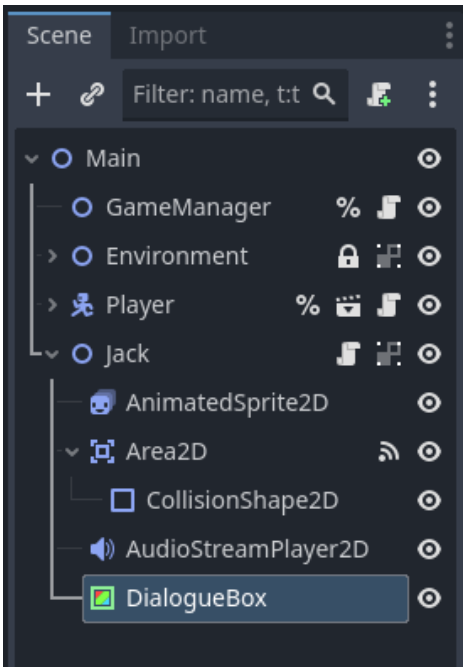
6 Playtest the project. What happens when the player approaches Jack?

## 7

When the player approaches Jack, all the coins spawn in, but Jack doesn't ask the player to retrieve anything!

The first thing that needs to be done is to create a dialogue box.

Add a new **ColorRect** as a child to Jack and rename it to **DialogueBox**.



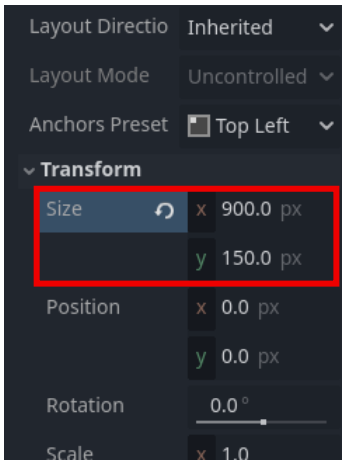

0

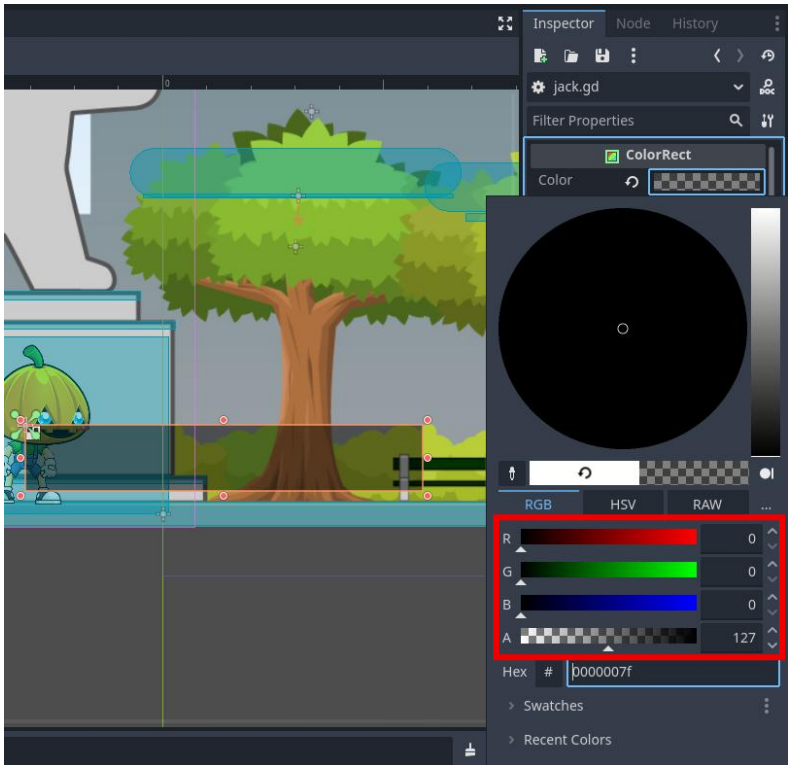
**New Concept: ColorRect**

**ColorRect** is a control node that has one property: **color**. It displays a rectangle that is filled with **color** which has support for transparency (alpha).

**8** In the **Inspector**, click the **Layout** and **Transform** drop-down menus to set the x and y sizes to **900px** and **150px** respectively.

Then, set the Color to black with the Alpha value set to **127** so the dialogue box is 50% transparent.



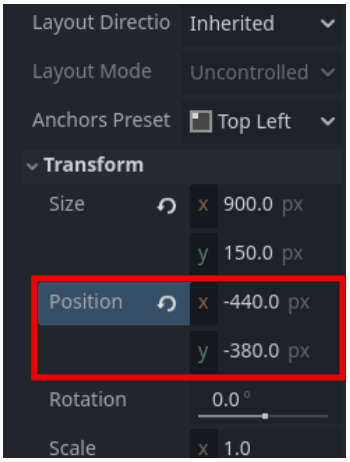



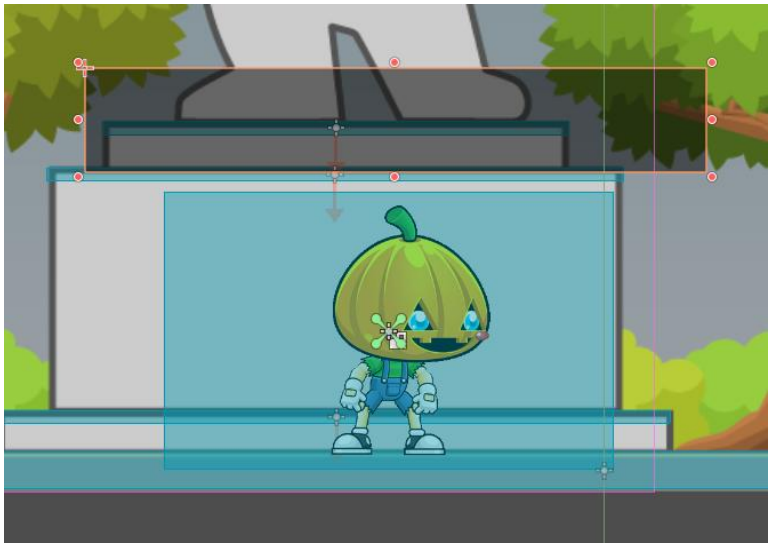

**Pro Tip:**

The RGBA sliders in Godot's color picker are 8-bit, which means they can have $2^8 = 256$ total values. The slider for each value ranges from 0 to 255, so halfway between these values would be 127.

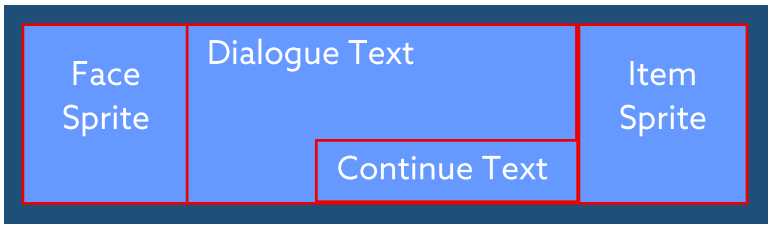
**9** Move the dialogue box so it hovers above Jack's head.

Tinker with the position values, but **x = -440px** and **y = -380px** will place the dialogue box right above Jack.



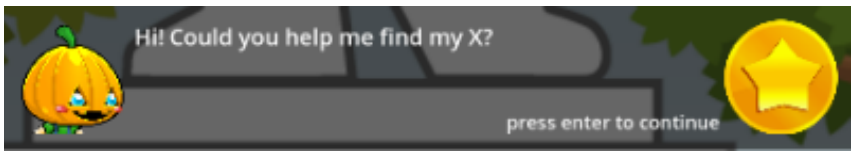
**10** Now that the background has been set for the dialogue box, it's time to discuss the layout of the items inside of it.



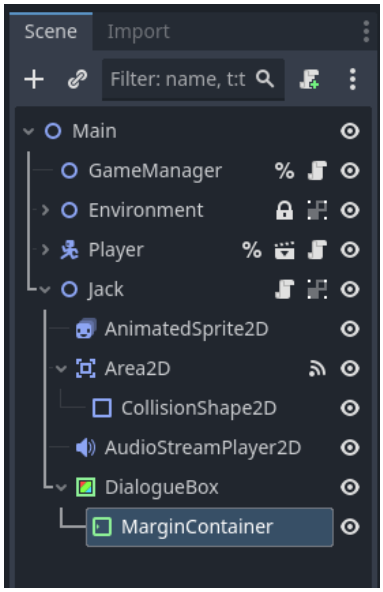

There should be a **margin** on the inside of the box so that none of the content touches the edges. Inside the dialogue box, there should be three items aligned horizontally:

1. Face Sprite
2. Dialogue Text (+ Continue Text)
3. Item Sprite

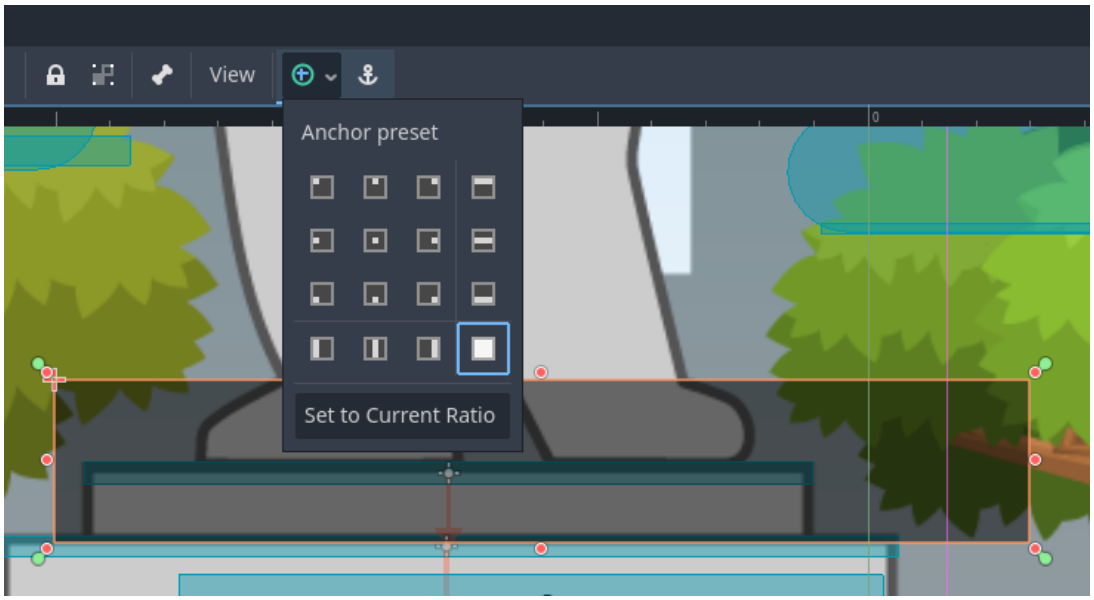
You will be building this layout in the following steps.



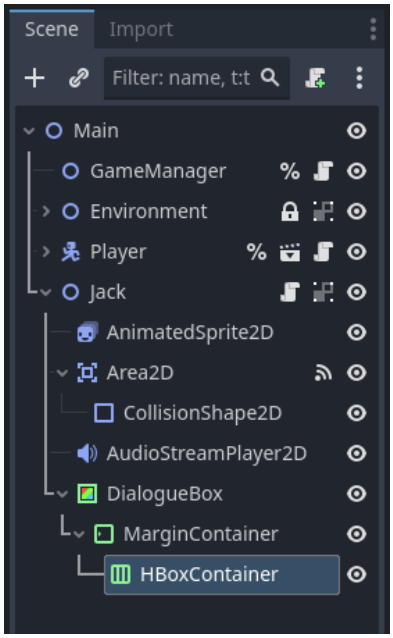
**11** To start, add a **MarginContainer** as a child node to **DialogueBox**.




In the **Toolbar**, locate the **Anchor Preset** icon and select **Full Rect**. This will expand the **MarginContainer** to fill the entirety of **DialogueBox**.



**12** Add an **HBoxContainer** as a child node to **MarginContainer**. This will allow the three items listed earlier to be aligned horizontally.

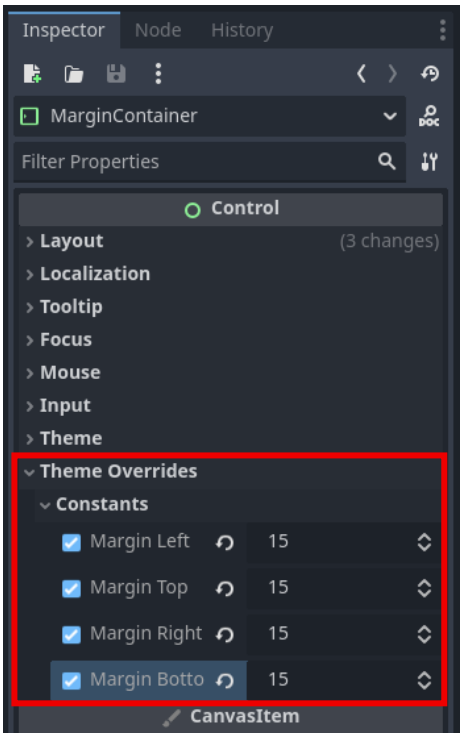



Hmm, something seems off. Notice that no margins have been applied yet based on the **HBoxContainer's** selection in the viewport filling the **DialogueBox**.

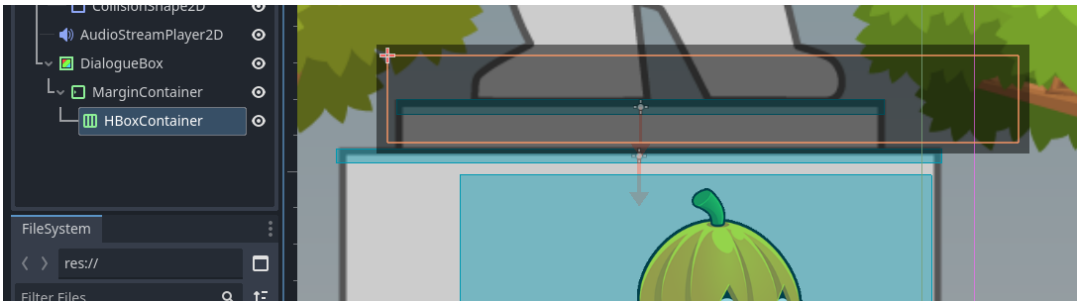
**New Concept: HBoxContainer**

The **HBoxContainer** node determines the position of its child nodes by arranging them horizontally and removes their ability to set their anchor preset.

**13** Select **MarginContainer**, and in the **Inspector** open the **Theme Overrides** and **Constants** drop-down menus. Set all the margins to **15** pixels.

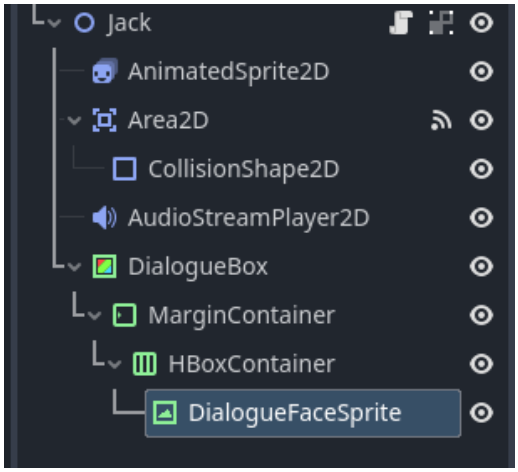


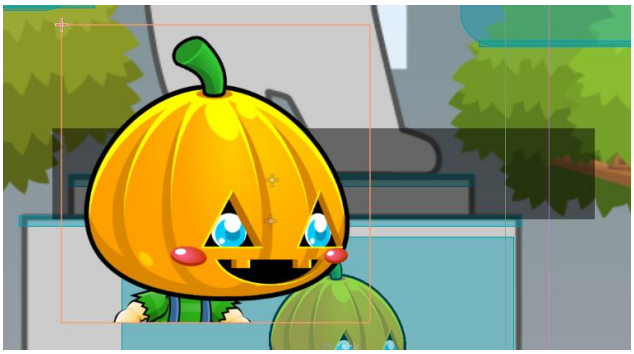

Select the **HBoxContainer** and zoom in to look at the viewport to confirm that the margins have been applied.



**14** Add a **TextureRect** as a child node to **HBoxContainer**, and rename it to **DialogueFaceSprite**.

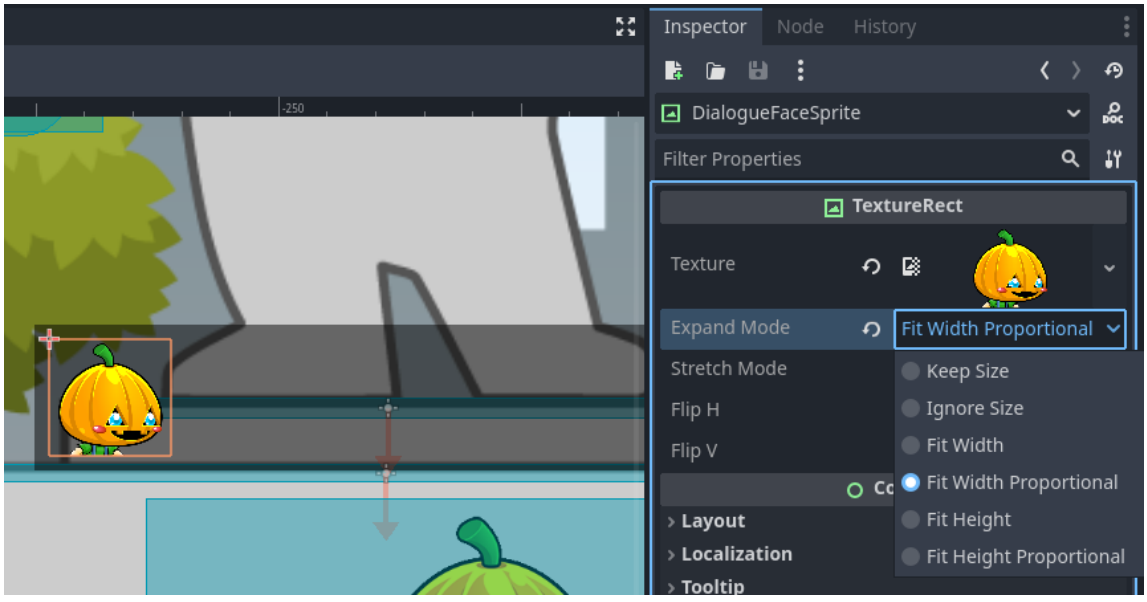
In the **Inspector**, set its **Texture** to **jackHead.png** by using **Quick Load**.



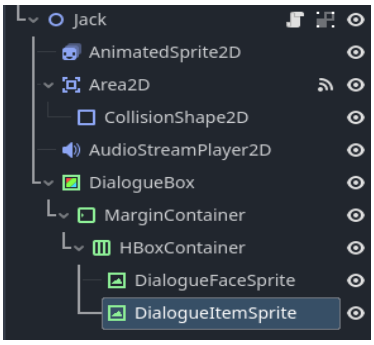
**15** Uh oh, it's way too big! This is because the **Expand Mode** is set to **Keep Size**, so the image uses its native resolution instead of scaling to fit the height of the **HBoxContainer**!

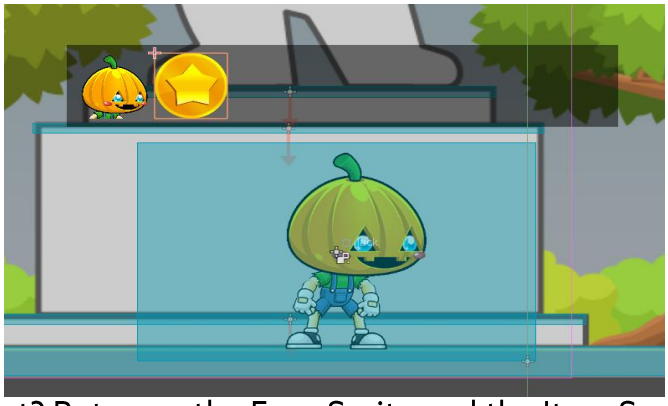
In the **Inspector**, set **Expand Mode** to **Fit Width Proportional**. This will set the height of **DialogueFaceSprite** to the height of the container it is in. Then, it will set the width so the aspect ratio of the **Texture** remains the same.



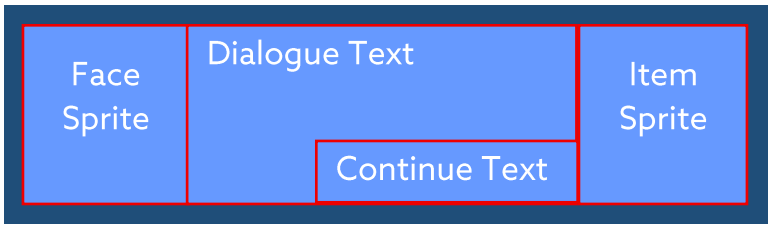
**16** With **DialogueFaceSprite** selected in Scene, press **CTRL + D** to duplicate it.

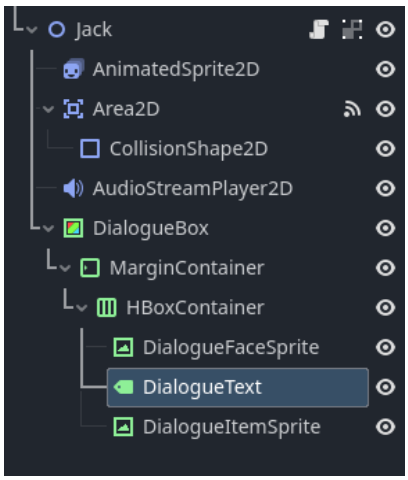
Rename the new node to **DialogueItemSprite**, reset its texture, then use **Quick Load** to set its texture to **star coin.png** as a placeholder. This texture will be updated in code during runtime.

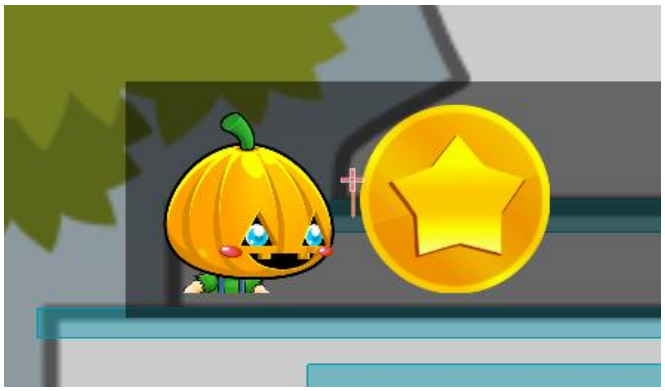


**17** Remember the complete layout? Between the Face Sprite and the Item Sprite, there needs to be Dialogue Text with Continue Text in its bottom-right corner.

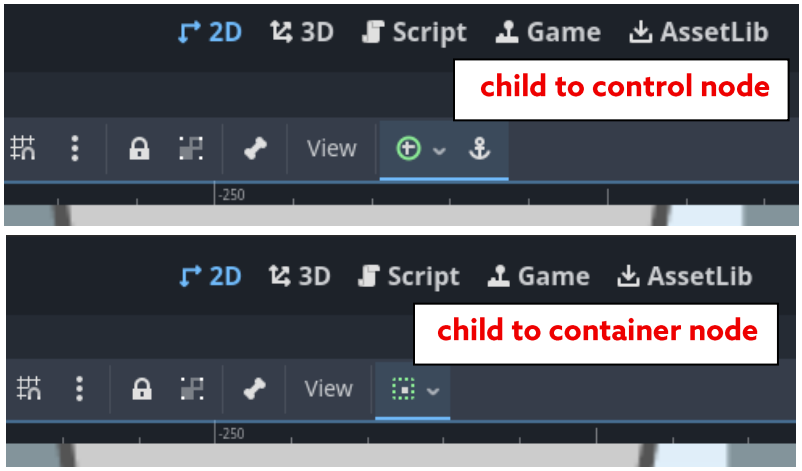


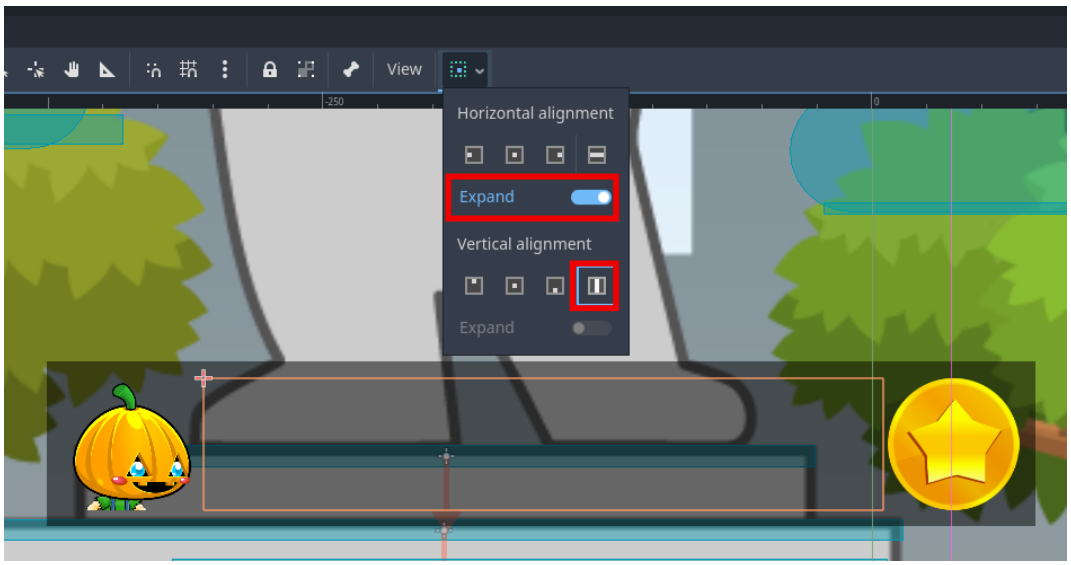



Add a **Label** as a child to **HBoxContainer** and rename it to **DialogueText**. Reposition it in the hierarchy so it is between **DialogueFaceSprite** and **DialogueItemSprite**. Notice a small selection appears between the two sprites in the viewport.

**18** Since **HBoxContainer** is a **Container** instead of a traditional **Control** node, the **Anchor Preset** button is replaced with a **Sizing Settings** button in the **Toolbar**.




To make **DialogueText** fill the rest of the **HBoxContainer**, its sizing settings need to be updated. Set the **Horizontal alignment** to **Expand** and set the **Vertical alignment** to **Fill**.



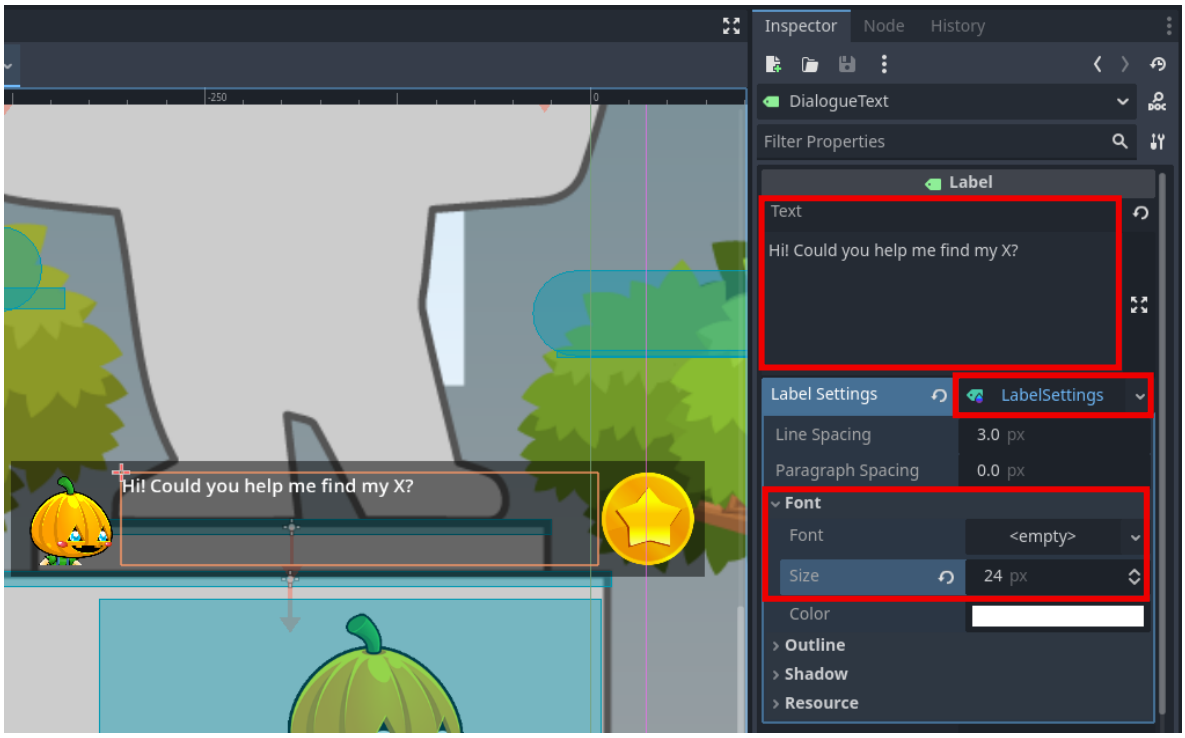

**Pro Tip:**

The node hierarchy for containers is **Node > CanvasItem > Control > Container > MarginContainer/HBoxContainer/...**

**19** In the Inspector, set the **Text** to some placeholder text:

"Hi! Could you help me find my X?"

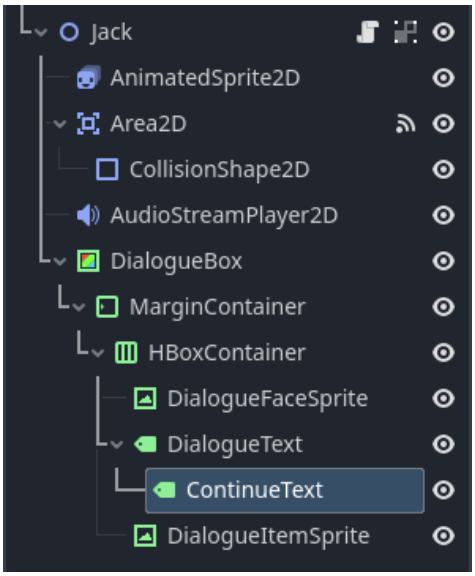
Then, set **Label Settings** to **New LabelSettings**. Select **New LabelSettings** to open the **Font** drop-down menu, and set the **Size** to 24px.



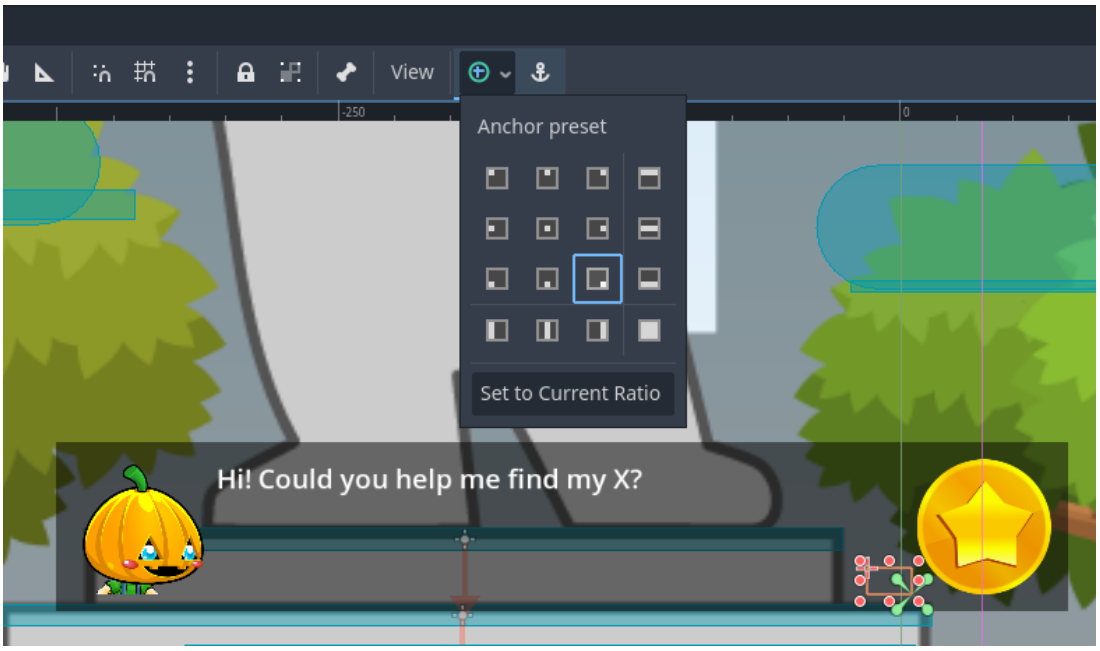
## 20

The player should get a prompt to close the dialogue box.

Add a **Label** as a child node to **DialogueText** and rename it to **ContinueText**.

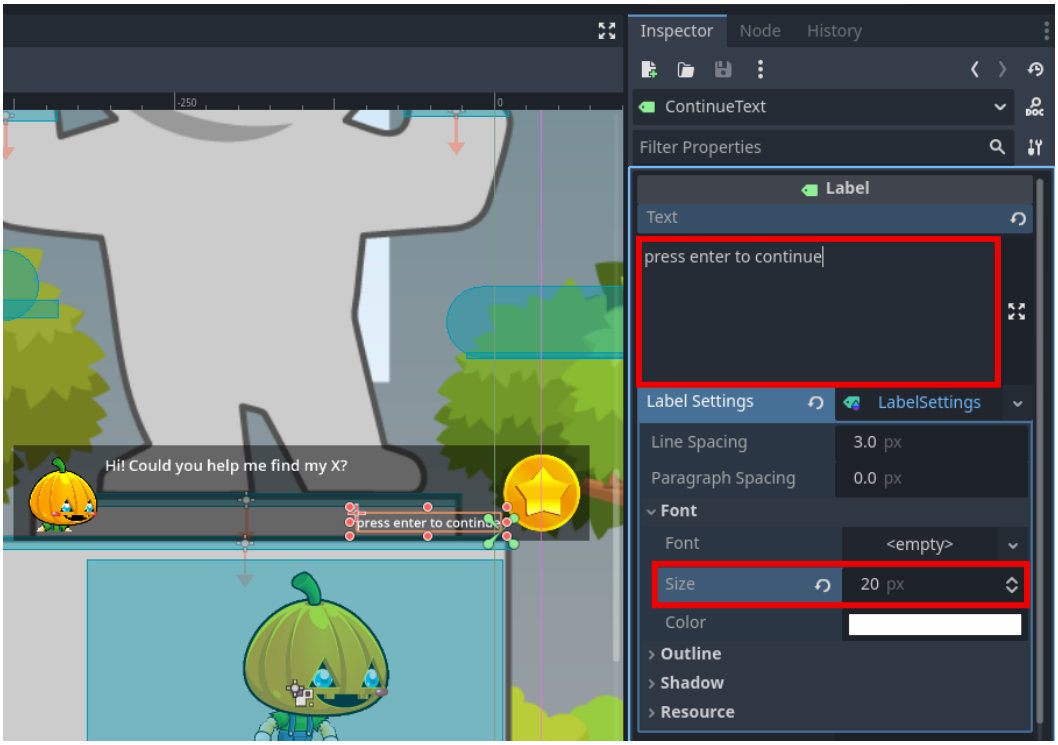



Since **DialogueText** is not a container node, it allows its children to be anchored to certain locations with the **Anchor Preset** button. Set the **Anchor Preset** of **ContinueText** to the bottom right.

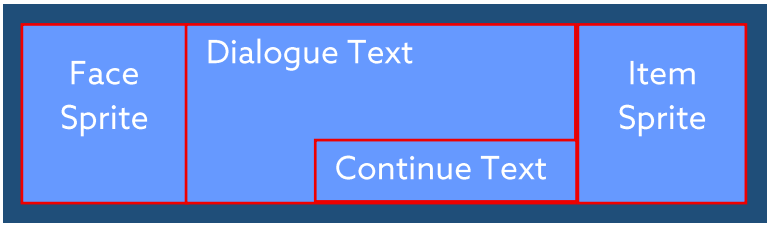


**21** In the Inspector, set the **Text** to “press enter to continue”.

Then, set **Label Settings** to **New LabelSettings**, open the **Font** drop-down menu, and set the **Size** to 20px.



**22** Playtest the project! Refer to previous steps and the layout graphic if the dialogue box does not appear as expected.



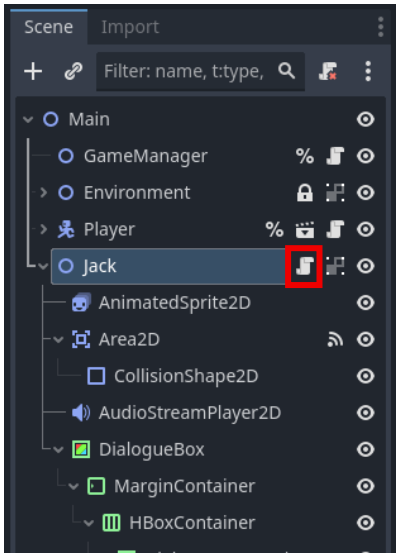
Pause for **Sensei Stop #1**!

Check in with a Code Sensei before moving on. Make sure the dialogue box's nodes are set up properly.

**Reminder:** Save your work!

**23** Now that the nodes have been set up, it's time to code the dialogue box.

Open the **jack.gd** script which is attached to **Jack** and navigate to **TODO 1**.




```gdscript
class_name Jack
extends Node2D

# References
@onready var player: Node = %Player
@onready var manager: GameManager = %GameManager

@onready var audio_player: AudioStreamPlayer2D = $AudioStreamPlayer2D
@onready var jack_sprite: AnimatedSprite2D = $AnimatedSprite2D
# --------------------
# TODO 1
# Reference dialogue
# --------------------

var pickups_spawned: bool = false
var desired_item_id: String
var item_found: bool = false

# --------------------
# TODO 2 + PY
# select_random_item() function
# --------------------
```

## 24

The following nodes will be used to code the dialogue box:

1. **DialogueBox** (ColorRect)
2. **DialogueText** (Label)
3. **DialogueItemSprite** (TextureRect)

Add the code necessary to reference these nodes under **TODO 1**.

```
10  # --------------------
11  # TODO 1
12  # Reference dialogue
13  # --------------------
14  @onready var dialogue_box: ColorRect = $DialogueBox
15
16
```

DialogueText

**Pro Tip**:

Drag a node from **Scene** into the code view onto the desired line of code. Hold CTRL, then release mouse click. This is an easy way to make an @onready reference to the node.

**25** Check the code! Update the script as needed.

```
# --------------------
# TODO 1
# Reference dialogue
# --------------------
@onready var dialogue_box: ColorRect = $DialogueBox
@onready var dialogue_text: Label = $DialogueBox/MarginContainer/HBoxContainer/DialogueText
@onready var dialogue_item_sprite: TextureRect = $DialogueBox/MarginContainer/HBoxContainer/DialogueItemSprite

```

**26** Navigate to **TODO 2**. Define a new function called `select_random_item()` which takes no parameters and returns `void`.

```
# --------------------
# TODO 2 + PY
# select_random_item() function
# --------------------
# select_random_item ???

```

Inside the function, Jack will select a random value from `manager.item_ids` as the desired item for the player to find. The `item_ids` array found in **game_manager.gd** is shown below.

```
# Array of item string ids. Matches the sprites in sprite_frames
var item_ids: Array[String] = [
    "film", "balloons", "life preserver", "bullseye", "bubble pipe",
    "key", "fish", "birdhouse", "red airhorn", "magic hat"
]
```

**27** Instead of creating a new variable and setting it equal to a random index in an array, a cleaner alternative is to use `pick_random()`. It acts as a shorthand for the random index selection.

> `pick_random()`: part of the Array class, it returns a random member of the array.
>
> **Parameters:** *none*
>
> **Returns (*Variant*)**: a random member of the array

**28** Inside of the `select_random_item()` function, set the `desired_item_id` class-level variable to a random member of the `manager.item_ids` array by using `pick_random()`.

```
# --------------------
# TODO 2 + PY
# select_random_item() function
# --------------------
# select_random_item ???
#     desired_item_id ???
```

**Reminder:**

Class-level describes a variable that belongs to the entire script rather than a single function. At the top of the script, the variables that are outside of any function are called "class-level" variables.

**29** After, call the `write_dialogue()` function with the String parameter:

**"Hi! Can you help me find my " + desired_item_id + "?"**

```
# --------------------
# TODO 2 + PY
# select_random_item() function
# --------------------
# select_random_item ???
#     desired_item_id ???
#     write_dialogue ???
```

Don't worry about the error – the next steps go through creating the new `write_dialogue()` function that will update the dialogue box's nodes.

**30** Check the code! Update the script as needed.

```
# --------------------
# TODO 2 + PY
# select_random_item() function
# --------------------
func select_random_item() -> void:
    desired_item_id = manager.item_ids.pick_random()
    write_dialogue("Hi! Can you help me find my " + desired_item_id + "?")

```

**31** Before creating the `write_dialogue()` function, it's important to use the `select_random_item()` function where it's needed. The first time that Jack talks to the player, Jack needs to select a random item.

Navigate to **TODO 3** and observe the surrounding code. **TODO 3** is located inside of the `_on_area_2d_body_entered()` function which is called from the **body_entered** signal by the **Area2D** surrounding Jack. It is also located inside of an `if`-statement that checks if the `pickups_spawned` class-level variable is false. If so, `pickups_spawned` is set to true, guaranteeing the code inside the `if`-statement only runs once.

Call `select_random_item()` inside the `if`-statement.

```
    # The first time, spawn collectibles
    if not pickups_spawned:
        pickups_spawned = true
        manager.call_deferred("spawn_pickups")
        # --------------------
        # TODO 3
        # call select_random_item()
        # --------------------
        select_random_item()

```

**32** Navigate to **TODO 4**. Define a new function called `write_dialogue()` which takes a `message` parameter of type `String` and returns `void`.

Inside the function, set the `text` property of the `dialogue_text` label equal to the `message` parameter.

```
30  # --------------------
31  # TODO 4
32  # write_dialogue() function
33  # --------------------
34  # write_dialogue ???
35  #     dialogue_text.text ???
36
```

**33** How can the `texture` property of the `dialogue_item_sprite` be set equal to the correct texture? Luckily, **game_manager.gd** handles storing the textures into code by creating a dictionary called `texture_dict`. Values can be retrieved from this dictionary like an array, but instead of asking for an index, it asks for the first value in a mapping. In this case, it asks for a `String` to return the corresponding `Texture2D`.

```
15  # A dictionary that maps ids to textures
16  var texture_dict: Dictionary[String, Texture2D] = {}
```

Set the `texture` property of `dialogue_item_sprite` equal to the Texture2D retrieved from accessing `manager.texture_dict` with the value of `desired_item_id`.

```
30  # --------------------
31  # TODO 4
32  # write_dialogue() function
33  # --------------------
34  # write_dialogue ???
35  #     dialogue_text.text ???
36  #     dialogue_item_sprite.texture ???
37
```

## 34

Check the code! Update the script as needed.

```
30 # --------------------
31 # TODO 4
32 # write_dialogue() function
33 # --------------------
34 func write_dialogue(message: String) -> void:
35     dialogue_text.text = message
36     dialogue_item_sprite.texture = manager.texture_dict[desired_item_id]
37
```

## 35

Playtest the project! Refer to previous steps and the layout graphic if it does not appear as expected.



Pause for **Sensei Stop #2**!

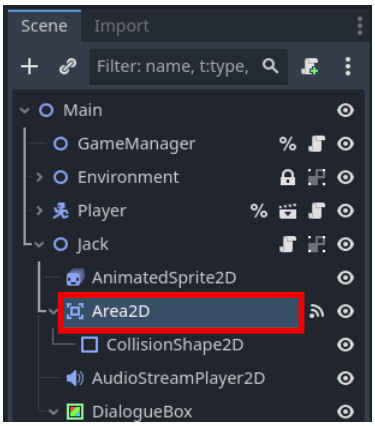
Check in with a Code Sensei before moving on. Ensure Jack asks the player to retrieve a random item on each playtest.

**Reminder:** Save your work!

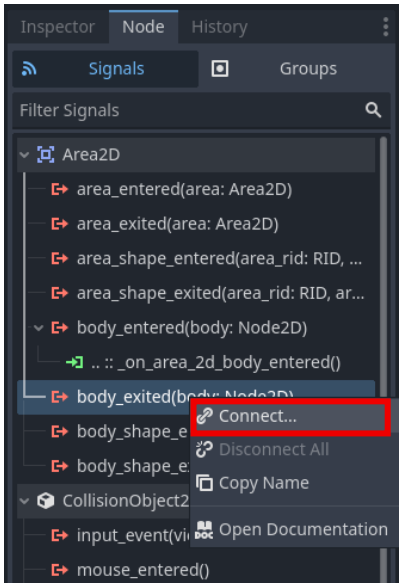
**36** It's important to only show the dialogue box when the player is conversing with Jack. Inside of `_on_area_2d_body_entered()`, locate **TODO 5**. Underneath, set the `visible` property of `dialogue_box` to true.

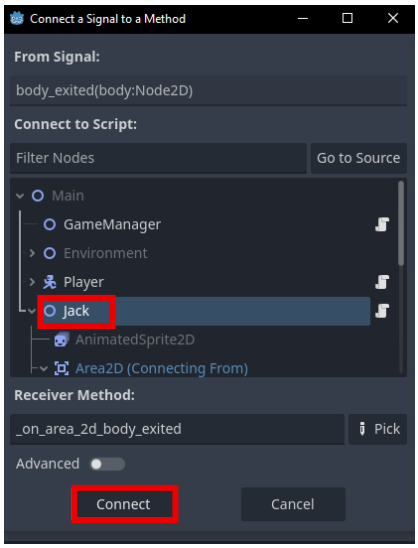
```
45    # --------------------
46    # TODO 5
47    # make dialogue_box visible
48    # --------------------
49    dialogue_box.visible = true
50
```

**37** The dialogue box should disappear when the player walks away from Jack. In **Scene**, select **Area2D** which is a child node of Jack.



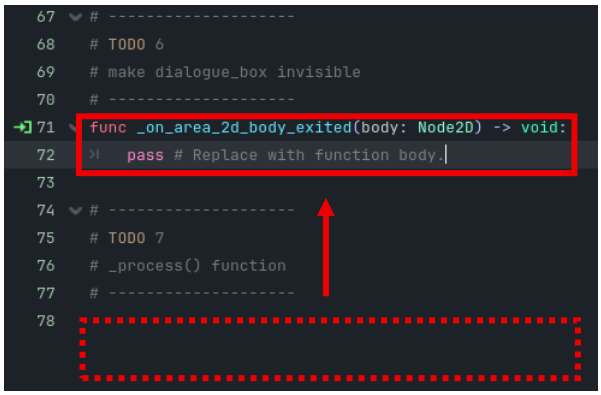

Toggle the Inspector panel to **Node**, and connect the **body_exited** signal to a new `_on_area_2d_body_exited()` receiver method on Jack.



## 38

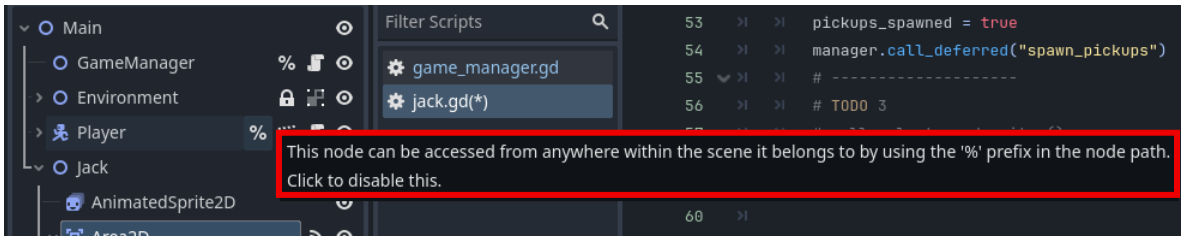
In the script editor, cut (**CTRL+X**) the `_on_area_2d_body_exited()` method located at the bottom of the script, and paste (**CTRL+V**) it underneath **TODO 6**.


## 39

The dialogue box should only be made invisible if the exiting body is the player.

Inside the new function, remove `pass`. Add an `if`-statement that checks if the `body` is equal to `%Player`. If true, set the `visible` property of `dialogue_box` to `false`.




```
67 # --------------------
68 # TODO 6
69 # make dialogue_box invisible
70 # --------------------
71 func _on_area_2d_body_exited(body: Node2D) -> void:
72 #     if ???
73 #         dialogue_box ???
74
```

**New Concept: Unique Name**

Nodes can be accessed from scripts by using **Unique Names** instead of `get_node()` or `get_first_node_in_group()`. Unique Names can be granted to nodes in the **Scene** panel and are accessed in code by typing `%[node name]`.

**40** Check the code! Update the script as needed.

```
67  # --------------------
68  # TODO 6
69  # make dialogue_box invisible
70  # --------------------
71  func _on_area_2d_body_exited(body: Node2D) -> void:
72      if body == %Player:
73          dialogue_box.visible = false
74
```

**41** Playtest the project. Does the dialogue box disappear when the player exits the Area2D and reappear when the player re-enters?

Don't worry if the dialogue box is visible when the game starts – this will be updated at the end of the project.

**42** Currently, Jack does not face the player during gameplay and always looks to the right. Code can be added to the `_process()` method that uses the `flip_h` property of AnimatedSprite2D.

In **jack.gd**, navigate to **TODO 7**. Define the `_process()` method with the `_delta` parameter of type `float` which returns `void`. Inside the method, set the `flip_h` property of `jack_sprite` to the boolean result of the conditional `player.global_position.x < global_position.x`.

```
75  # --------------------
76  # TODO 7
77  # _process() function
78  # --------------------
79  # _process ???
80  #     jack_sprite.flip_h ???
81
```

## 43

Check the code! Update the script as needed.

```
76  # --------------------
77  # TODO 7
78  # _process() function
79  # --------------------
80  func _process(_delta: float) -> void:
81      jack_sprite.flip_h = player.global_position.x < global_position.x
82
```

## 44

Each time the player brings an item back to Jack, Jack should determine if it is the desired item. However, this must only happen after the player first received the prompt to find the item. In other words, the pickups must have already been spawned.

Navigate to **TODO 8**. To ensure the pickups have already spawned, write an `elif`-statement that checks if `player.get_held_item()` is not equal to `desired_item_id`.

```
61      # --------------------
62      # TODO 8
63      # Compare held item
64      # --------------------
65      # elif ???
66
```

**Reminder:**

The `elif`-statement runs when the first `if`-statement fails. This is only the case when the class-level `pickups_spawned` variable is true.

**45** Inside the `elif`-statement, call the `write_dialogue()` function with the parameter:

**`"Hey! That's not my " + desired_item_id + "!"`**

Then, set the class-level `item_found` variable to `false`. This variable will help determine when the player completes the game in later steps.

```
61    # --------------------
62    # TODO 8
63    # Compare held item
64    # --------------------
65    # elif ???
66    #     write_dialogue ???
67    #     item_found ???
68
```

**46** Below, write an `else`-statement which will execute whenever the player's item matches Jack's desired item. Inside, call the `write_dialogue()` function with the parameter:

**`"Thanks! You found my " + desired_item_id + "!"`**

Then, set the class-level `item_found` variable to `true`.

```
61    # --------------------
62    # TODO 8
63    # Compare held item
64    # --------------------
65    # elif ???
66    #     write_dialogue ???
67    #     item_found ???
68    # else
69    #     write_dialogue ???
70    #     item_found ???
71
```

**47** Check the code! Update the script as needed.

```
61    # --------------------
62    # TODO 8
63    # Compare held item
64    # --------------------
65    elif player.get_held_item() != desired_item_id:
66        write_dialogue("Hey! That's not my " + desired_item_id + "!")
67        item_found = false
68    else:
69        write_dialogue("Thanks! You found my " + desired_item_id + "!")
70        item_found = true
71
```

**48** Playtest the project. Does Jack always turn to face the player? Does Jack tell the player whether they brought the desired item after the first dialogue?

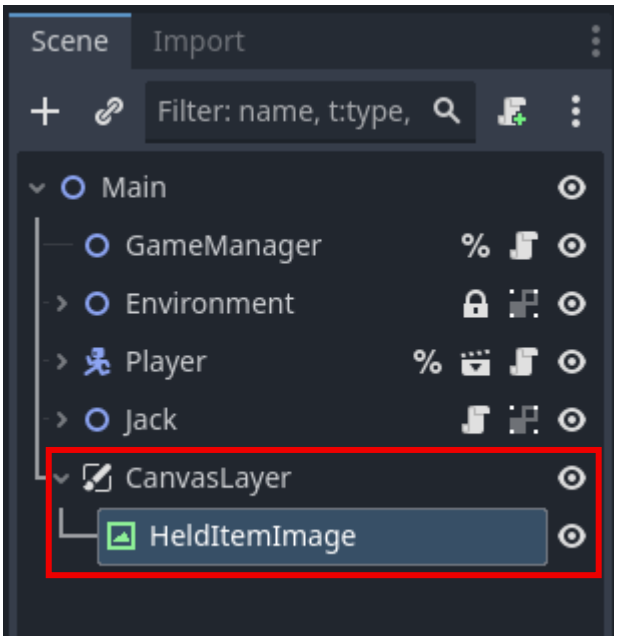

Pause for **Sensei Stop #3**!

Check in with a Code Sensei before moving on. Ensure the dialogue box disappears when the player leaves Jack, Jack always faces towards the player, and that Jack properly responds to the item that the player is holding.

**Reminder:** Save your work!

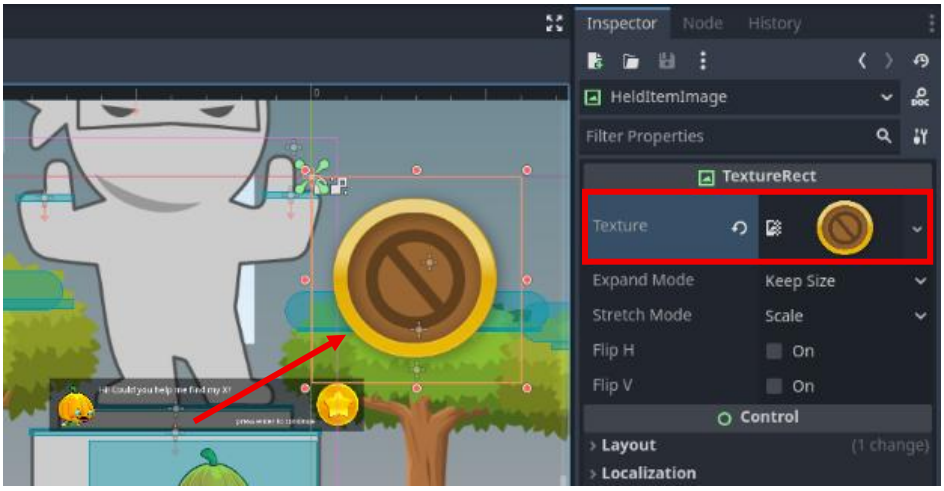
**49** Currently, the player may forget the current item they are holding, as there are no on-screen indicators that show the currently held item. This would make for a great UI element!

In **Scene**, add a **CanvasLayer** as a child node to **Main**. Add a **TextureRect** as a child node to **CanvasLayer** and rename it to **HeldItemImage**.

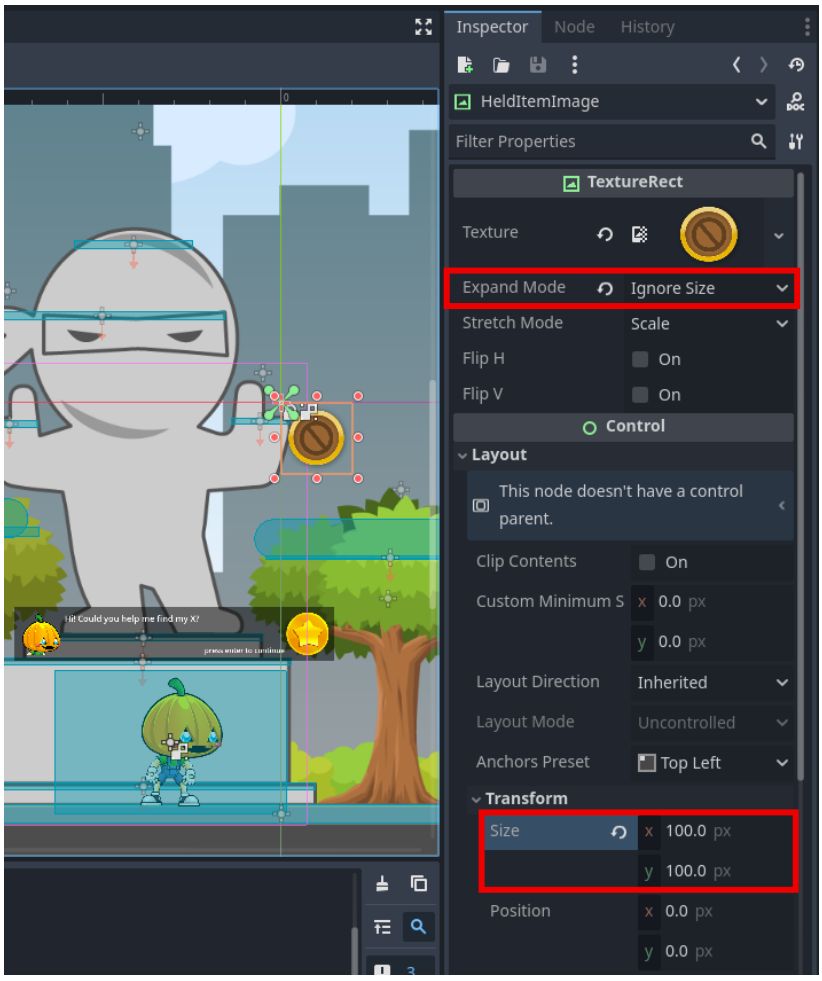


**50** In the **Inspector**, set the **Texture** to **holding.png** using **Quick Load**.

In the **2D** workspace, notice that the image is too big!

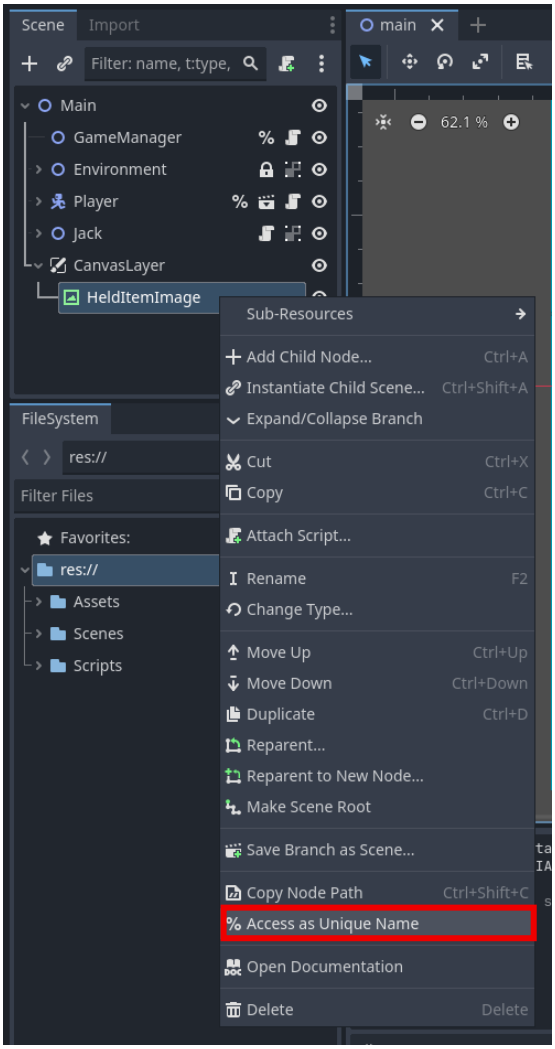


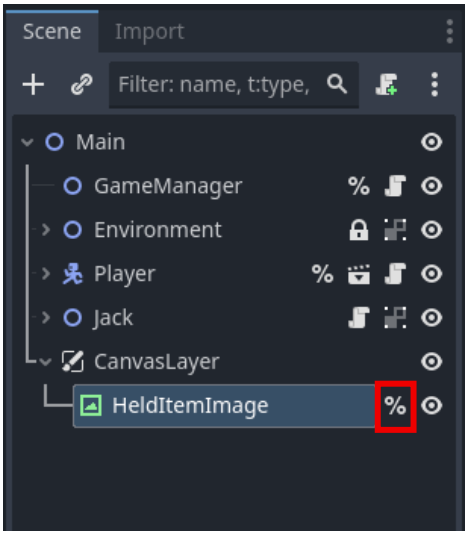

Set **Expand Mode** to **Ignore Size**, then open the **Layout** and **Transform** drop-down menus. Set the **Size** to 100px by 100px.



**51** To conveniently access the node from scripts, give it a Unique Name.

In **Scene**, right-click **HeldItemImage** and select **% Access as Unique Name**.



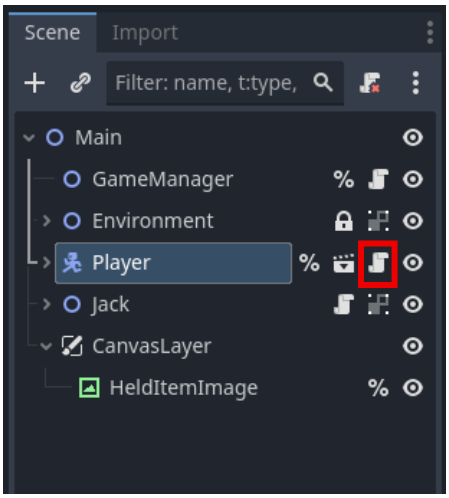




**Reminder:**

Unique Names can be granted to nodes in the **Scene** panel and are accessed in code by typing **%[node name]**.

**52** The functionality of showing the currently held item will be added to the Player's script.

Open the **player.gd** script which is attached to **Player** and navigate to **TODO 9**.




```gdscript
class_name Player
extends CharacterBody2D

# Movement vars
@export var speed: float = 300.0
@export var jump_velocity: float = -1000.0
@export var gravity_acceleration: float = 3000.0

# Scene references
@onready var manager: GameManager = %GameManager
# --------------------
# TODO 9
# Reference held item image
# --------------------

var _held_item_id: String = ""

func _physics_process(delta: float) -> void:
    # If we are in the air, apply gravity
    if not is_on_floor():
        velocity += Vector2.DOWN * gravity_acceleration * delta

    #If jump button pressed and we are on the ground
    if Input.is_action_just_pressed("ui_select") and is_on_floor():
        velocity.y = jump_velocity

```

**53** Under **TODO 9**, reference **HeldItemImage** by dragging it in and holding CTRL before releasing.

```gdscript
# --------------------
# TODO 9
# Reference held item image
# --------------------
@onready var held_item_image: TextureRect = %HeldItemImage

```

**54** The items have their own script, **pickup.gd**, which calls the player's `set_held_item()` function after being collected. The only thing that needs to be done is to set the texture of the held item. `_held_item_id` is a class-level variable in player.gd which can help with this.

Navigate to **TODO 10**. Below, set the `texture` property of `held_item_image` equal to the Texture2D retrieved from accessing `manager.texture_dict[id]`.

```
37  func get_held_item():
38      return _held_item_id
39
40  # Set the held item to the given id and update UI with the correct texture
41  func set_held_item(id: String):
42      _held_item_id = id
43      # --------------------
44      # TODO 10
45      # Set held item image
46      # --------------------
47      # held_item_image.texture ???
48
```

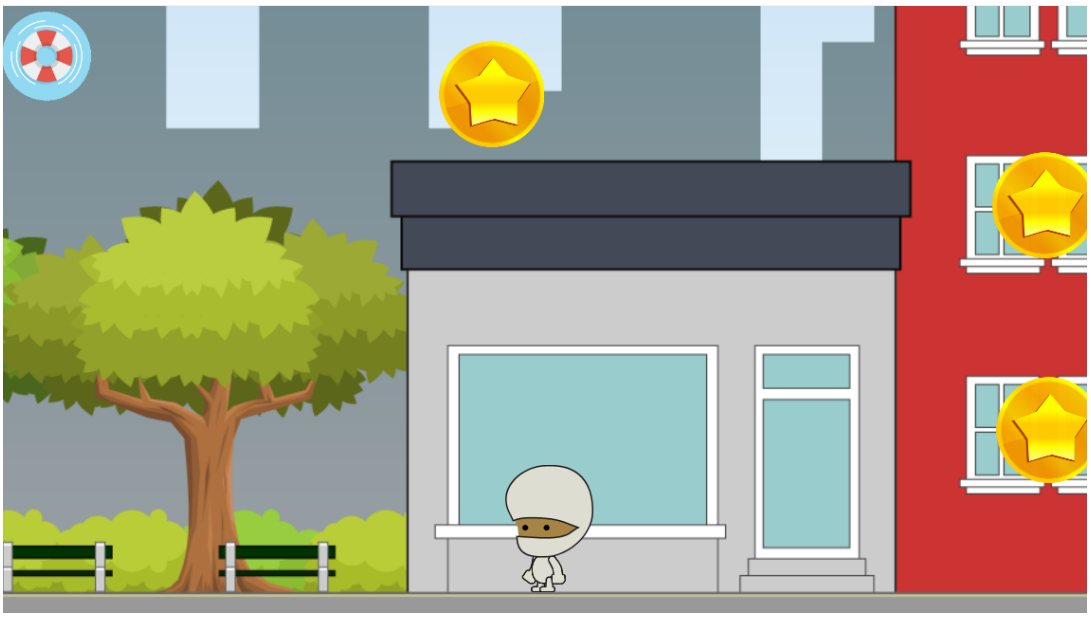
**Reminder:**

`texture_dict` is an array in **game_manager.gd** that maps the `String` id of an item to its `Texture2D`.

**55** Check the code! Update the script as needed.

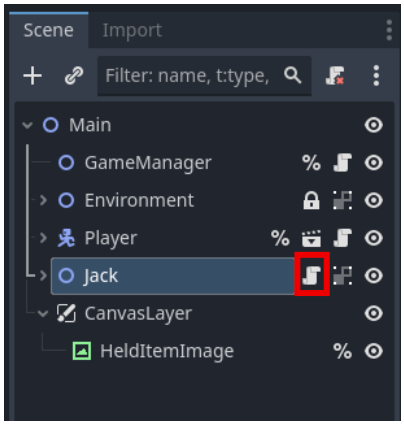
```
43      # --------------------
44      # TODO 10
45      # Set held item image
46      # --------------------
47      held_item_image.texture = manager.texture_dict[id]
48
```

**56** Playtest the project. Does the currently held item appear in the top-left corner?



**57** The game is nearly complete! The last thing to do is to allow the player to close the dialogue box and reset the game when the player gives the desired item to Jack.

Open the **jack.gd** script which is attached to **Jack** and navigate to **TODO 7**.




```
72  # --------------------
73  # TODO 6
74  # make dialogue_box invisible
75  # --------------------
76  func _on_area_2d_body_exited(body: Node2D) -> void:
77      if body == %Player:
78          dialogue_box.visible = false
79
80  # --------------------
81  # TODO 7
82  # _process() function
83  # --------------------
84  func _process(_delta: float) -> void:
85      jack_sprite.flip_h = player.global_position.x < global_position.x
86
```

**58** Inside the `_process()` method, write a new `if`-statement that checks if the `"ui_accept"` action was just pressed `and` the `dialogue_box` is visible. Inside the `if`-statement, set the `visible` property of `dialogue_box` to `false`.

```
80  # --------------------
81  # TODO 7
82  # _process() function
83  # --------------------
84  func _process(_delta: float) -> void:
85      jack_sprite.flip_h = player.global_position.x < global_position.x
86
87      # if ???
88      #     dialogue_box.visible ???
89
```

**59** Inside of the `if`-statement, write another `if`-statement that checks the boolean value of the class-level `item_found` variable. If true, reload the current scene using `get_tree()`.

```
80  # --------------------
81  # TODO 7
82  # _process() function
83  # --------------------
84  func _process(_delta: float) -> void:
85      jack_sprite.flip_h = player.global_position.x < global_position.x
86
87      # if ???
88      #     dialogue_box.visible ???
89      #     if ???
90      #         get_tree ???
91
```

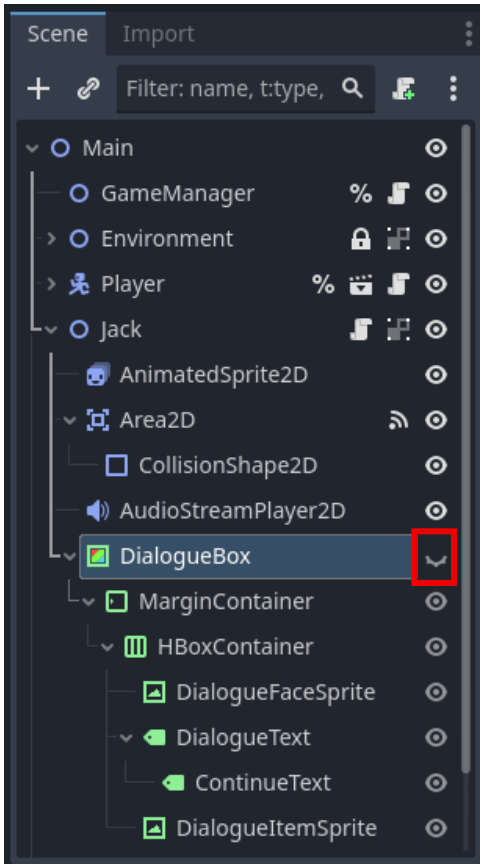
**Reminder:**

`item_found` is set each time Player starts conversing with Jack. It represents whether the Player's held item matches Jack's desired item.

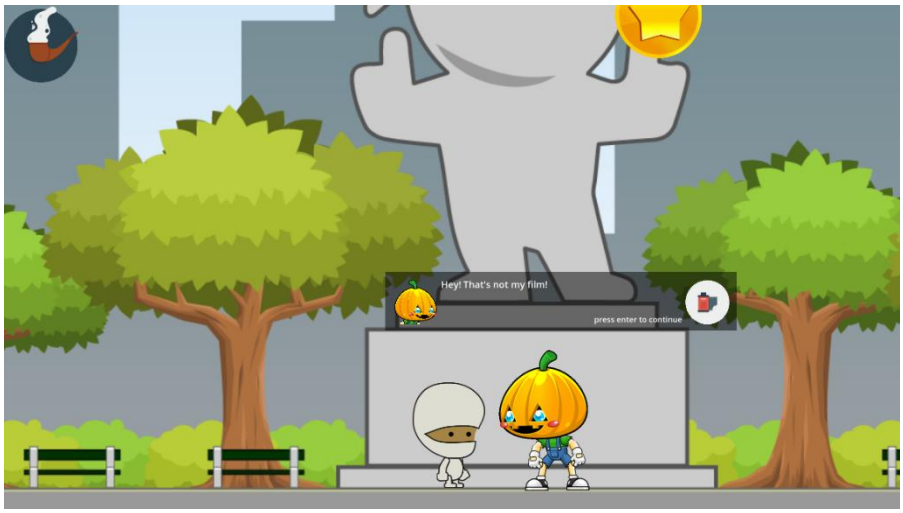
**60** Check the code! Update the script as needed.

```
# --------------------
# TODO 7
# _process() function
# --------------------
func _process(_delta: float) -> void:
    jack_sprite.flip_h = player.global_position.x < global_position.x

    if Input.is_action_just_pressed("ui_accept") and dialogue_box.visible:
        dialogue_box.visible = false
        if item_found:
            get_tree().reload_current_scene()

```

**61** Now that the implementation is finished, set **DialogueBox** to be invisible by default in **Scene**.



**62** Playtest the project! Is everything working as intended?





Pause for **Sensei Stop #4**!

Congratulations on upgrading the Scavenger Hunt project in Godot! Great job!

Before submitting, check in with a Code Sensei to check that the currently held item is shown in the corner and the player can control the dialogue box then reflect on the following:

- What did you learn about UI layouts, dialogue systems, and Unique Names?
- What did you enjoy most when creating this project?
- What was something you found difficult and why?

**Reminder**: Save your work!