Criterion C: Development

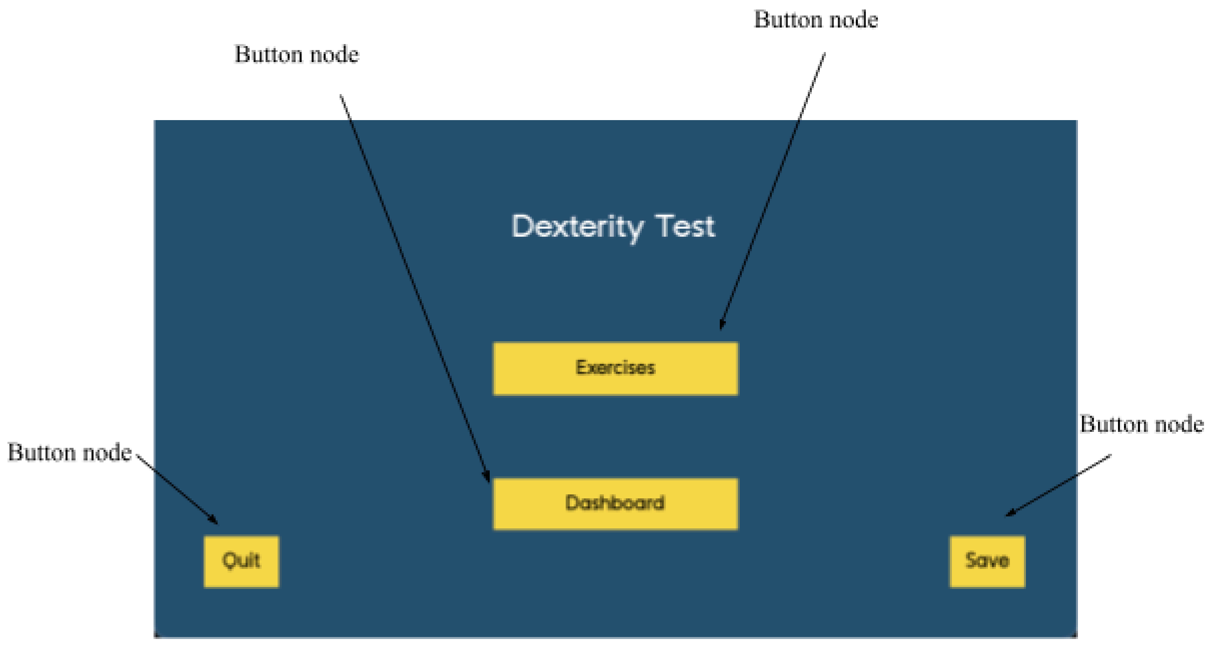
Techniques Used:

- Converting arrays to strings
- Saving/ loading
- Arrays
- Node trees
- File Management
- UI
- Signals
- Timers
- Random number generation
- For loops
- Functions
- Global script

All code is found in **Appendix 6**.

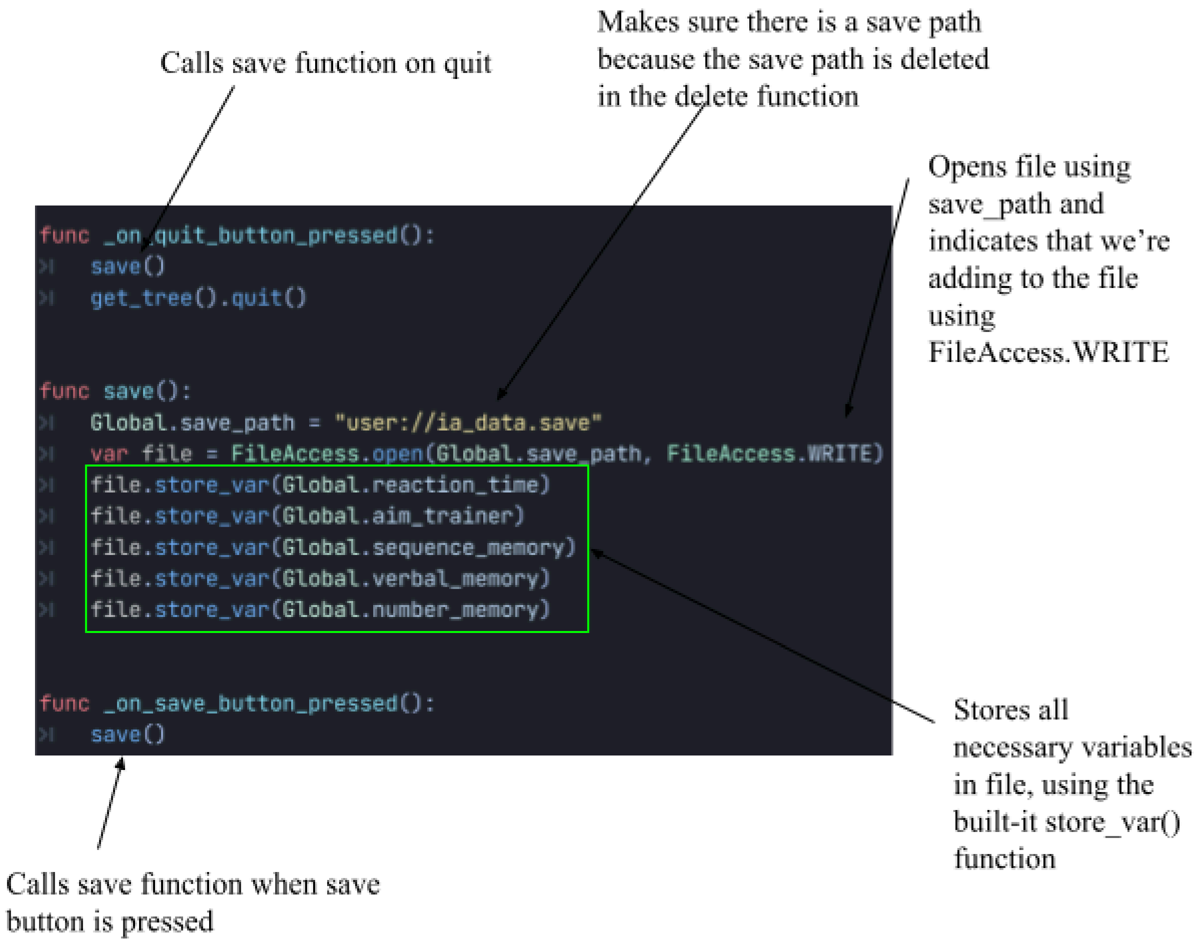
Structure:

The way you code in Godot is to use nodes, there are hundreds of nodes you could choose from, but this project will use a select few. These nodes can be organized into trees, with parent-child relationships with the parent nodes affecting the child nodes, and information received from the child nodes can be sent to the parent node and processed by the parent node. These trees can be organized into different scenes, which can also be instantiated into other scenes.



When the exercises button is pressed the user will be redirected to the exercises scene. When the user presses the dashboard button the user will be redirected to the dashboard scene. When the save button is pressed the save function will be called. When the quit button is pressed the save function will also be called then the app will end.

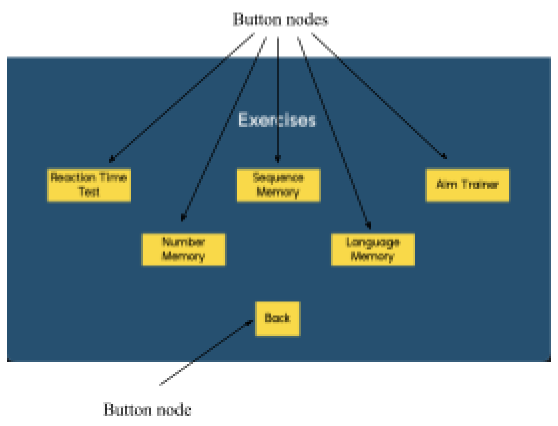
Screenshot #1: *Main menu*


Calls save function on quit
Makes sure there is a save path because the save path is deleted in the delete function
Opens file using save_path and indicates that we're adding to the file using FileAccess.WRITE
Stores all necessary variables in file, using the built-it store_var() function
Calls save function when save button is pressed

```
func _on_quit_button_pressed():
	save()
	get_tree().quit()


func save():
	Global.save_path = "user://ia_data.save"
	var file = FileAccess.open(Global.save_path, FileAccess.WRITE)
	file.store_var(Global.reaction_time)
	file.store_var(Global.aim_trainer)
	file.store_var(Global.sequence_memory)
	file.store_var(Global.verbal_memory)
	file.store_var(Global.number_memory)


func _on_save_button_pressed():
	save()
```

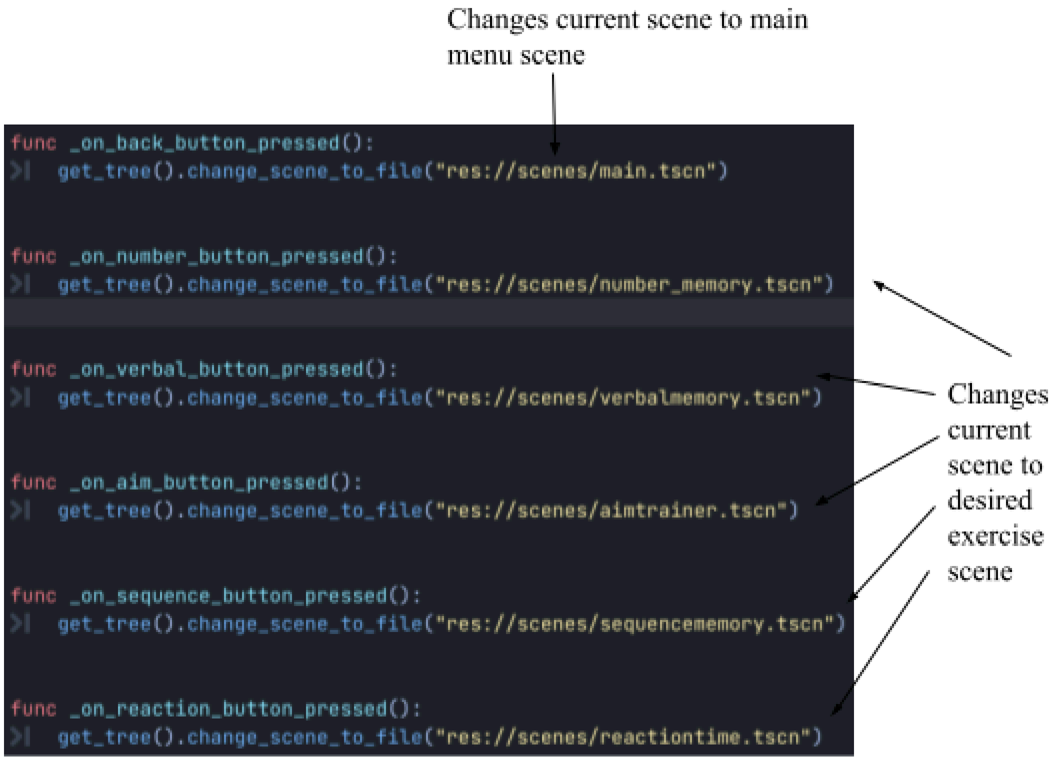
Screenshot #2: *Save function*

Screenshot #2 shows how the save function works, and it works by creating a file on the user's computer and adding the data to that file using the built-in store_var() function.



Each button node (other than the back button) will navigate the user to their desired exercise. The back button node will navigate the user to the main menu scene. This is where the user can pick between all exercises which will test the user's reaction time, hand-eye coordination, and memory. The only code being run on this menu is to switch scenes to the correct scene depending on which button is being pressed. All exercises also have a basic menu page describing what the exercise is. These menus won't be shown as there isn't any relevant code being run in those menus (other than changing scenes), and the UI of all the menus is consistent.

Screenshot #3: *Exercises Menu*

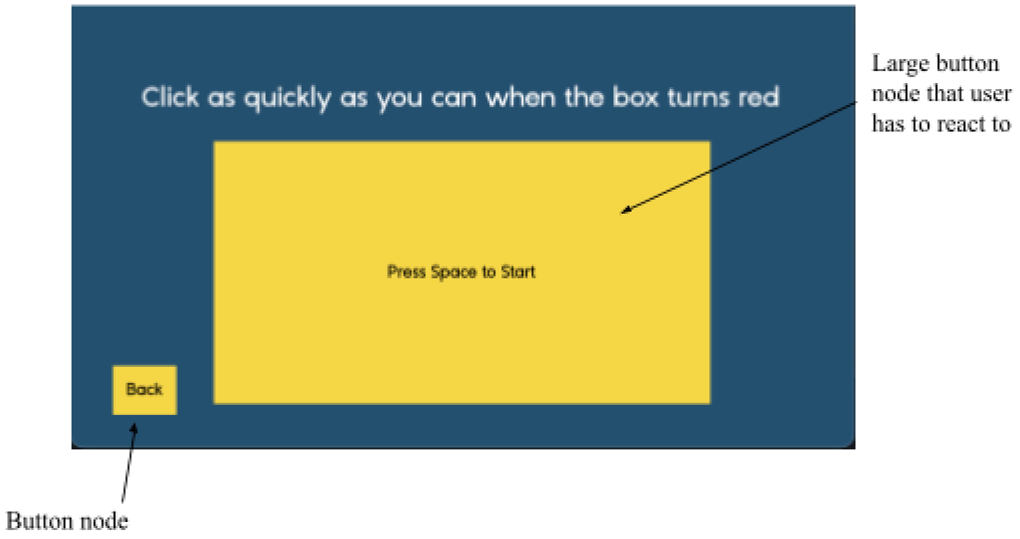

Changes current scene to main menu scene
Changes current scene to desired exercise scene

```
func _on_back_button_pressed():
	get_tree().change_scene_to_file("res://scenes/main.tscn")


func _on_number_button_pressed():
	get_tree().change_scene_to_file("res://scenes/number_memory.tscn")


func _on_verbal_button_pressed():
	get_tree().change_scene_to_file("res://scenes/verbalmemory.tscn")


func _on_aim_button_pressed():
	get_tree().change_scene_to_file("res://scenes/aimtrainer.tscn")


func _on_sequence_button_pressed():
	get_tree().change_scene_to_file("res://scenes/sequencememory.tscn")


func _on_reaction_button_pressed():
	get_tree().change_scene_to_file("res://scenes/reactiontime.tscn")
```

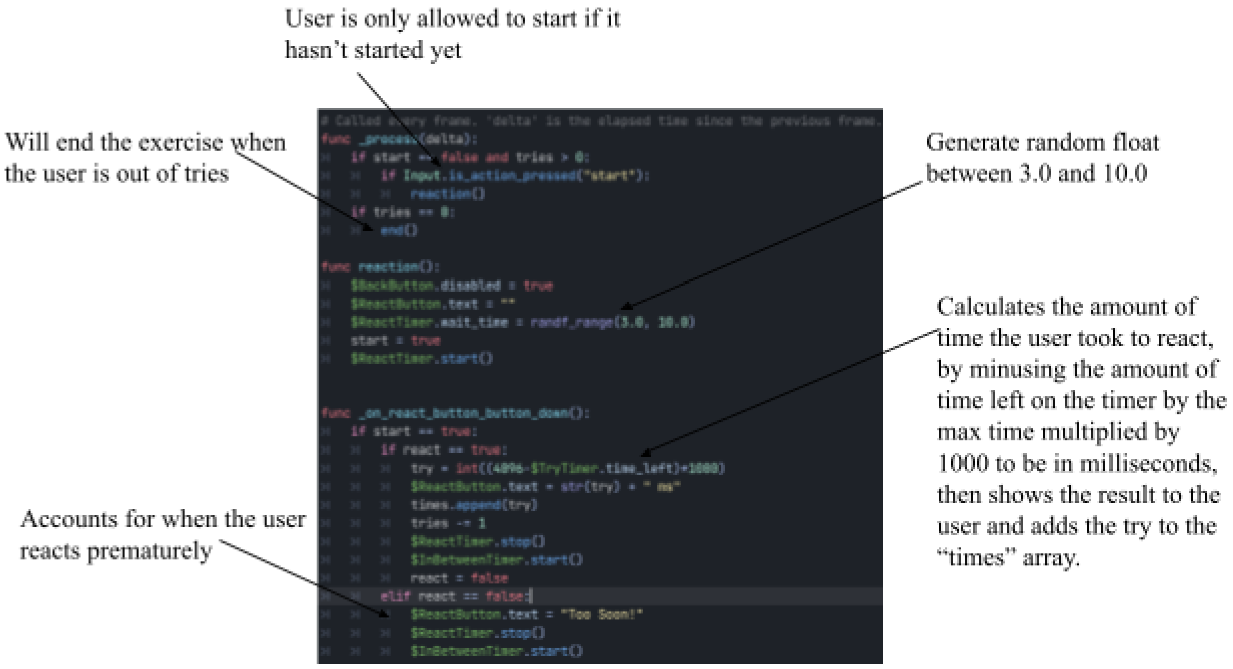
Screenshot #4: *Exercises Menu Navigation System*

Screenshot #4 shows the code behind changing scenes, and it's pretty simple, just use the change_scene_to_file() function then put the path to the desired scene as the parameter.

The user has to press space to begin the exercise. Once the exercise begins, a random float between 3.0 and 10.0 will be generated to determine the amount of time the user has to wait for the large button node to turn red. The user is given 5 attempts and then an average of all the times will be determined and presented to the user. While the exercise is underway the back button will be disabled(this is the same for all the tests). Before or after the exercise if the back button is pressed it will navigate the user to the reaction time exercise menu scene.


Click as quickly as you can when the box turns red
Press Space to Start
Back
Large button node that user has to react to
Button node

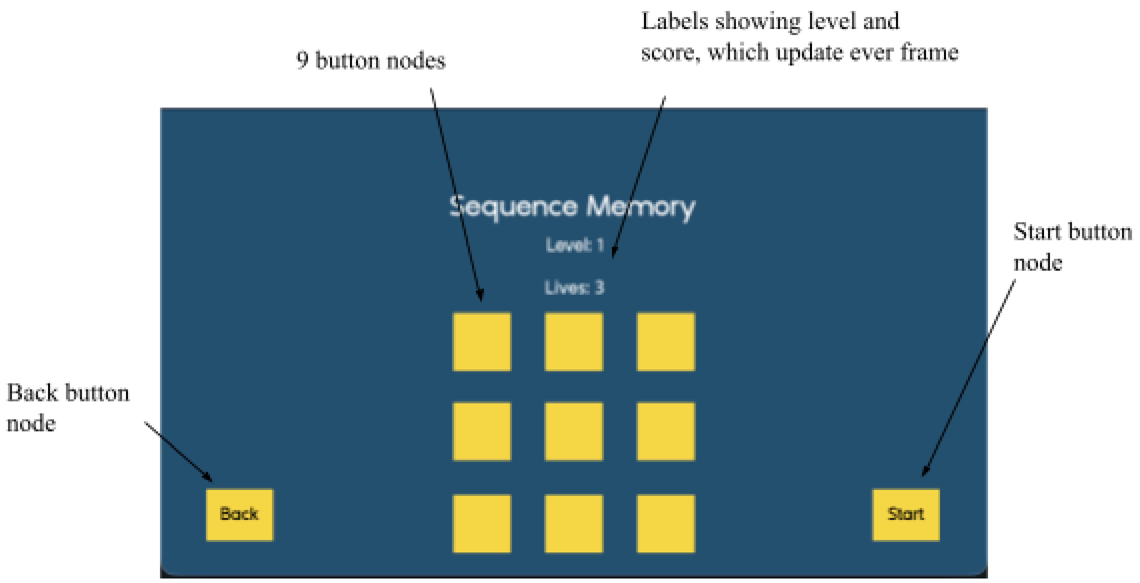
Screenshot #5: *Reaction Time Test*


User is only allowed to start if it hasn't started yet
Will end the exercise when the user is out of tries
Generate random float between 3.0 and 10.0
Calculates the amount of time the user took to react, by minusing the amount of time left on the timer by the max time multiplied by 1000 to be in milliseconds, then shows the result to the user and adds the try to the "times" array.
Accounts for when the user reacts prematurely

```
# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(delta):
	if start == false and tries > 0:
		if Input.is_action_pressed("start"):
			reaction()
	if tries == 0:
		end()

func reaction():
	$BackButton.disabled = true
	$ReactButton.text = ""
	$ReactTimer.wait_time = randf_range(3.0, 10.0)
	start = true
	$ReactTimer.start()


func _on_react_button_button_down():
	if start == true:
		if react == true:
			try = int((4096-$TryTimer.time_left)*1000)
			$ReactButton.text = str(try) + " ms"
			times.append(try)
			tries -= 1
			$ReactTimer.stop()
			$InBetweenTimer.start()
			react = false
		elif react == false:
			$ReactButton.text = "Too Soon!"
			$ReactTimer.stop()
			$InBetweenTimer.start()
```

Screenshot #6: *Reaction Button Logic*

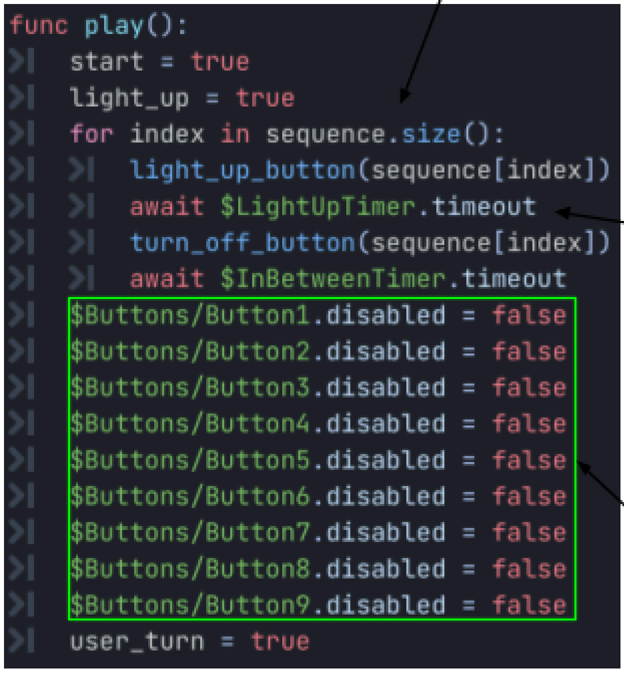
Screenshot #6 shows some of the code used in the reaction time test. The amount of time the user has to wait changed from 5.0s - 15.0s to 3.0s - 10.0s as sometimes the user had to wait too long.



The 9-button nodes will turn red in a random order, one at a time. When the sequence is over the user has to press the buttons in the order they were lighted up. The length of the sequence will increase with the level. So if the level is 3 then the length of the sequence will be 3. The start button node will disappear when pressed, and will also disable the back button node. The back button node will be enabled when the exercise is over and when pressed will navigate the user back to the sequence memory menu scene.

Screenshot #7: *Sequence Memory Test*

Utilizes a for loop so that it can light up each button in order



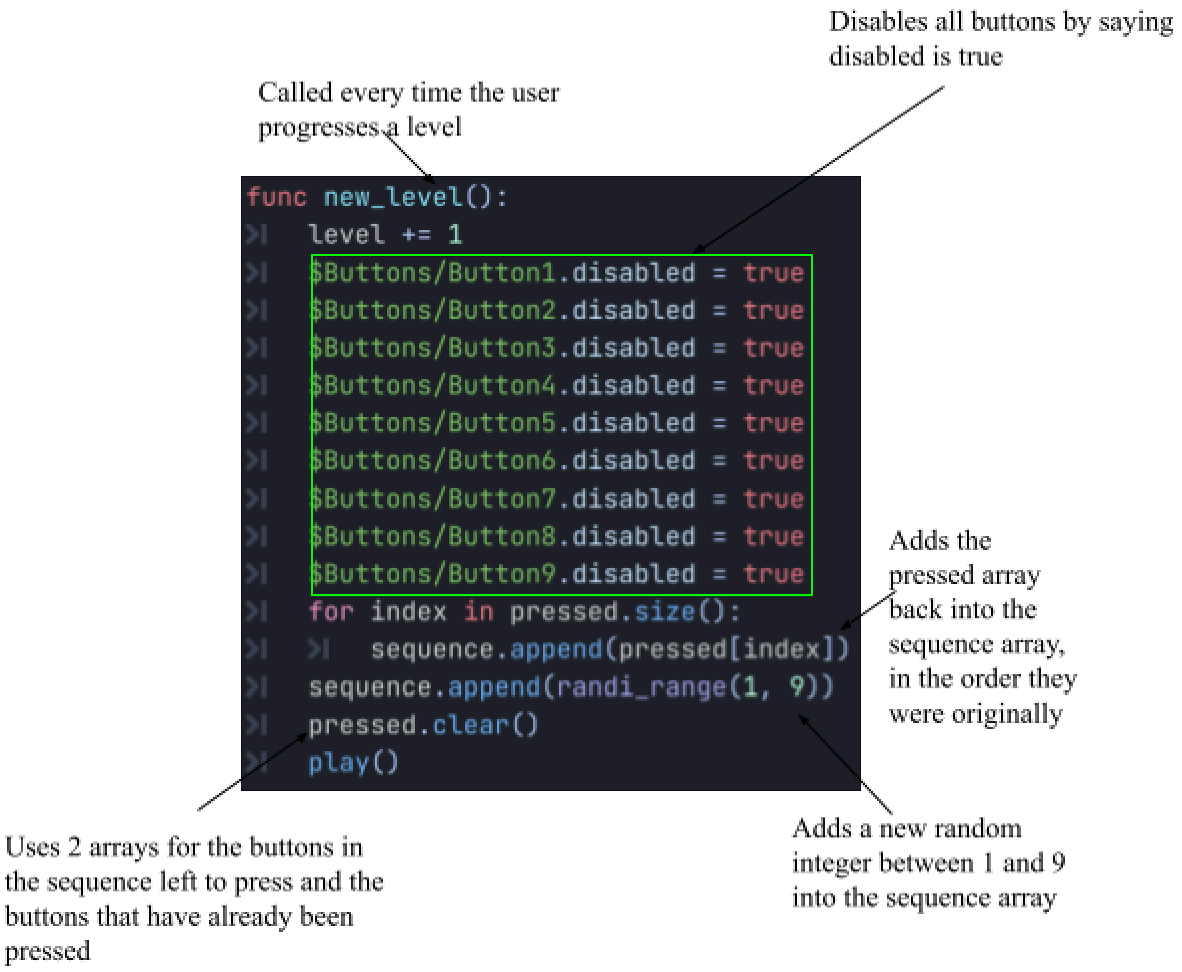
```
func play():
	start = true
	light_up = true
	for index in sequence.size():
		light_up_button(sequence[index])
		await $LightUpTimer.timeout
		turn_off_button(sequence[index])
		await $InBetweenTimer.timeout
	$Buttons/Button1.disabled = false
	$Buttons/Button2.disabled = false
	$Buttons/Button3.disabled = false
	$Buttons/Button4.disabled = false
	$Buttons/Button5.disabled = false
	$Buttons/Button6.disabled = false
	$Buttons/Button7.disabled = false
	$Buttons/Button8.disabled = false
	$Buttons/Button9.disabled = false
	user_turn = true
```

Lights up the next button in the sequence by calling the light_up_button(i) function, then waits for the timer to go off before setting the button's color back to the original color, using the turn_off_button(i) function.

Enables all buttons, by saying disabled is false

Screenshot #8: *Sequence Memory Lighting Up Buttons Logic*

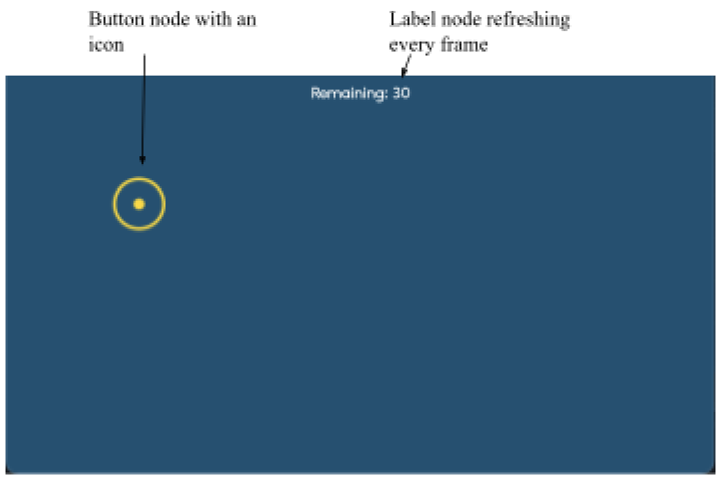
Screenshot #8 shows the play() that uses a for loop to iterate through the sequence array and light up each button in sequence then allows the user to click on the buttons.


Called every time the user progresses a level
Disables all buttons by saying disabled is true
Adds the pressed array back into the sequence array, in the order they were originally
Uses 2 arrays for the buttons in the sequence left to press and the buttons that have already been pressed
Adds a new random integer between 1 and 9 into the sequence array

```
func new_level():
	level += 1
	$Buttons/Button1.disabled = true
	$Buttons/Button2.disabled = true
	$Buttons/Button3.disabled = true
	$Buttons/Button4.disabled = true
	$Buttons/Button5.disabled = true
	$Buttons/Button6.disabled = true
	$Buttons/Button7.disabled = true
	$Buttons/Button8.disabled = true
	$Buttons/Button9.disabled = true
	for index in pressed.size():
		sequence.append(pressed[index])
	sequence.append(randi_range(1, 9))
	pressed.clear()
	play()
```

Screenshot #9: *Sequence Memory New Level Logic*

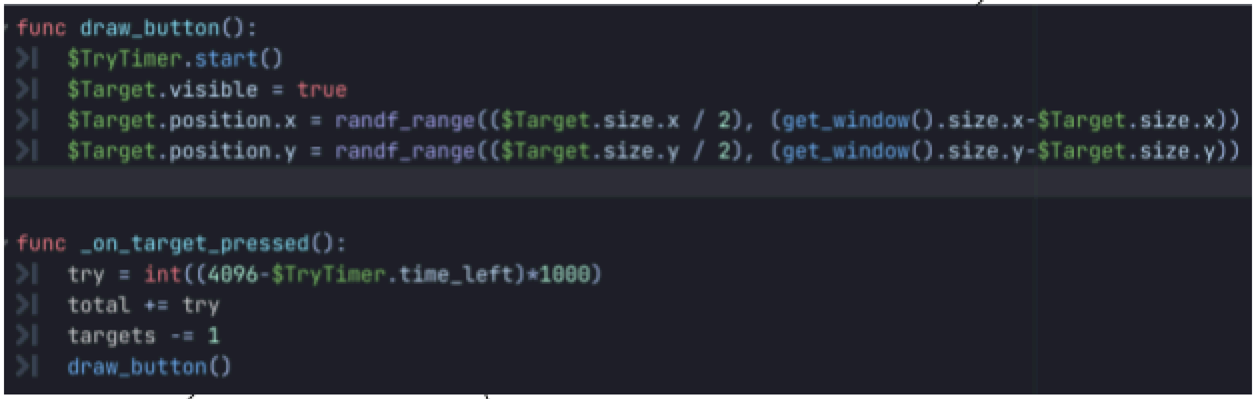
Screenshot #9 shows the new_level() function that disables all buttons, then recreates the sequence array using a for loop and empties the pressed array.



The button node's position will be randomly generated within the window of the application, using the code shown below. The label node shows the user the remaining amount of targets they have to click before they're done, refreshing every frame. When the exercise is completed a back button will appear that will navigate the user back to the aim trainer menu scene when pressed.

Screenshot #10: *Aim Trainer Test UI*

Sets the target's x position to a random place in between half the target's x size to the window's x size minus the targets x size and same thing for the y position



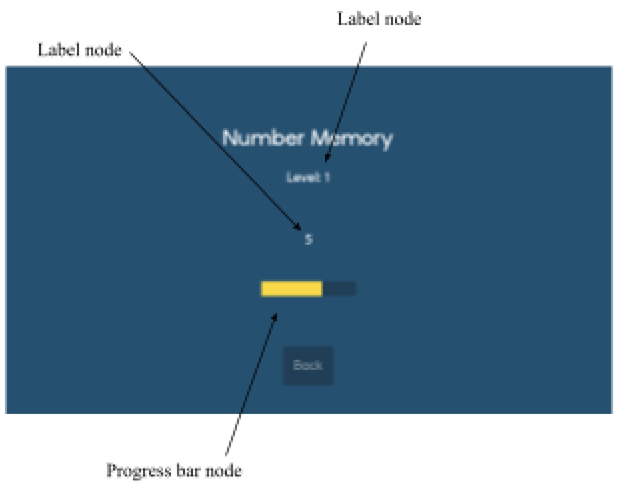
```
func draw_button():
	$TryTimer.start()
	$Target.visible = true
	$Target.position.x = randf_range(($Target.size.x / 2), (get_window().size.x-$Target.size.x))
	$Target.position.y = randf_range(($Target.size.y / 2), (get_window().size.y-$Target.size.y))


func _on_target_pressed():
	try = int((4096-$TryTimer.time_left)*1000)
	total += try
	targets -= 1
	draw_button()
```

Redraws the target in a new position, will repeat 30 times, then calculate an average of the tries

Calculates the amount of time it took for the user to press the target by minussing the max amount of time left by the actual time left to get the amount of time it took for the user to press multiplied by 1000 to be in milliseconds (same as math from reaction time test)

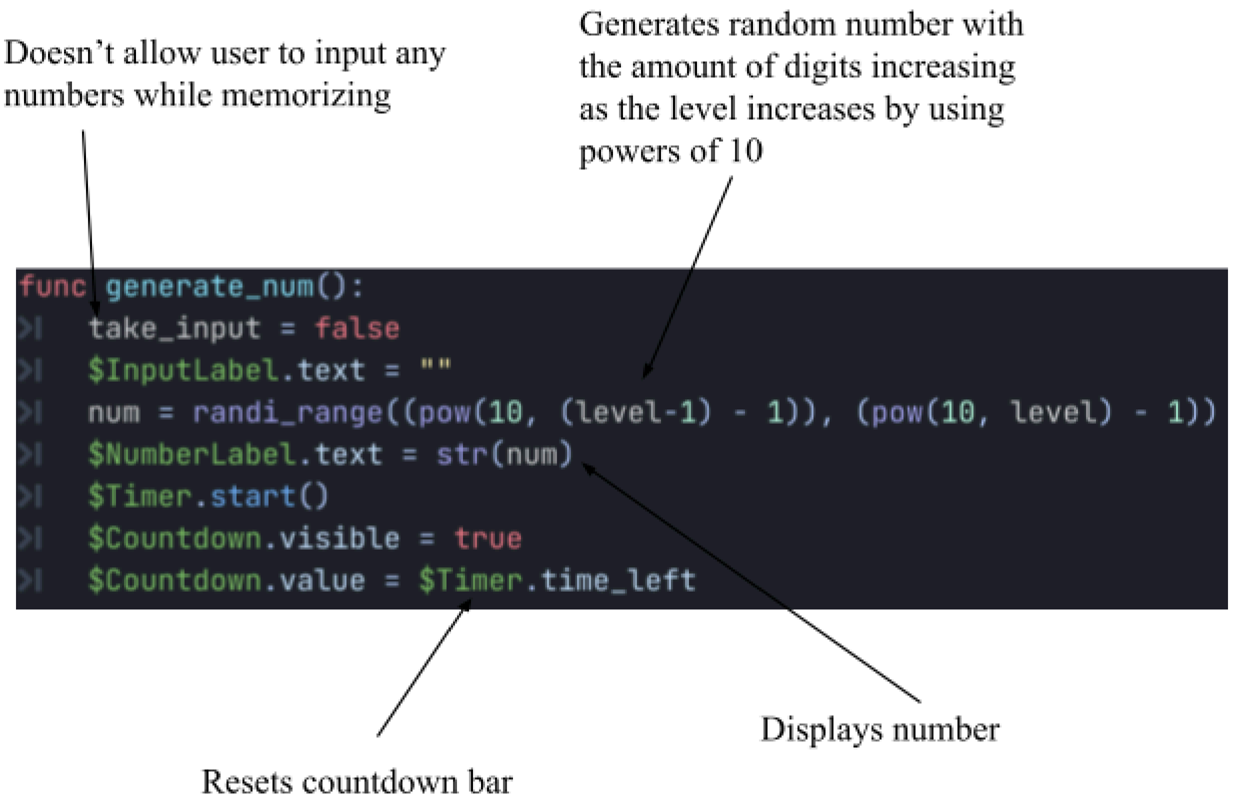
Screenshot #11: *Aim Trainer Target Logic*

Screenshot #11 shows the functions draw_button() and _on_target_pressed(). _on_target_pressed() is always called when the target button is pressed, which then calculates the amount of time taken for the button to be pressed. The draw_button() function just draws the target to a new random position on the screen.



A random number is generated based on the level, each level corresponding to the number of digits. A bar is shown below the number for a 3-second countdown so that the user knows how much time they have left to memorize the number. When the exercise is completed the back button will enable and when pressed will navigate the user to the number memory menu scene.

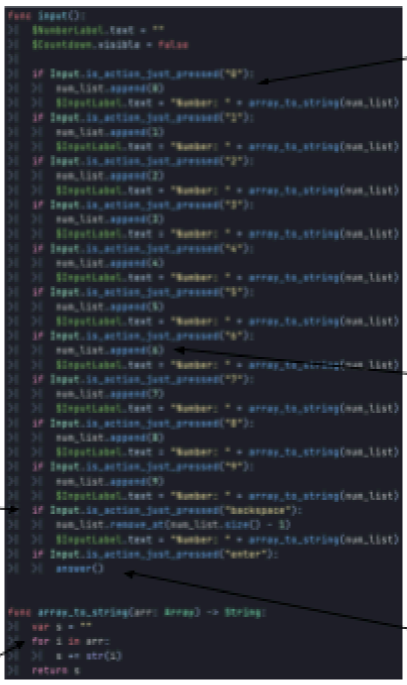
Screenshot #12: *Number Memory Test*


Doesn't allow user to input any numbers while memorizing
Generates random number with the amount of digits increasing as the level increases by using powers of 10
Displays number
Resets countdown bar

```
func generate_num():
    take_input = false
    $InputLabel.text = ""
    num = randi_range((pow(10, (level-1) - 1)), (pow(10, level) - 1))
    $NumberLabel.text = str(num)
    $Timer.start()
    $Countdown.visible = true
    $Countdown.value = $Timer.time_left
```

Screenshot #13: *Number Memory Test Number Generation*

Screenshot #13 shows the generate_num() function that generates a random number with the amount of digits increasing relative to the level the user is on.



```
func input():
	$NumberLabel.text = ""
	$Countdown.visible = false
	
	if Input.is_action_just_pressed("0"):
		num_list.append(0)
		$InputLabel.text = "Number: " + array_to_string(num_list)
	if Input.is_action_just_pressed("1"):
		num_list.append(1)
		$InputLabel.text = "Number: " + array_to_string(num_list)
	if Input.is_action_just_pressed("2"):
		num_list.append(2)
		$InputLabel.text = "Number: " + array_to_string(num_list)
	if Input.is_action_just_pressed("3"):
		num_list.append(3)
		$InputLabel.text = "Number: " + array_to_string(num_list)
	if Input.is_action_just_pressed("4"):
		num_list.append(4)
		$InputLabel.text = "Number: " + array_to_string(num_list)
	if Input.is_action_just_pressed("5"):
		num_list.append(5)
		$InputLabel.text = "Number: " + array_to_string(num_list)
	if Input.is_action_just_pressed("6"):
		num_list.append(6)
		$InputLabel.text = "Number: " + array_to_string(num_list)
	if Input.is_action_just_pressed("7"):
		num_list.append(7)
		$InputLabel.text = "Number: " + array_to_string(num_list)
	if Input.is_action_just_pressed("8"):
		num_list.append(8)
		$InputLabel.text = "Number: " + array_to_string(num_list)
	if Input.is_action_just_pressed("9"):
		num_list.append(9)
		$InputLabel.text = "Number: " + array_to_string(num_list)
	if Input.is_action_just_pressed("backspace"):
		num_list.remove_at(num_list.size() - 1)
		$InputLabel.text = "Number: " + array_to_string(num_list)
	if Input.is_action_just_pressed("enter"):
		answer()


func array_to_string(arr: Array) -> String:
	var s = ""
	for i in arr:
		s += str(i)
	return s
```

Uses is_action_just_pressed() so that the user isn't inputting every frame

Adds each inputted number to the array

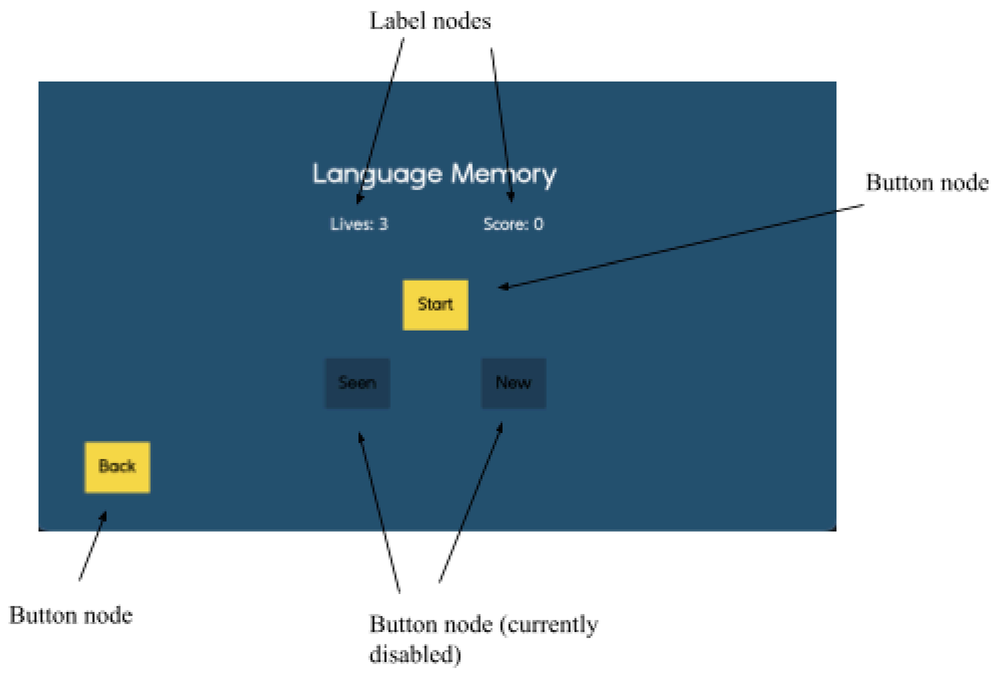
Deletes most recently inputted number into the array, same function backspace would normally have

When user enters their answer, their answer will be checked

Creates an empty string and adds each number from the array to it in the correct order, then returns the string

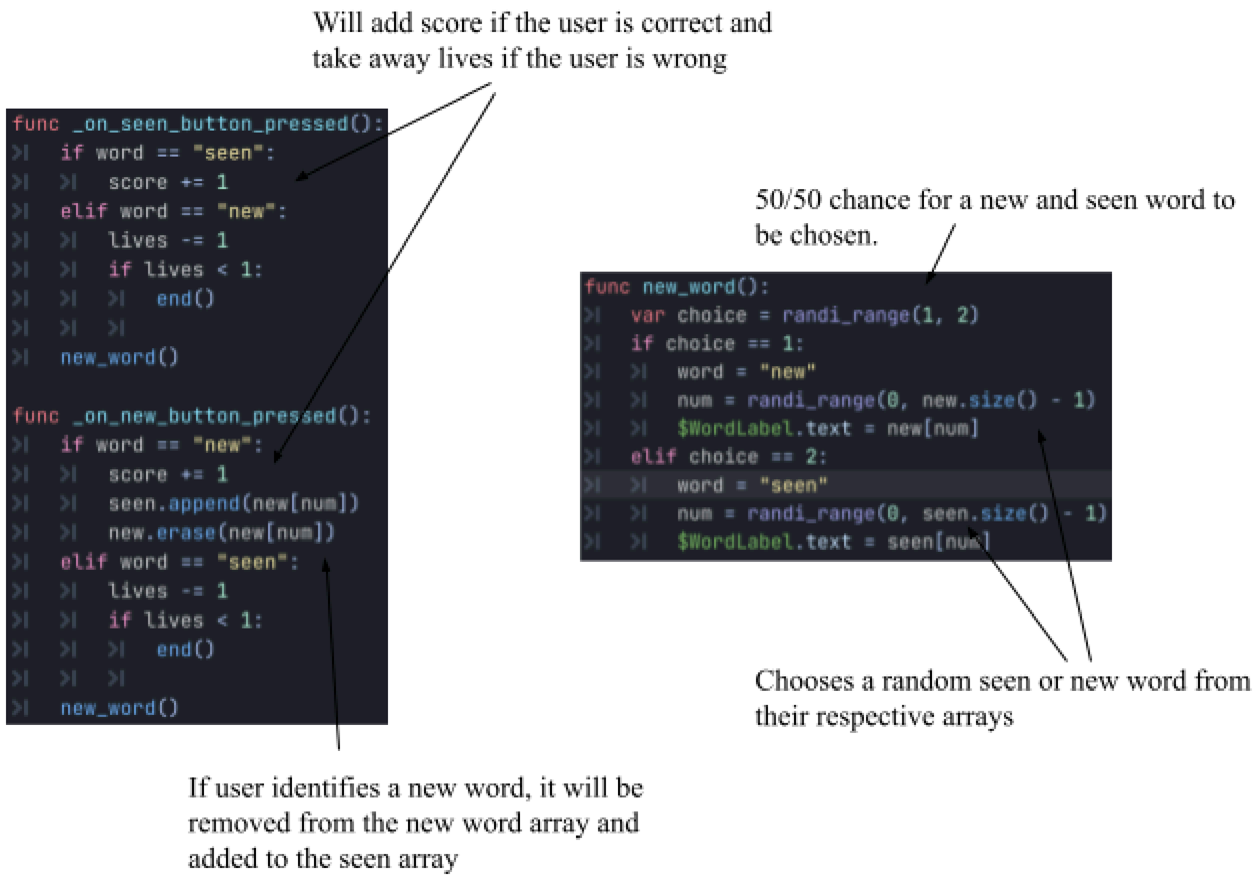
Screenshot #14: *Number Memory Test Input and Array to String Function*

Screenshot #14 shows the input() function and the array_to_string() function. All the input() function does is wait for input from the user and only allow the user to input numbers or the backspace or enter buttons. The array_to_string() function just converts any given array into a string by iterating through the array and adding each index to a string.



When the start button node is pressed, it will disappear and the back button node will disable then the user will be shown a random word and must remember whether or not they've seen the word. After 3 mistakes the exercise will end and the user will be shown their score. The back button node will then be enabled and when pressed will send the user back to the language memory menu scene.

Screenshot #15: *Language Memory Test*



Will add score if the user is correct and take away lives if the user is wrong

```
func _on_seen_button_pressed():
	if word == "seen":
		score += 1
	elif word == "new":
		lives -= 1
		if lives < 1:
			end()
			
	new_word()

func _on_new_button_pressed():
	if word == "new":
		score += 1
		seen.append(new[num])
		new.erase(new[num])
	elif word == "seen":
		lives -= 1
		if lives < 1:
			end()
			
	new_word()
```

If user identifies a new word, it will be removed from the new word array and added to the seen array

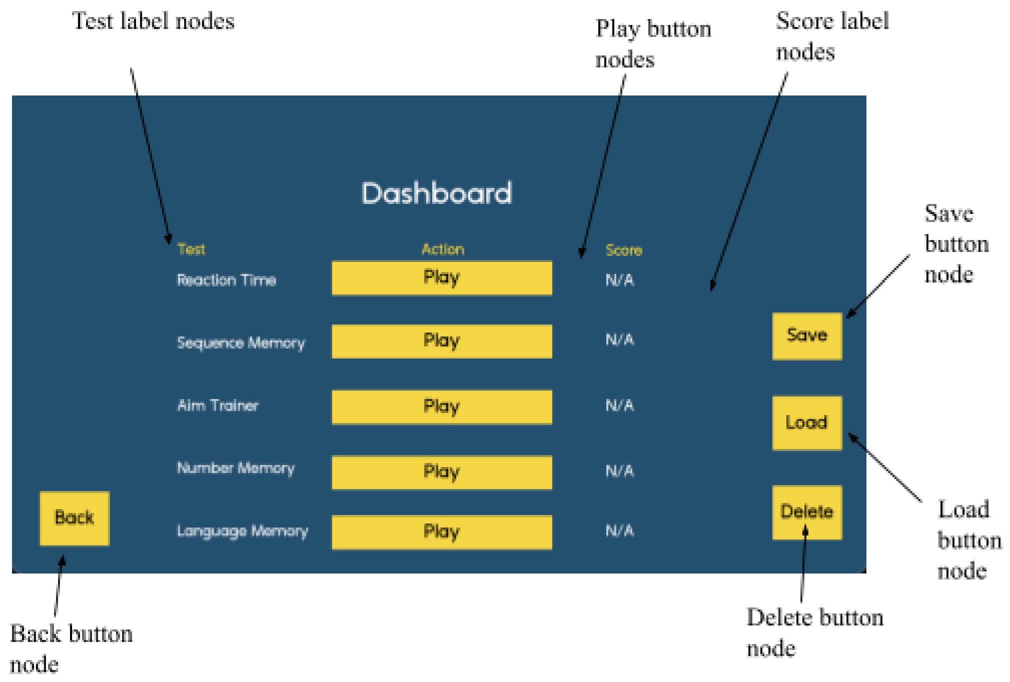
50/50 chance for a new and seen word to be chosen.

```
func new_word():
	var choice = randi_range(1, 2)
	if choice == 1:
		word = "new"
		num = randi_range(0, new.size() - 1)
		$WordLabel.text = new[num]
	elif choice == 2:
		word = "seen"
		num = randi_range(0, seen.size() - 1)
		$WordLabel.text = seen[num]
```

Chooses a random seen or new word from their respective arrays

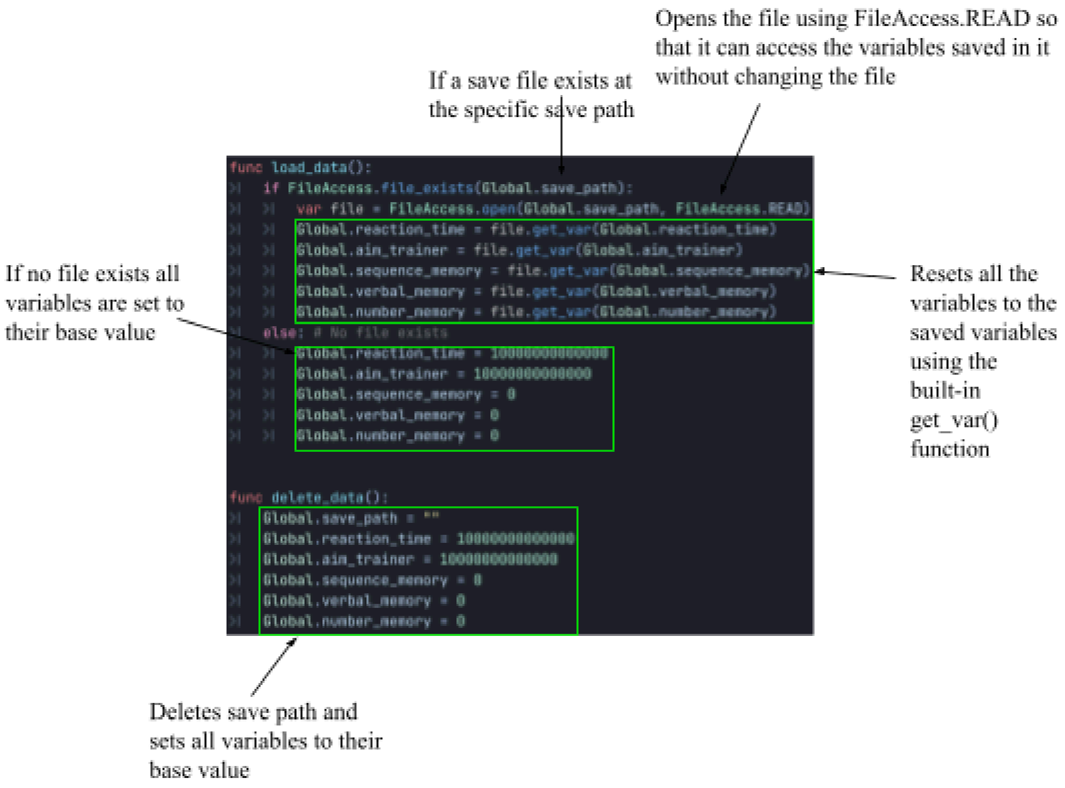
Screenshot #16: *Language Memory Test new_word() and Seen/new Buttons*

Screenshot #16 shows the new_word() function as well as the functions called when the seen and new buttons are pressed. The new_word() function is called at the end of both the seen/ new button functions and will randomly select a word from either the new array or the seen array.



The user can go to the menus of each exercise through the dashboard using the play button nodes and can figure out which exercise to choose from using the test label nodes. This is where the user will be able to load their saved progress using the load button node, optionally restart using the delete button node, and save their data (same as in the main menu scene) using the save button node. The user's score will be shown here as well using the score label nodes. The back button node will navigate the user back to the main menu scene.

Screenshot #17: *Dashboard Scene*

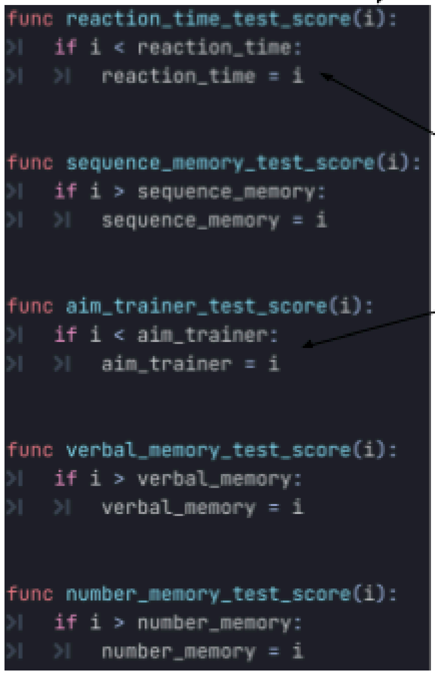

Opens the file using FileAccess.READ so that it can access the variables saved in it without changing the file
If a save file exists at the specific save path
If no file exists all variables are set to their base value
Resets all the variables to the saved variables using the built-in get_var() function
Deletes save path and sets all variables to their base value

```
func load_data():
	if FileAccess.file_exists(Global.save_path):
		var file = FileAccess.open(Global.save_path, FileAccess.READ)
		Global.reaction_time = file.get_var(Global.reaction_time)
		Global.aim_trainer = file.get_var(Global.aim_trainer)
		Global.sequence_memory = file.get_var(Global.sequence_memory)
		Global.verbal_memory = file.get_var(Global.verbal_memory)
		Global.number_memory = file.get_var(Global.number_memory)
	else: # No file exists
		Global.reaction_time = 100000000000000
		Global.aim_trainer = 100000000000000
		Global.sequence_memory = 0
		Global.verbal_memory = 0
		Global.number_memory = 0


func delete_data():
	Global.save_path = ""
	Global.reaction_time = 100000000000000
	Global.aim_trainer = 100000000000000
	Global.sequence_memory = 0
	Global.verbal_memory = 0
	Global.number_memory = 0
```

Screenshot #18: *Loading and Deleting User Data*

Screenshot #18 shows the load_data() and delete_data() functions that do exactly what their names are. The load_data() function retrieves data from the file and delete_data() deletes the file altogether.

Parameter is the
user's score



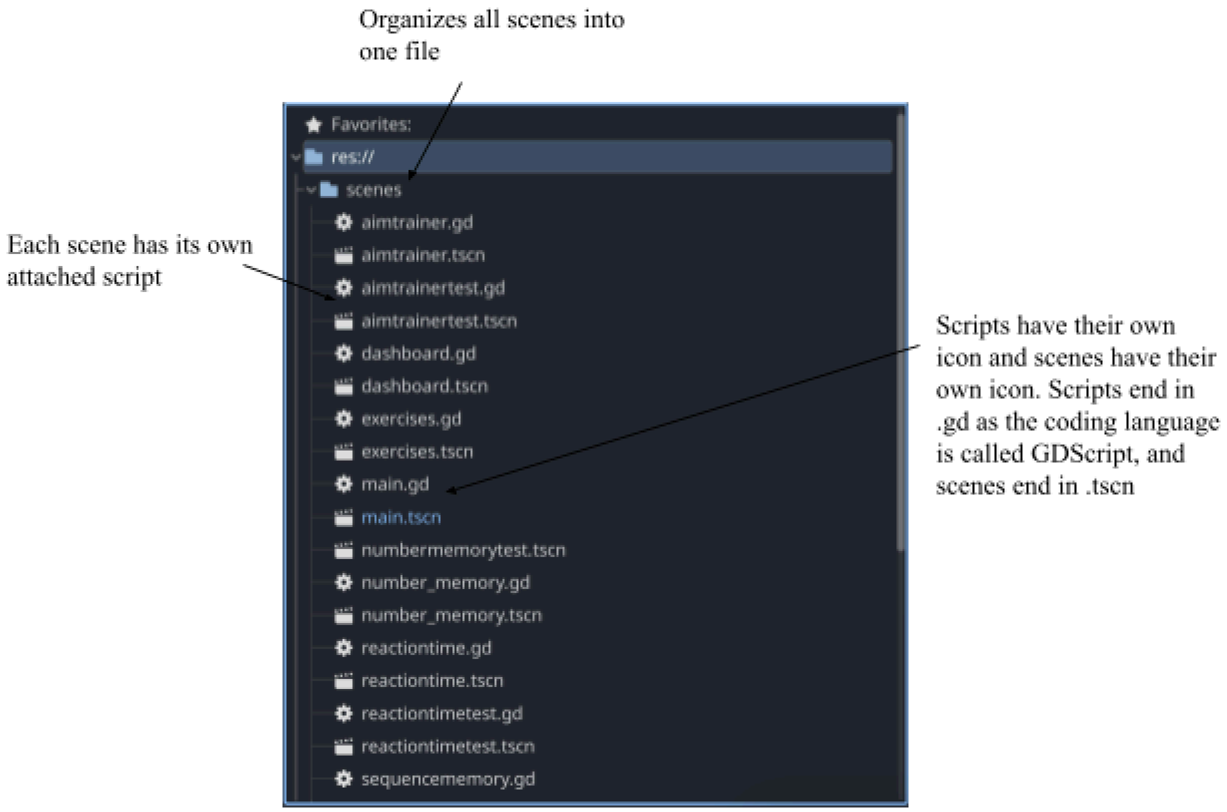
```
func reaction_time_test_score(i):
	if i < reaction_time:
		reaction_time = i


func sequence_memory_test_score(i):
	if i > sequence_memory:
		sequence_memory = i


func aim_trainer_test_score(i):
	if i < aim_trainer:
		aim_trainer = i


func verbal_memory_test_score(i):
	if i > verbal_memory:
		verbal_memory = i


func number_memory_test_score(i):
	if i > number_memory:
		number_memory = i
```

The display will only
change if the user's score is
lower than it previously was

For all other tests the
display will be updated
if the score is higher than
it previously was

These functions will be called when
their respective exercises are completed.

Screenshot #19: *Global Script - Updating Score*



Organizes all scenes into one file

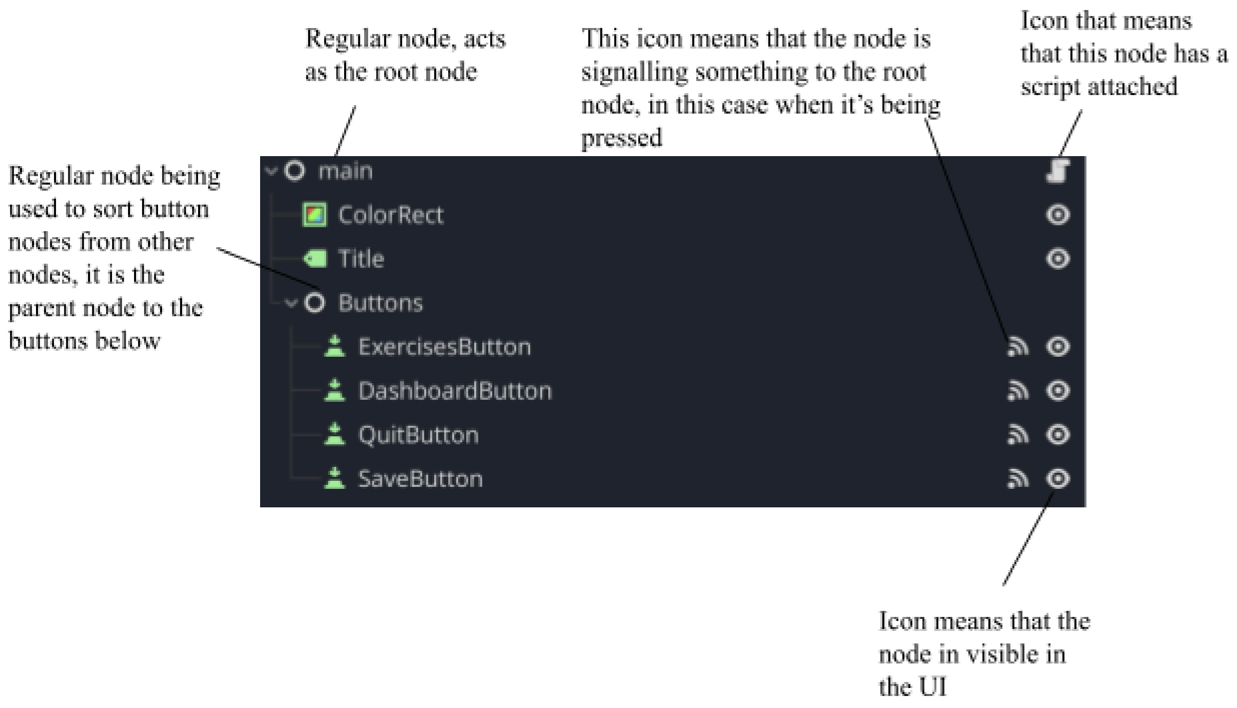
Each scene has its own attached script

Scripts have their own icon and scenes have their own icon. Scripts end in .gd as the coding language is called GDScript, and scenes end in .tscn

```
Favorites:
res://
  scenes
    aimtrainer.gd
    aimtrainer.tscn
    aimtrainertest.gd
    aimtrainertest.tscn
    dashboard.gd
    dashboard.tscn
    exercises.gd
    exercises.tscn
    main.gd
    main.tscn
    numbermemorytest.tscn
    number_memory.gd
    number_memory.tscn
    reactiontime.gd
    reactiontime.tscn
    reactiontimetest.gd
    reactiontimetest.tscn
    sequencememory.gd
```

Screenshot #20: *File Management*

Screenshot #20 shows how the scene and code files are managed.


Regular node, acts as the root node
This icon means that the node is signalling something to the root node, in this case when it's being pressed
Icon that means that this node has a script attached
Regular node being used to sort button nodes from other nodes, it is the parent node to the buttons below
main
ColorRect
Title
Buttons
ExercisesButton
DashboardButton
QuitButton
SaveButton
Icon means that the node in visible in the UI

Screenshot #21: *Node Tree - Main Scene Example*

Screenshot #21 shows an example of a node tree and what all the icons mean.

Word Count: 474