ETHICAL SOFTWARE COMMUNITY

Godot

TABLE OF CONTENTS

Godot Basic

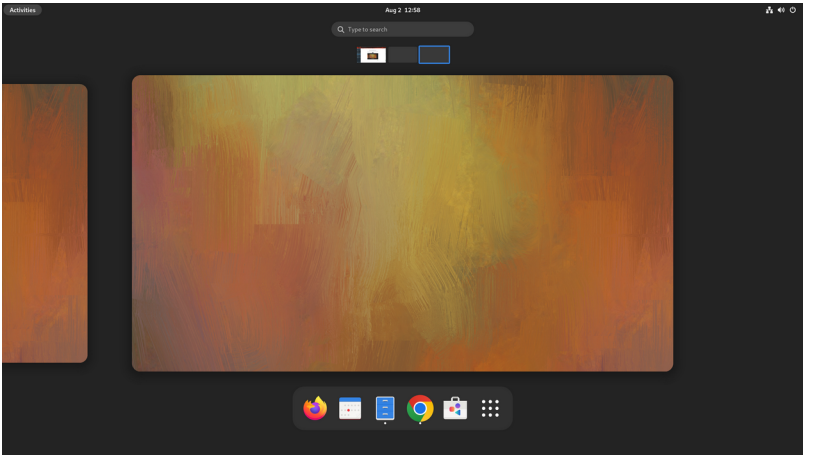
Godot is a cross-platform, free and open-source game engine released under the MIT license.

Installing Godot on Fedora

Step 1. First, on your Fedora, move your mouse to the **top-left corner** of your screen.

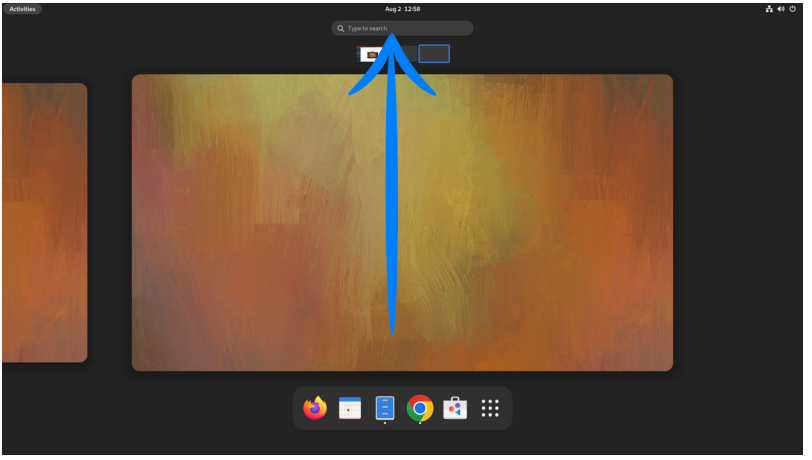
You should see the following screen:

Example Output:



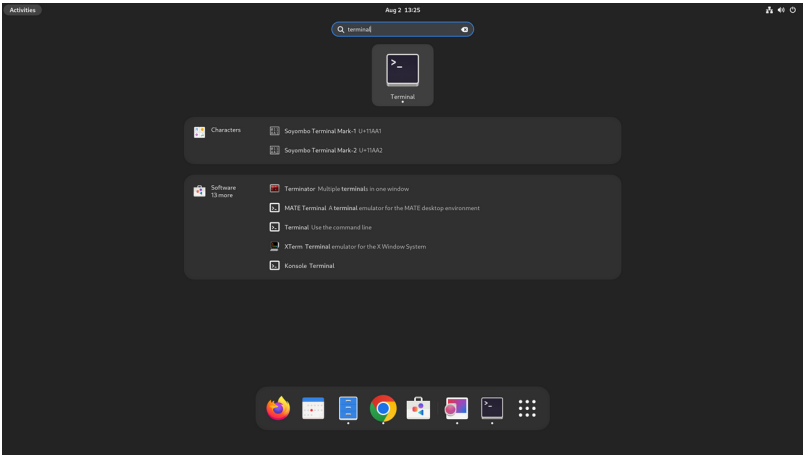
Step 2. Left Click "**Type to search**".

Example Output:



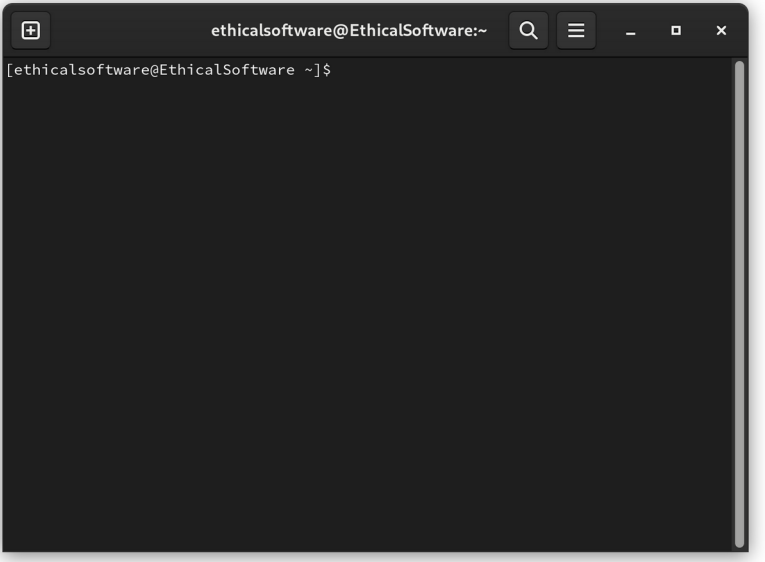
Step 3. Enter the word "**Terminal**".

Example Output:



Step 4. After Left Clicking on the Terminal Icon. You should see the following terminal pop out.

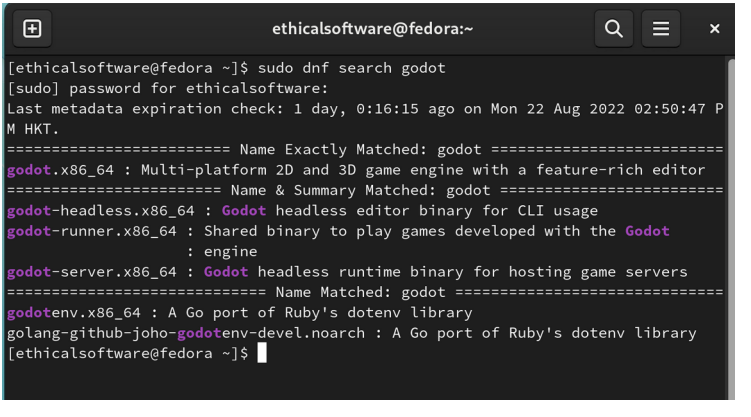
Example Output:



Step 5. Enter the below command in the terminal for searching a package and enter your user password. In our case, we will search for "**godot**".

```
sudo dnf search godot
```

Example Output:

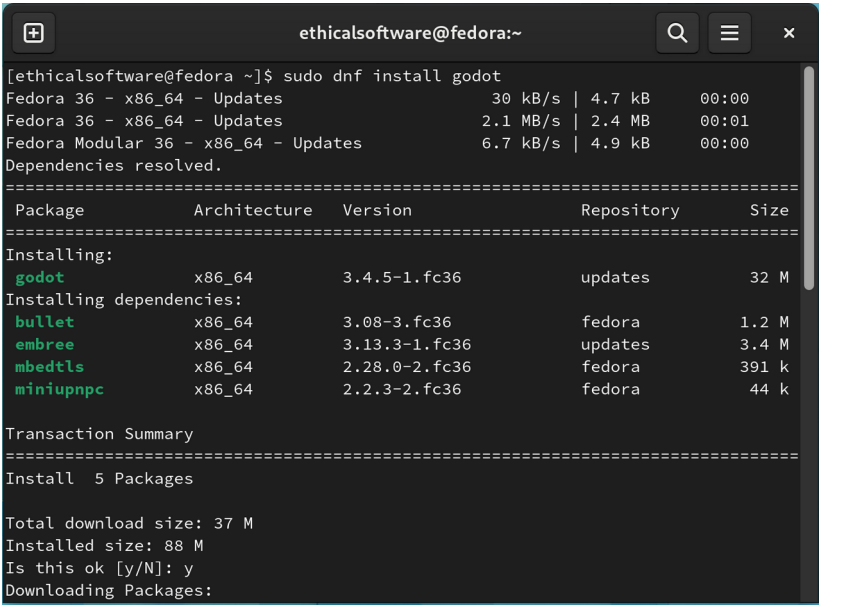


Step 6. As we found that there are “**godot**“packages available for install, we will execute the following command and begin to install “**godot**”.

```
sudo dnf install godot
```

After Entering the above command, type in the “y” key to confirm this installation.

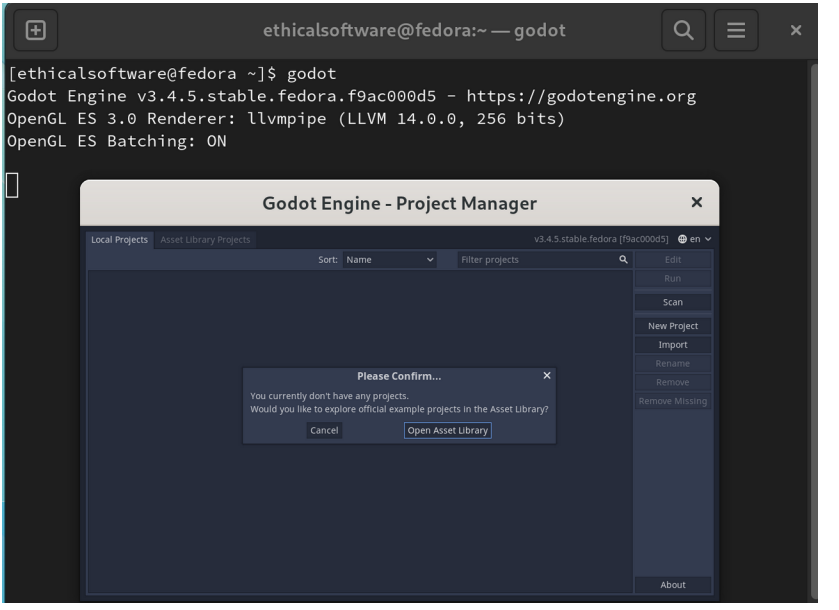
Example Output:



Step 7. If there's no error occurs, enter the following command to start "godot"

```
godot
```

Example Output:

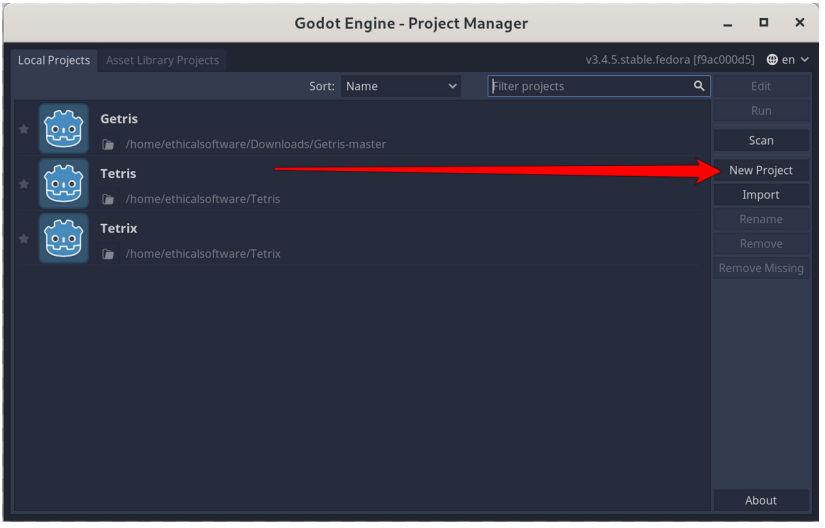


E.N.D

Drag and Drop on Godot

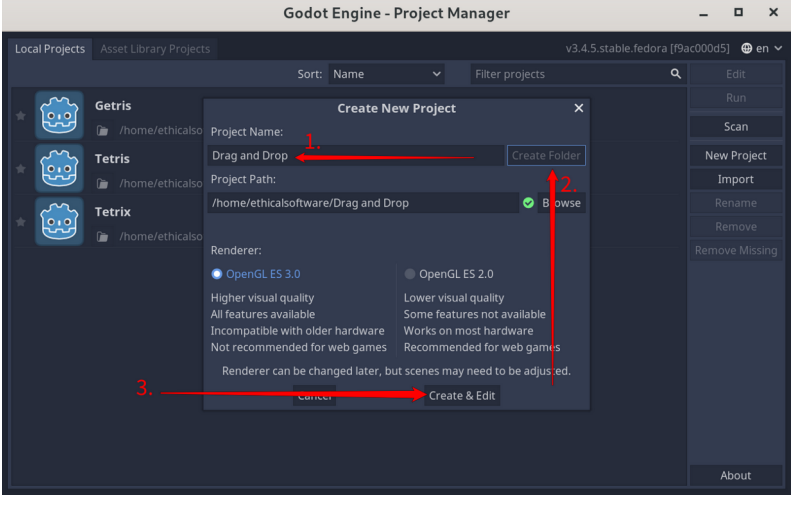
Step 1. Open Godot Application and Click Net Project.

Example Output:



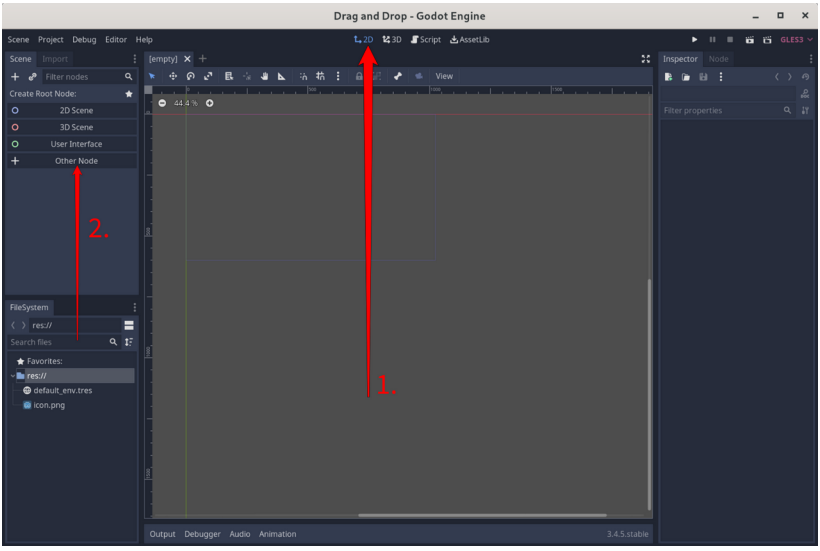
Step 2. Enter the Project Name as “Drag and Drop”, Then Click on “Create Folder”, Finally Click “Create & Edit”.

Example Output:



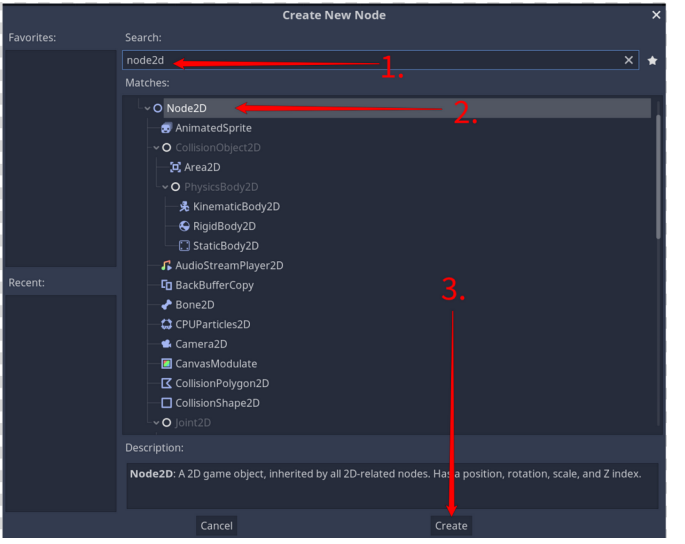
Step 3. Click on the 2D icon to switch to 2D Control Panel. Then Click **Other Node**.

Example Output:



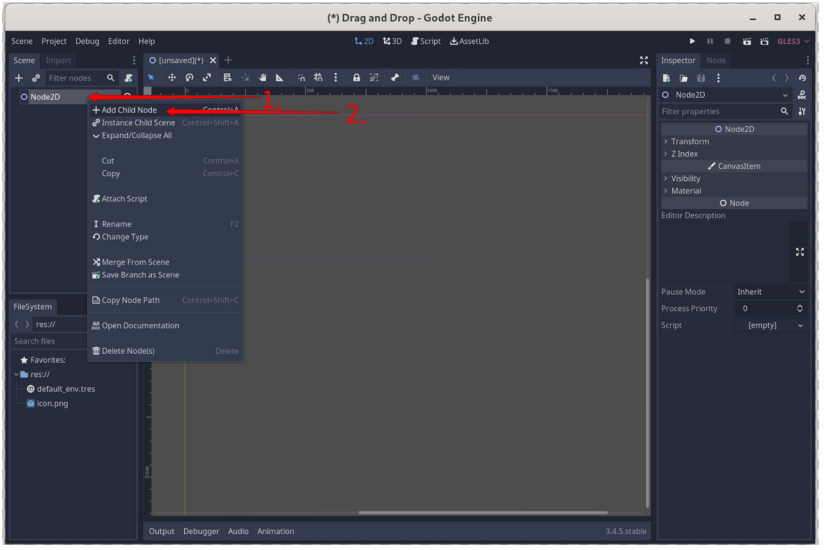
Step 4. Enter “node2d” in search to create a 2D node, and then Click **Create**.

Example Output:



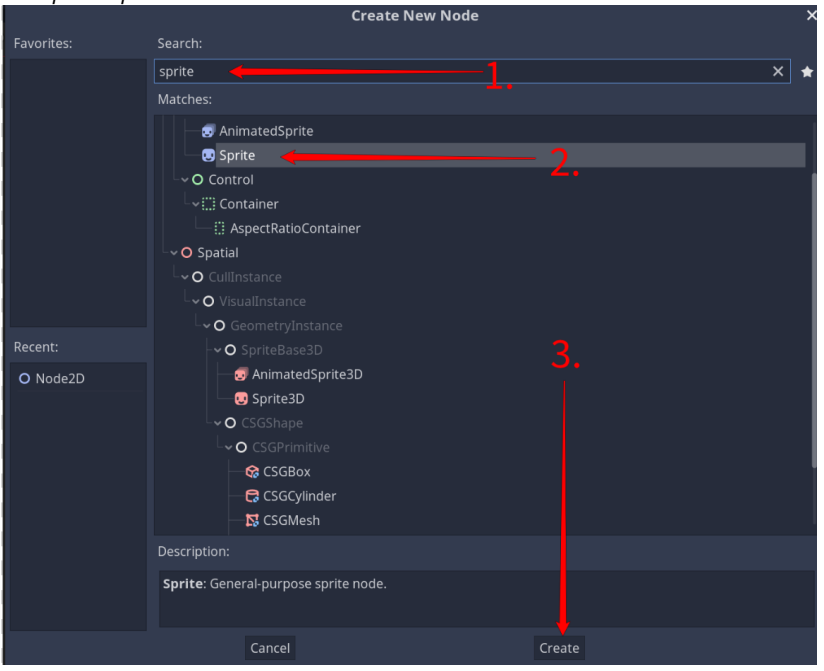
Step 5. **Right Click** On the Node2D, and Left Click "**Add Child Node**".

Example Output:



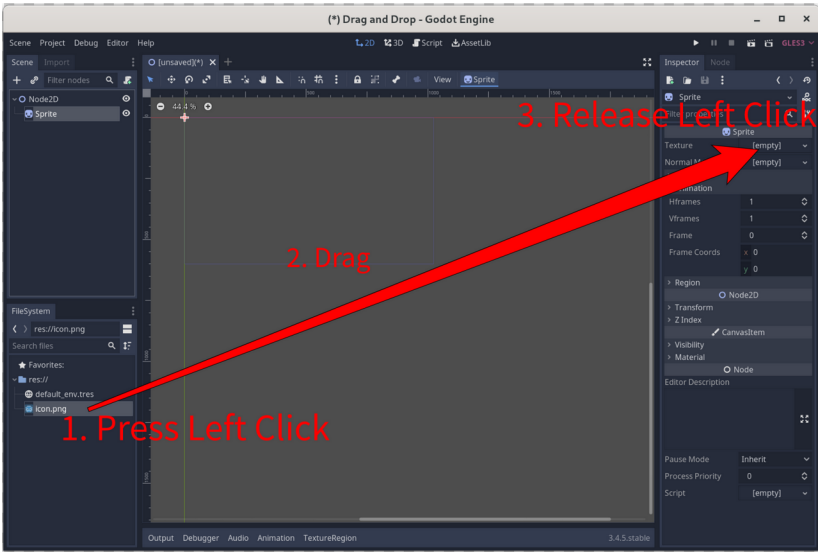
Step 6. Same Process as in Step 4, but enter the word "**sprite**"

Example Output:



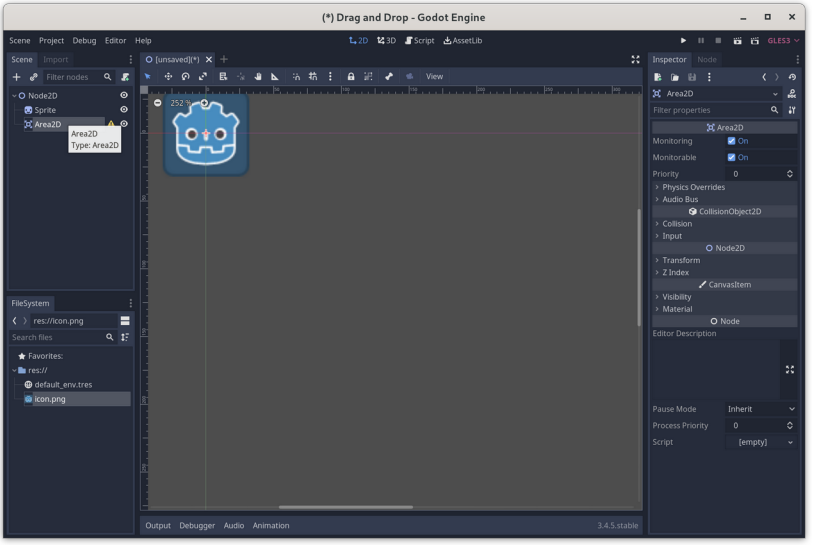
Step 7. Left Mouse **Press** on the “icon.png”, then drag it to the Texture “Empty” Box, and **release** your Left Mouse.

Example Output:



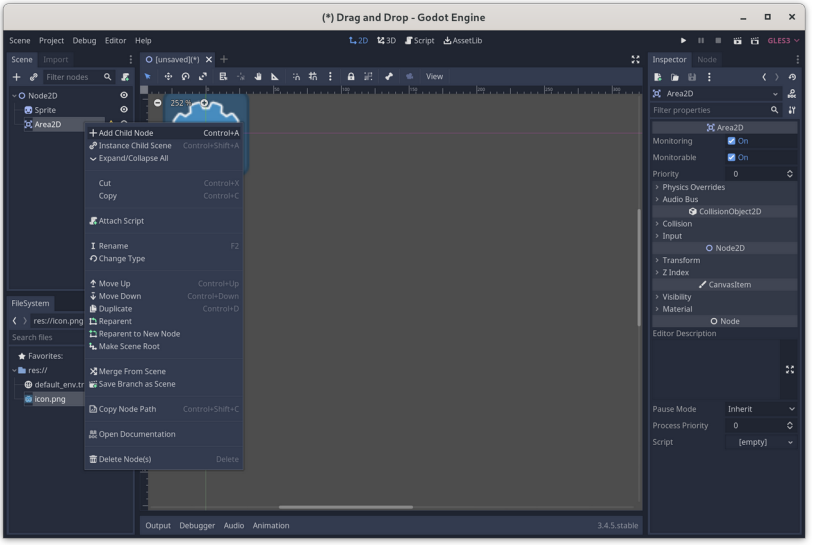
Step 8. Follow **Step5-6**, but in Step 6, enter the word “**area2d**”

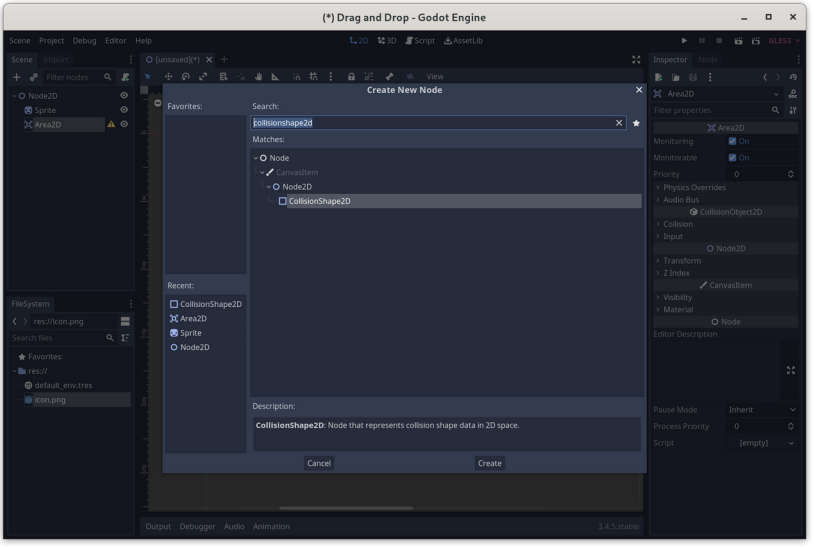
Example Output:

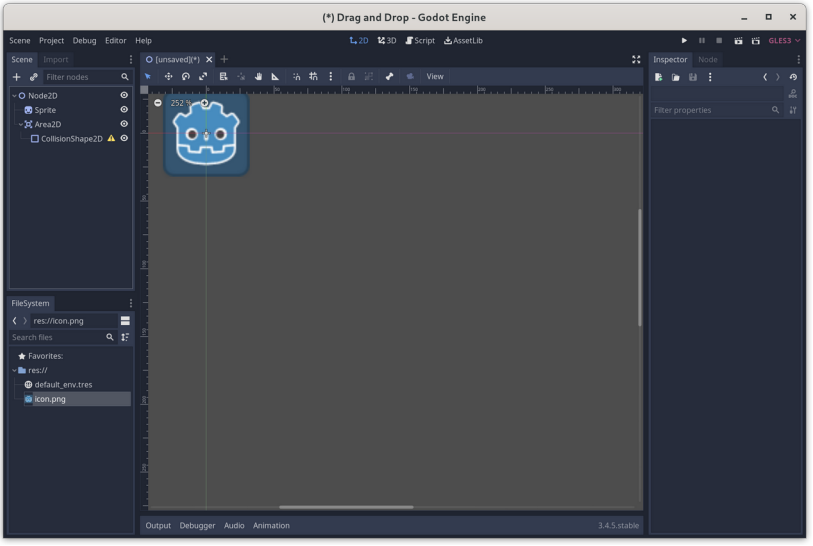


Step 9. **Right Click** On the Area2D, and press "**Add child Node**". Then Follow **Step 6** but enter the word "**collisionshape2d**"

Example Output:

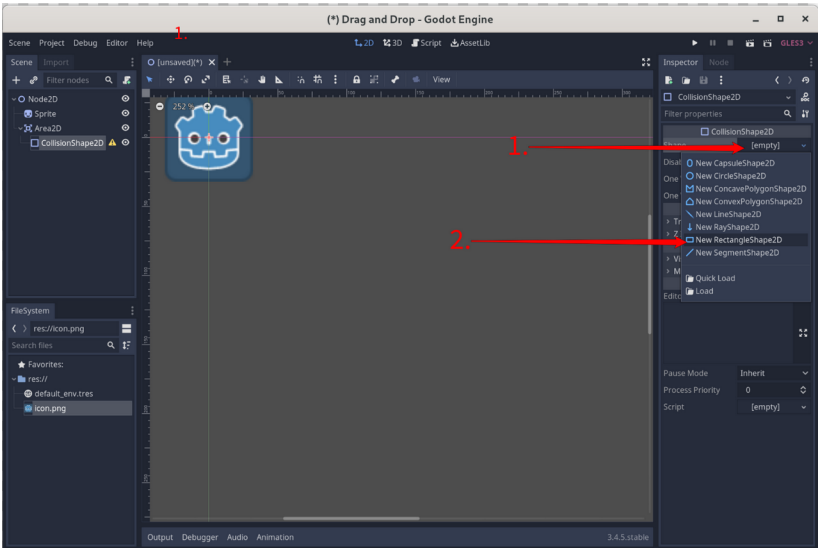






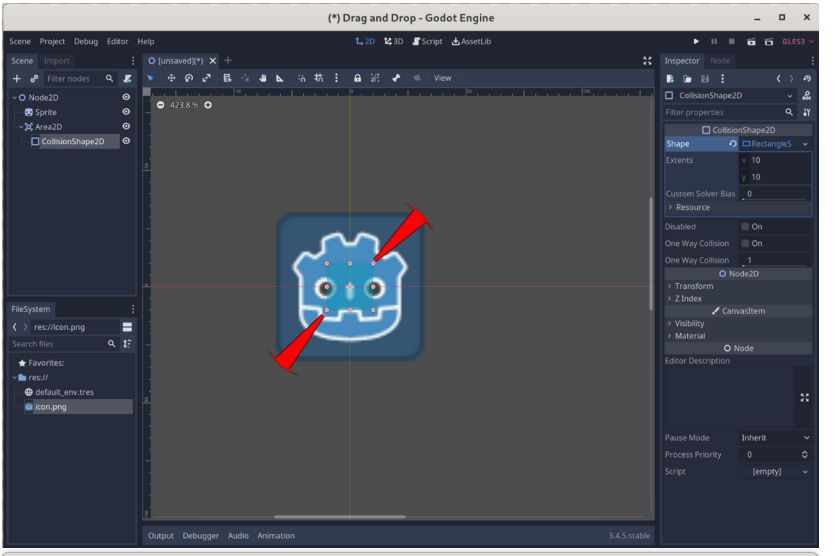
Step 10. Select The shape of "**collisionshape2d**" to be "New RectangleShape2D"

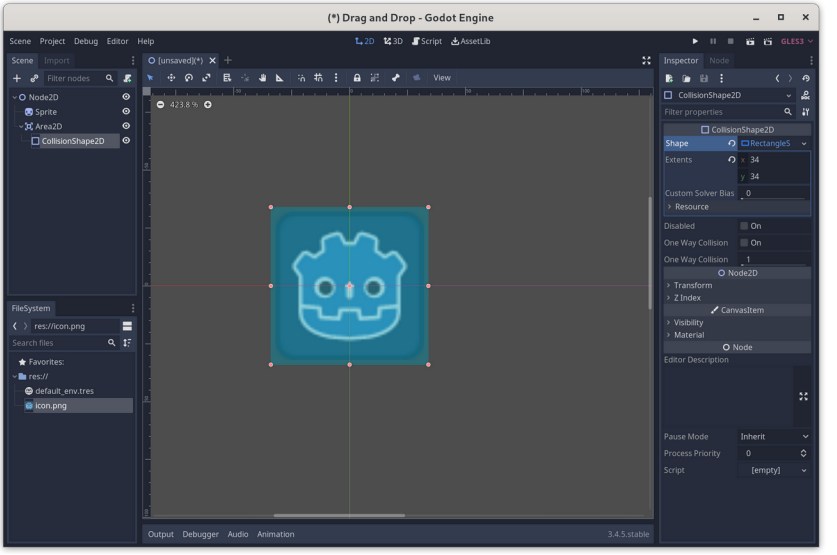
Example Output:



Step 11. Drag to two Red Edge Points of the Half Blue Rectangle and drag its size to be covering the while icon.

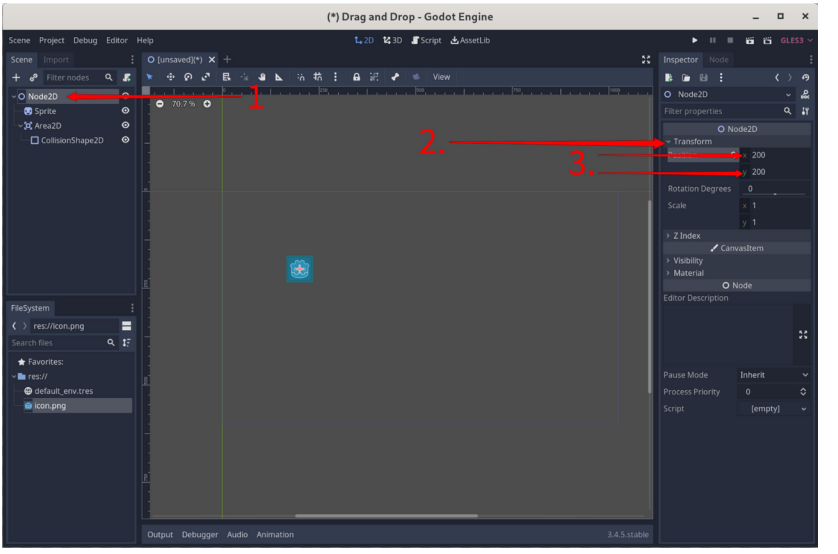
Example Output:





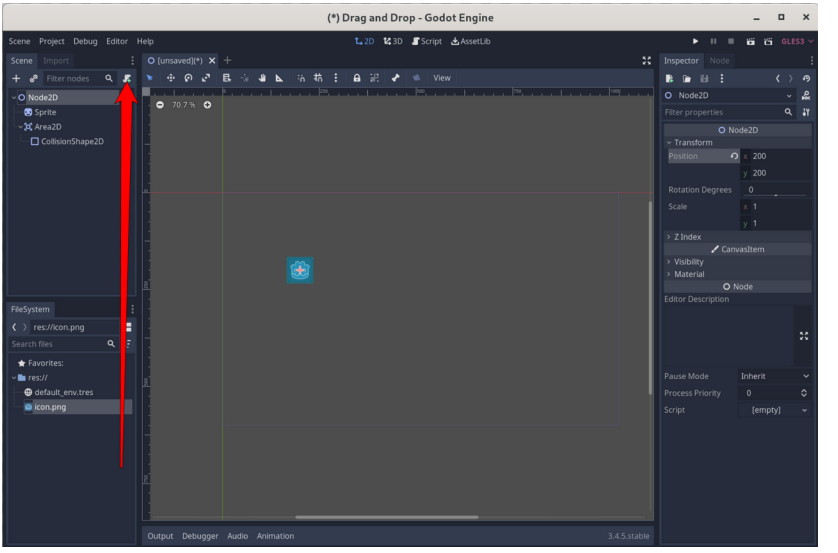
Step 12. Click on the "**Node2D**" and select the "**Transform**" option, then enter the X and Y coordinate of this node. In my case, I use x = 200, y = 200.

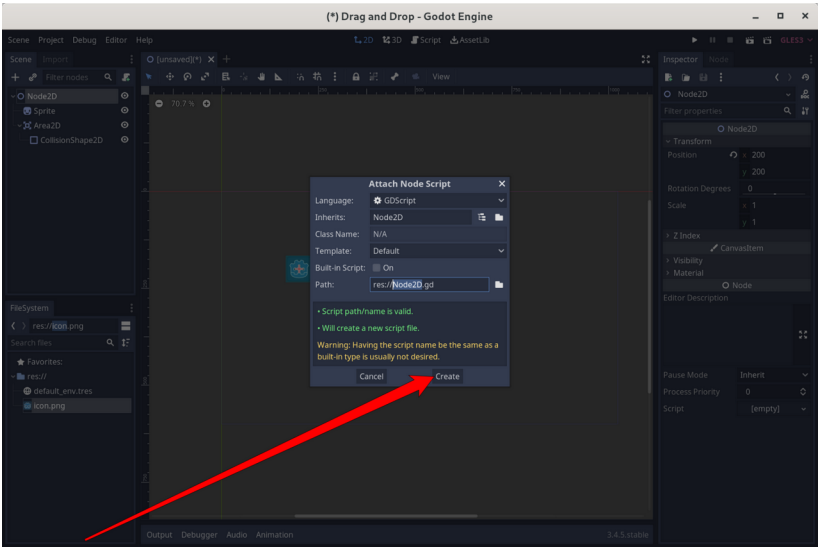
Example Output:

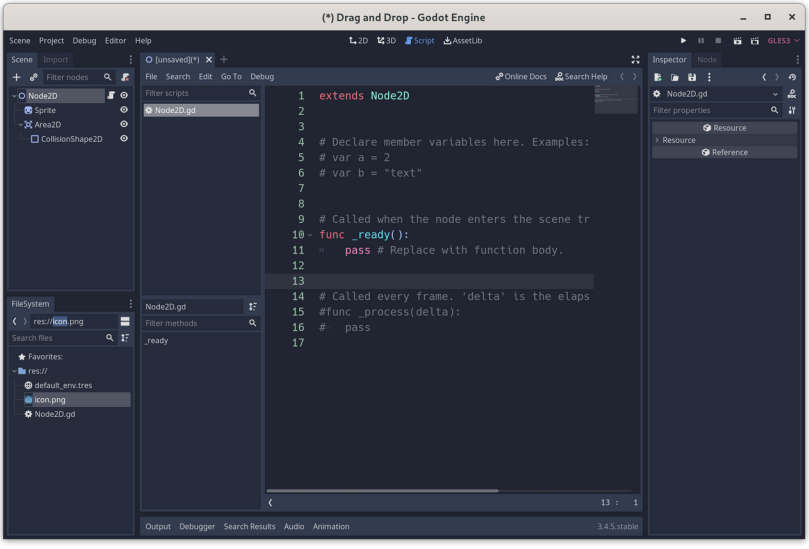


Step 13. Click on the End of the Red Arrow, it allows you to create script for "Node2D", then Click Create in the popped out window.

Example Output:

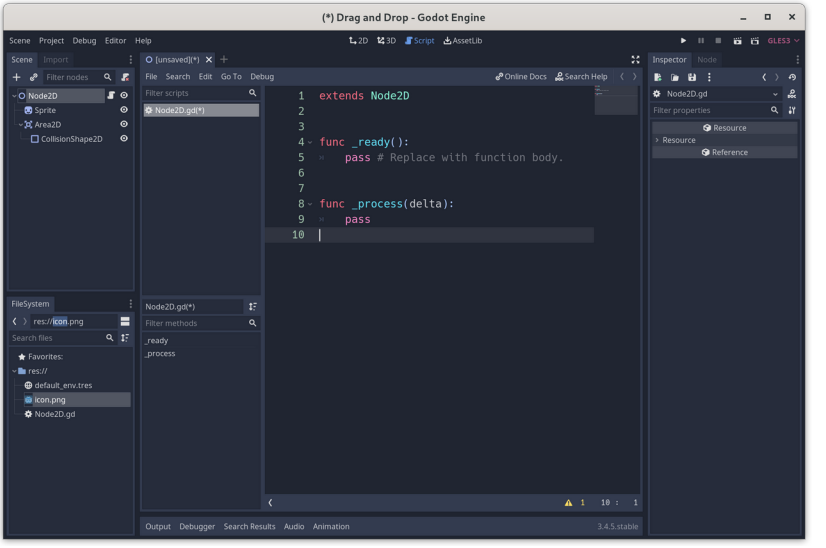


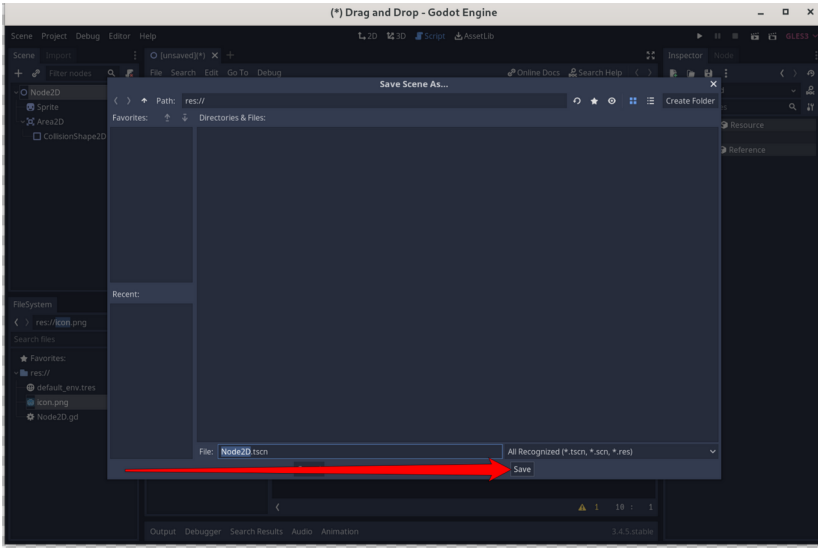

(*) Drag and Drop - Godot Engine
Attach Node Script
Language:
GDScript
Inherits:
Node2D
Class Name:
N/A
Template:
Default
Built-in Script:
On
Path:
res://Node2D.gd
• Script path/name is valid.
• Will create a new script file.
Warning: Having the script name be the same as a built-in type is usually not desired.
Cancel
Create


(*) Drag and Drop - Godot Engine
extends Node2D
Declare member variables here. Examples:
var a = 2
var b = "text"
Called when the node enters the scene tr
func _ready():
pass # Replace with function body.
Called every frame. 'delta' is the elaps
#func _process(delta):
pass

Step 14. Change the Codes inside as the codes shown below. Then hit the shortcut “ctrl + s” to save this file.

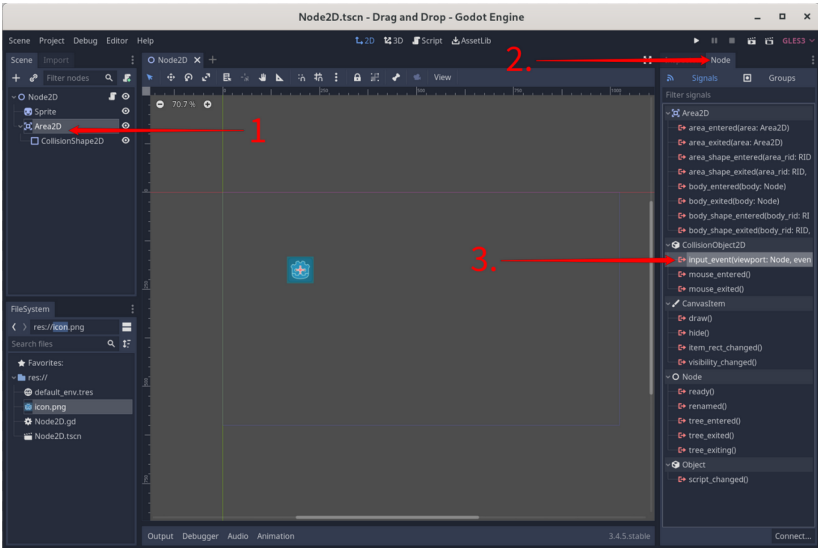
Example Output:

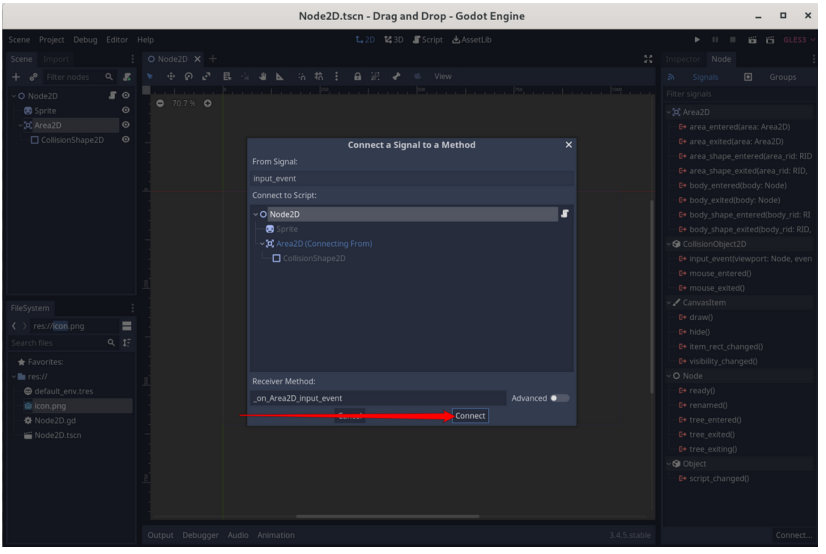




Step 15. Click on the Area2D icon, then Select the Node Tag. After that, Double Click on the "**input_event**". Last, when the popped up window appear, Click on "**Connect**".

Example Output:





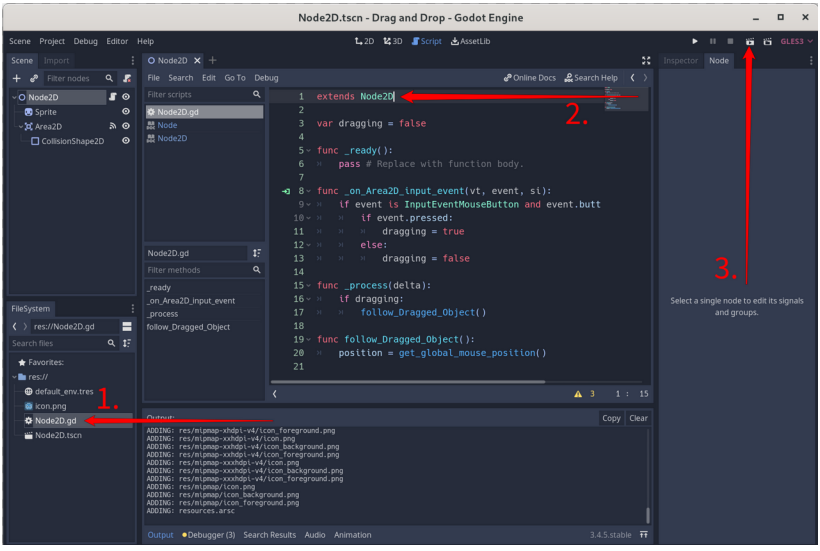
Step 16.

1. Click on the Node2D.gd script.

2. Type or Copy the Logic from below.

3. Play this Scene.

Example Output:



Codes:

```
extends Node2D

var dragging = false

func _ready():
    pass # Replace with function body.

func _on_Area2D_input_event(vt, event, si):
    if event is InputEventMouseButton and event.button_index ==
BUTTON_LEFT:
```

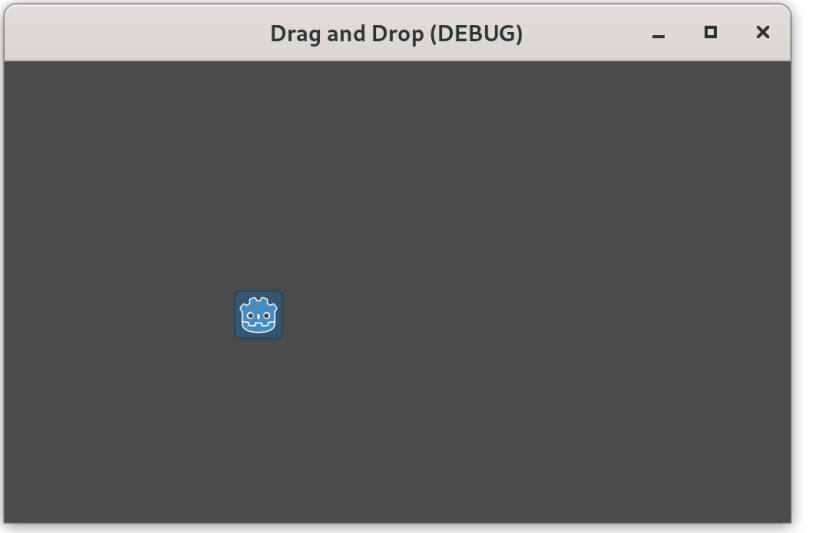
```
        if event.pressed:
            dragging = true
        else:
            dragging = false

func _process(delta):
    if dragging:
        follow_Dragged_Object()

func follow_Dragged_Object():
    position = get_global_mouse_position()
```

Logics:

- We have a variable named “dragging”, it helps us to determine whether the Icon is being dragged or not.
- The function “_on_Area2D_input_event” takes care of the mouse input and check If the mouse has pressed the icon, if so, change “dragging” to True. If not, vice-versa.
- The function “_process” will keep running when the Scene is playing. It serves as a controller to trigger the “follow_Dragged_Object” function.
- The function “follow_Dragged_Object” will set the current position of the “Node2D” as the mouse current position which makes it follows the mouse when dragging.

Step 17. Left Click on the Godot Icon and play with This Little guy.

Example Output:

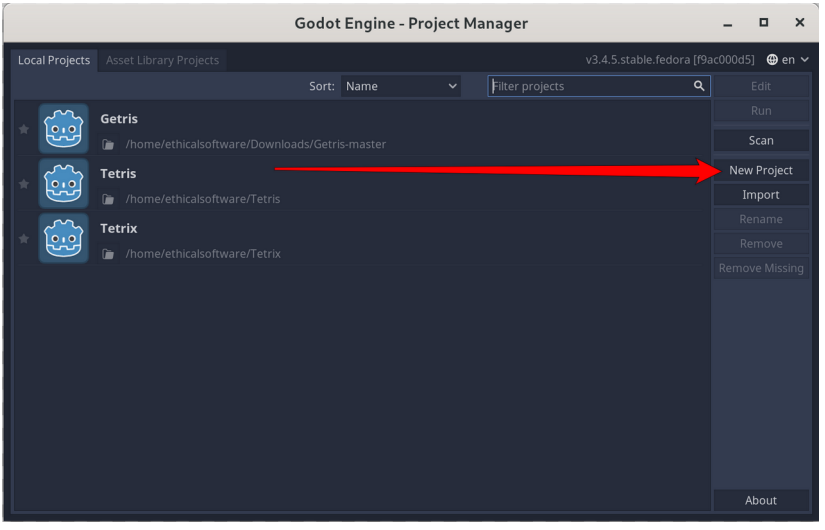


E.N.D

Swipe and Touch on Godot

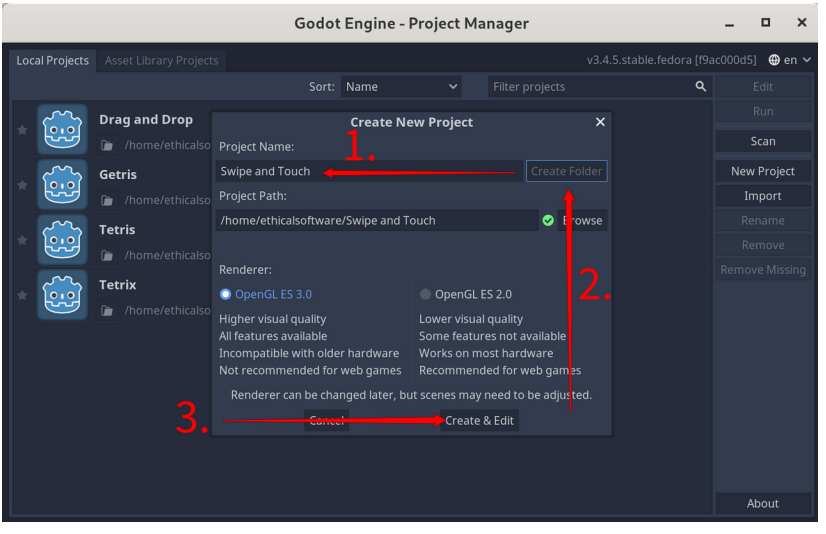
Step 1. Open Godot Application and Click Net Project.

Example Output:



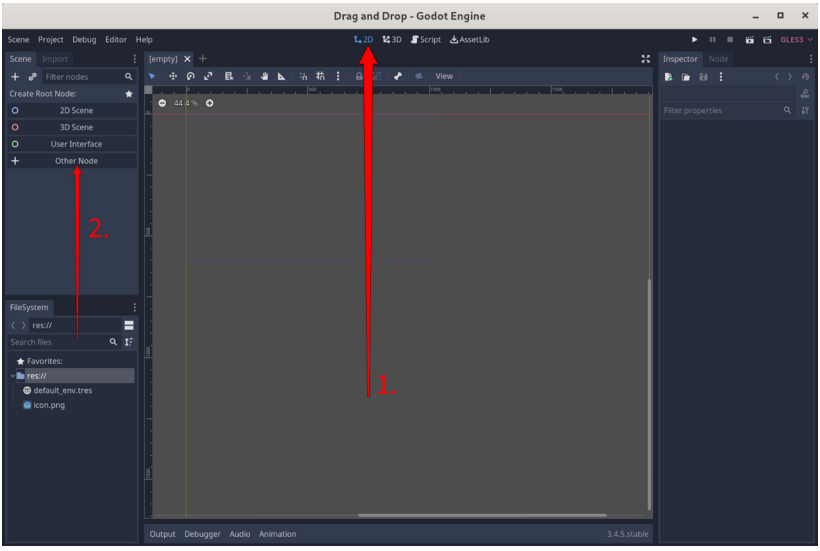
Step 2. Enter the Project Name as “Swipe and Touch”, Then Click on “Create Folder”, Finally Click “Create & Edit”.

Example Output:



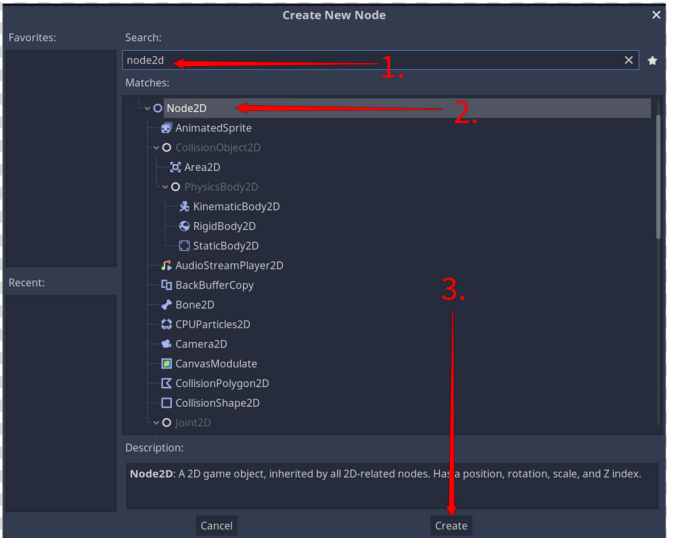
Step 3. Click on the 2D icon to switch to 2D Control Panel. Then Click **Other Node**.

Example Output:



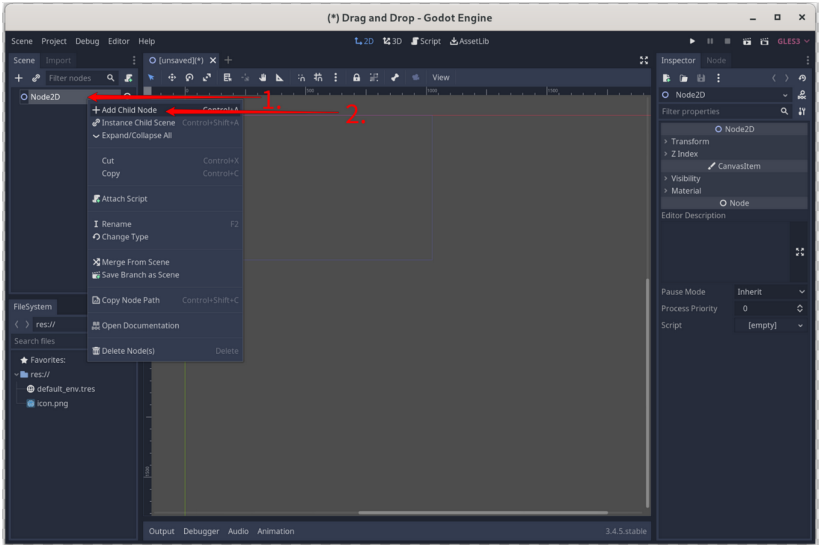
Step 4. Enter “node2d” in search to create a 2D node, and then Click **Create**.

Example Output:



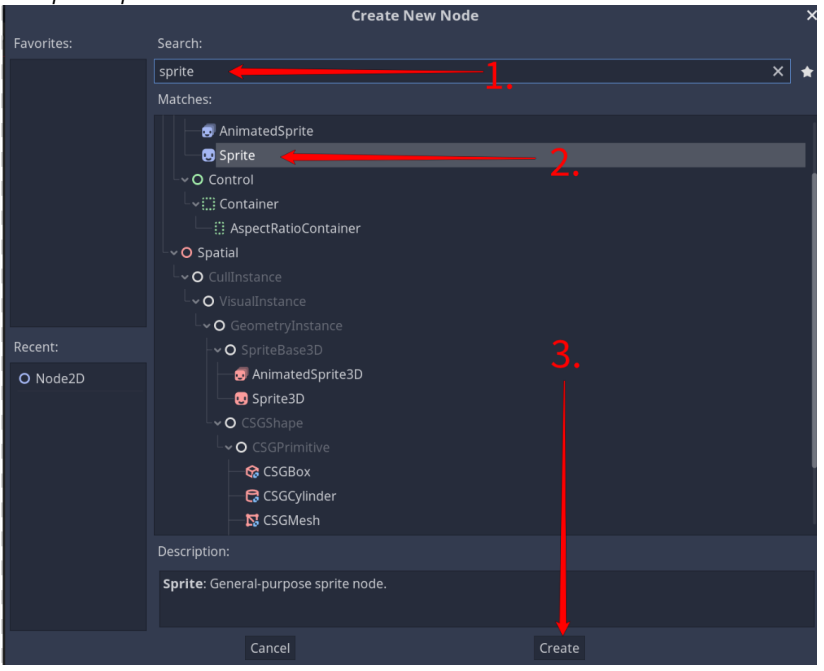
Step 5. **Right Click** On the Node2D, and Left Click "**Add Child Node**".

Example Output:



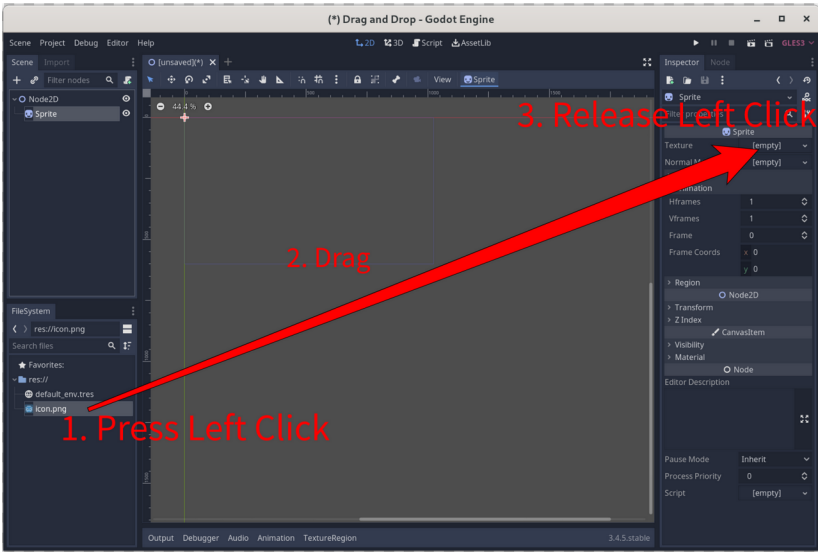
Step 6. Same Process as in Step 4, but enter the word "**sprite**"

Example Output:



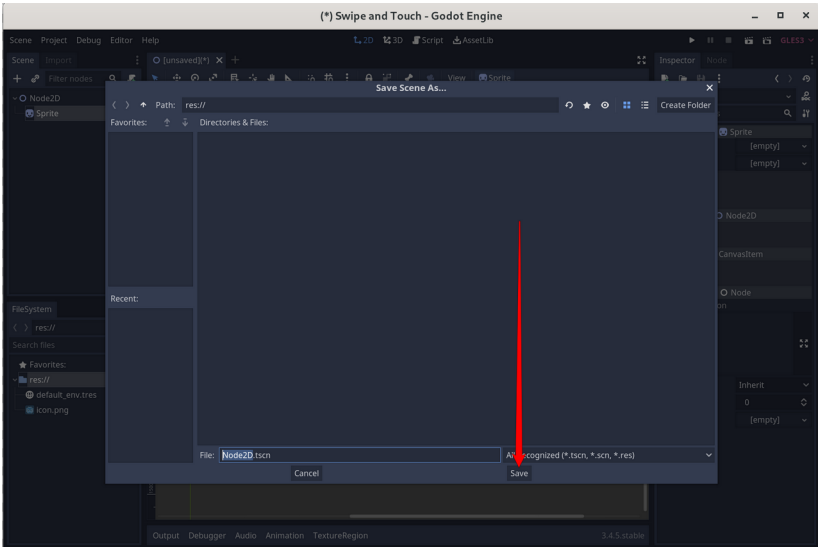
Step 7. Left Mouse **Press** on the “icon.png”, then drag it to the Texture “Empty” Box, and **release** your Left Mouse.

Example Output:



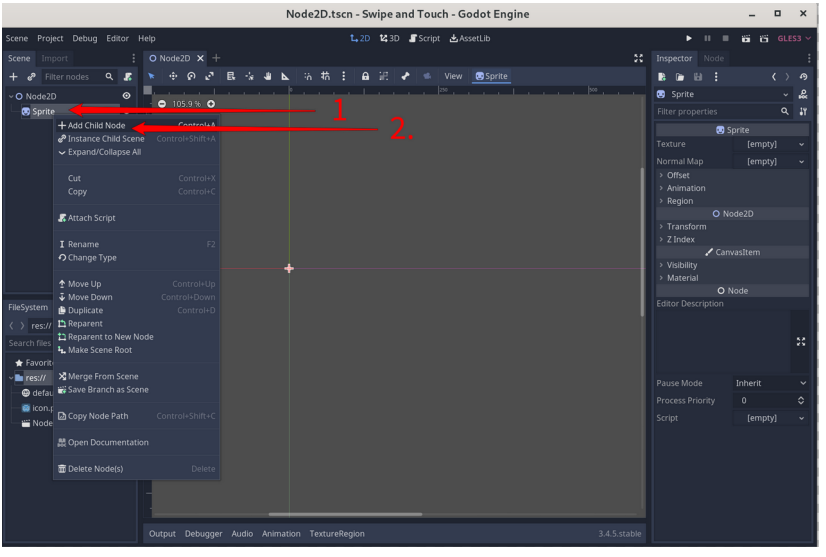
Step 8. Hit the shortcut “**ctrl + s**” to save this Scene.

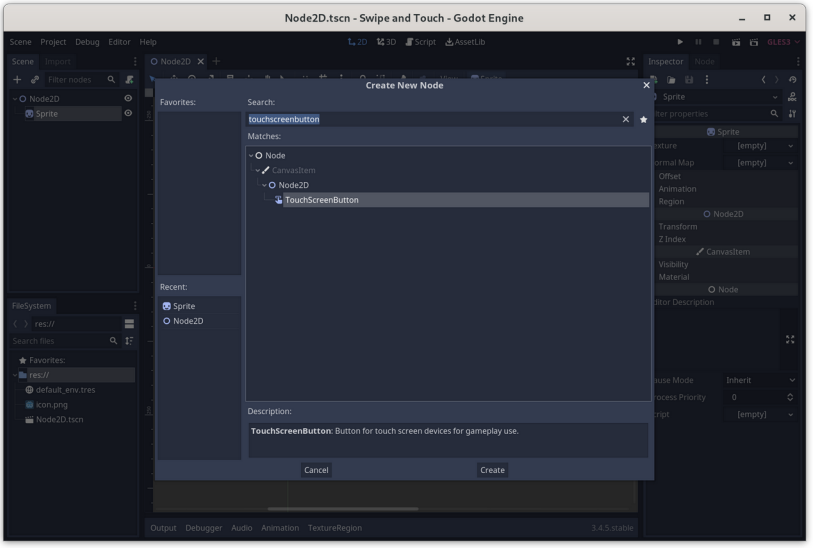
Example Output:

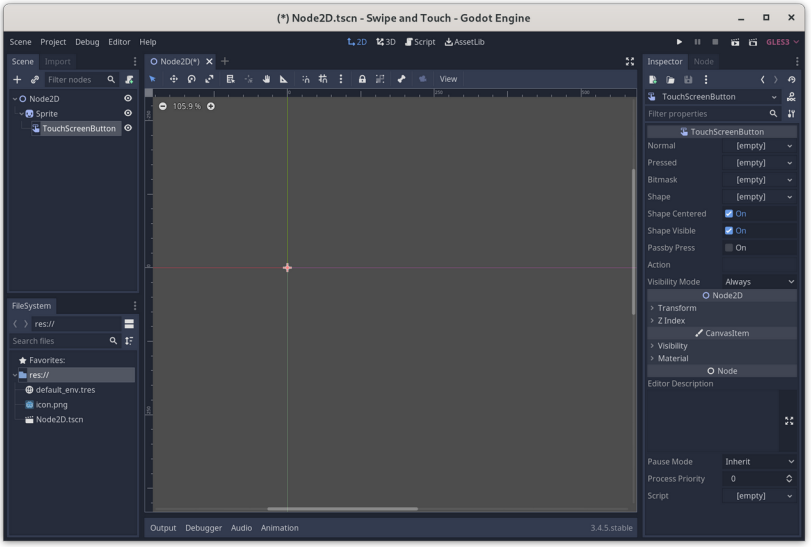


Step 9. Right Click on the Sprite and Left Click Add Child Node. Then Follow Step 6 but enter the word "**touchscreenbutton**".

Example Output:

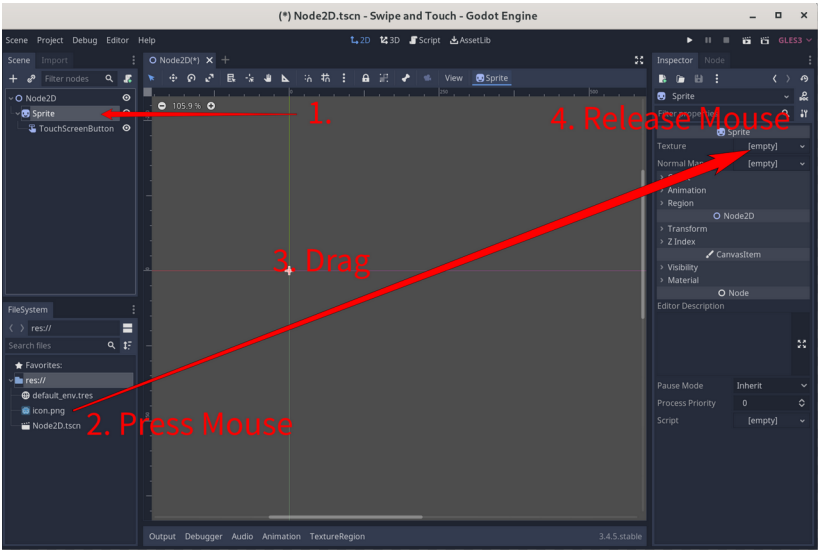


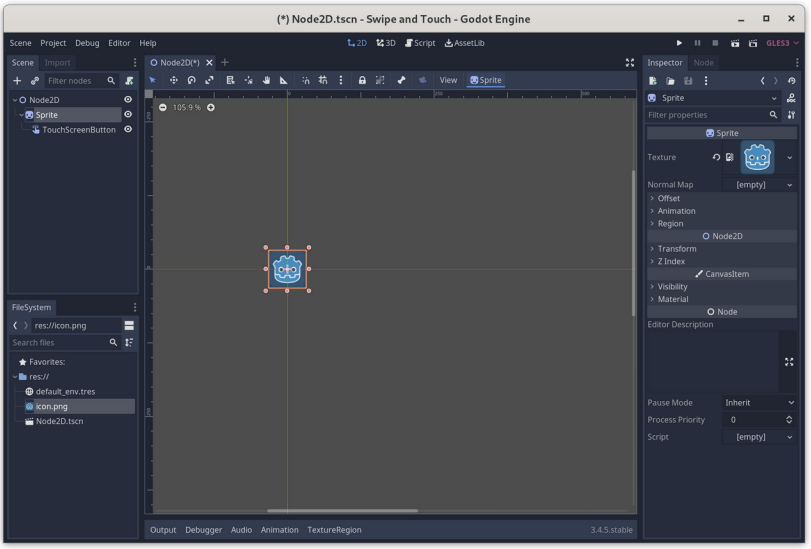




Step 10. Left Click on the Sprite, then drag the “icon.png” to the Texture Field.

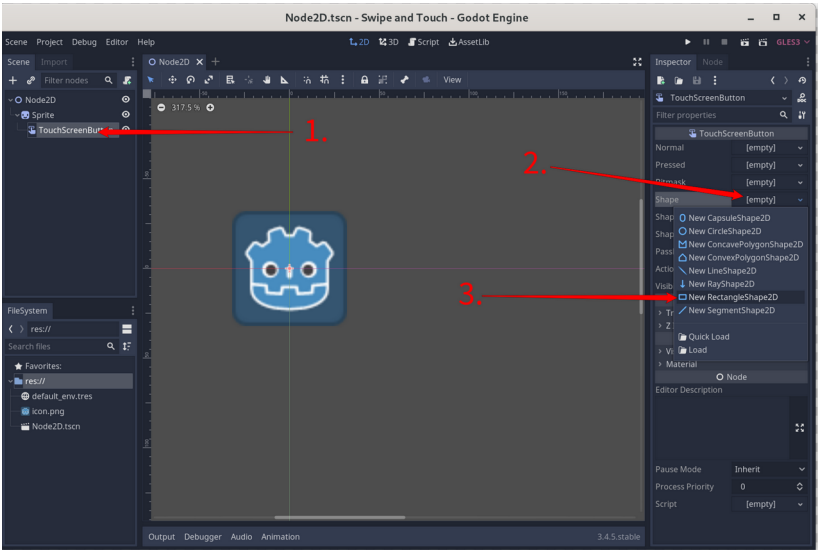
Example Output:

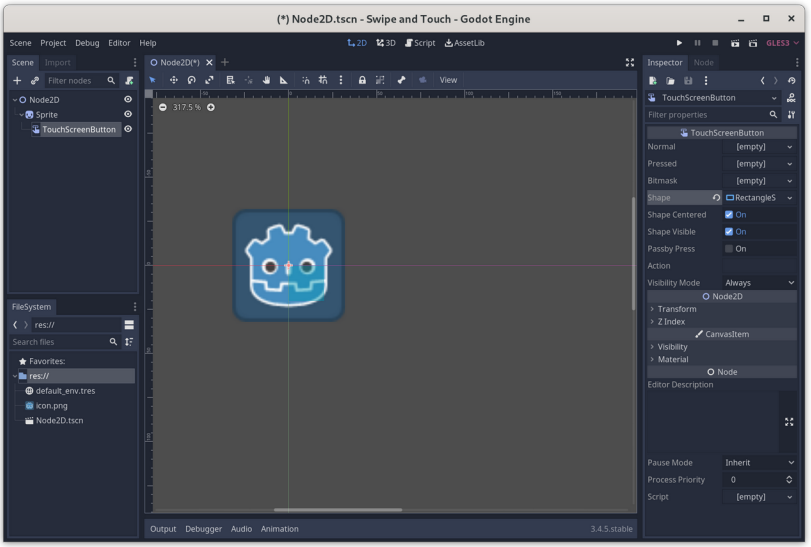




Step 11. First, Left Click on the "**TouchScreenButton**", then select the "empty" Shape Box. After that, pick the New RectangleShape2D.

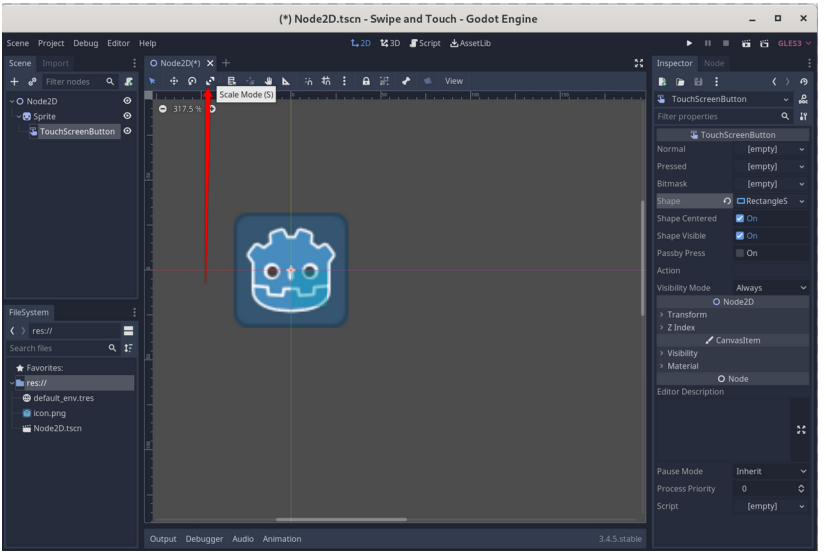
Example Output:





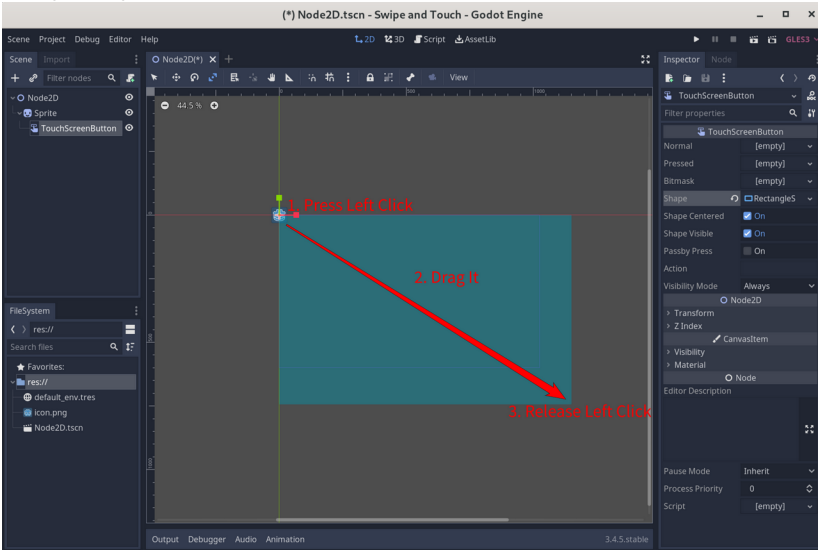
Step 12. Click On the Scale Mode Button as pointed by the red arrow.

Example Output:



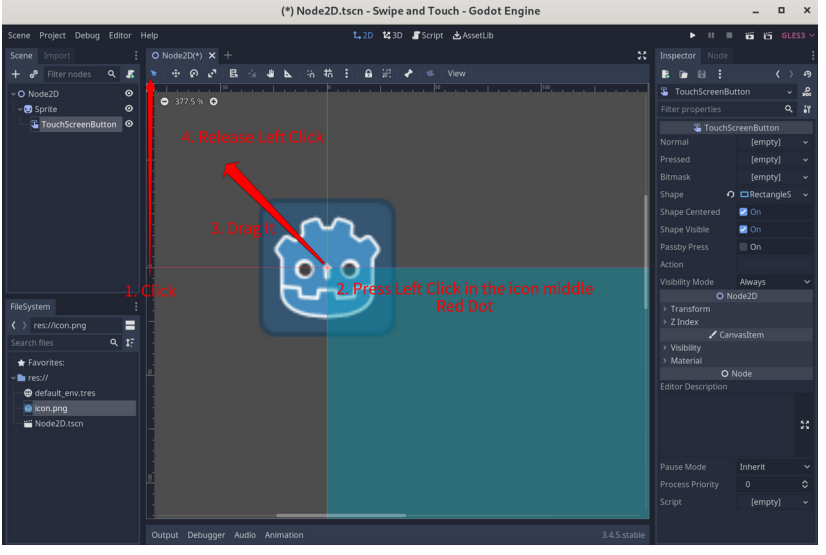
Step 13. **Press** your left key and **drag** it to the bottom right corner to cover the blue border of the scene area. Then **release** your left key.

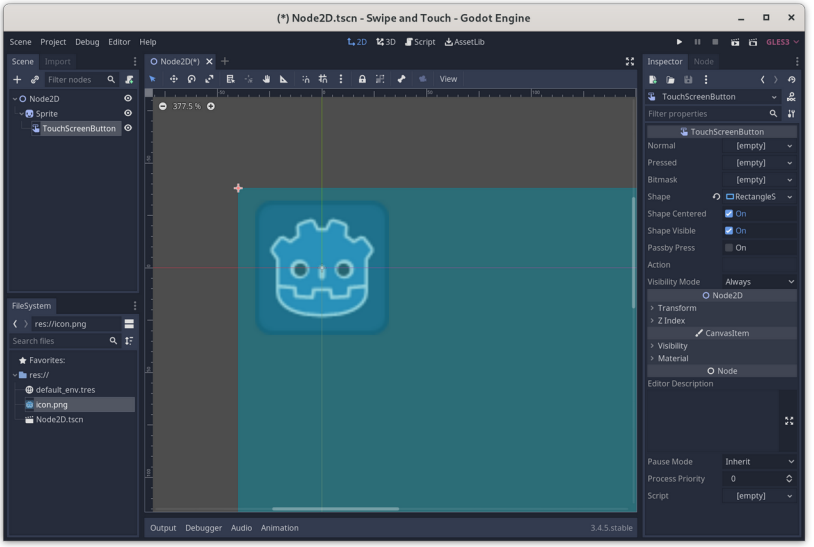
Example Output:



Step 14. First, Click on the "**Select Mode Button**", and **Press** the Little Red Dot in the middle of the Icon, then **Drag** it to the top-left area. Then **release** the press.

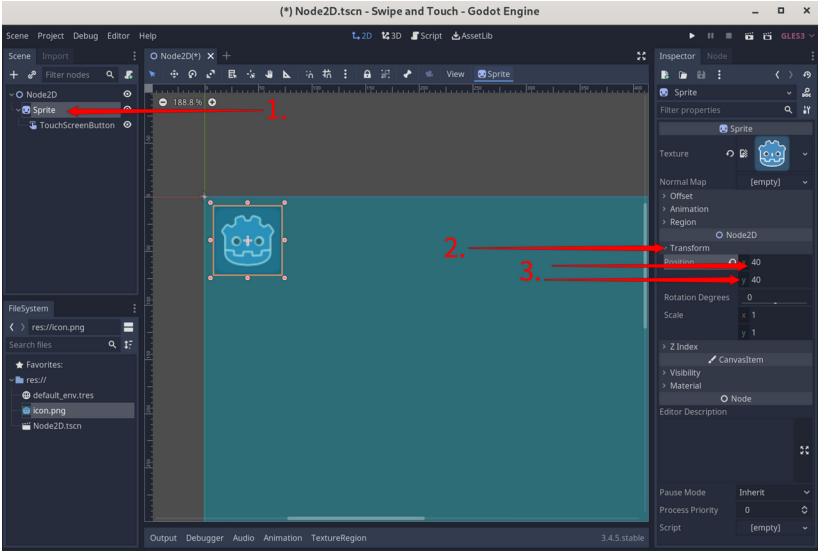
Example Output:





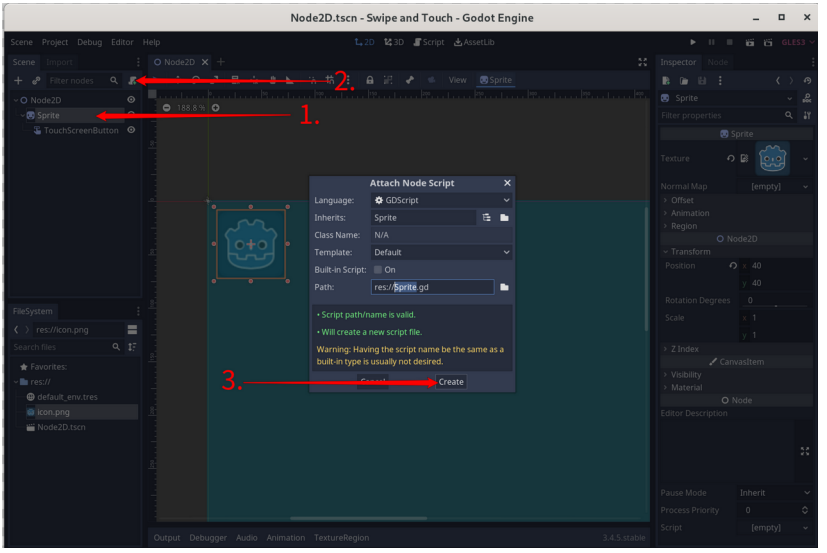
Step 15. Click on the “Sprite” and Click on the “Transform” Label. Then enter x and y to be “40”.

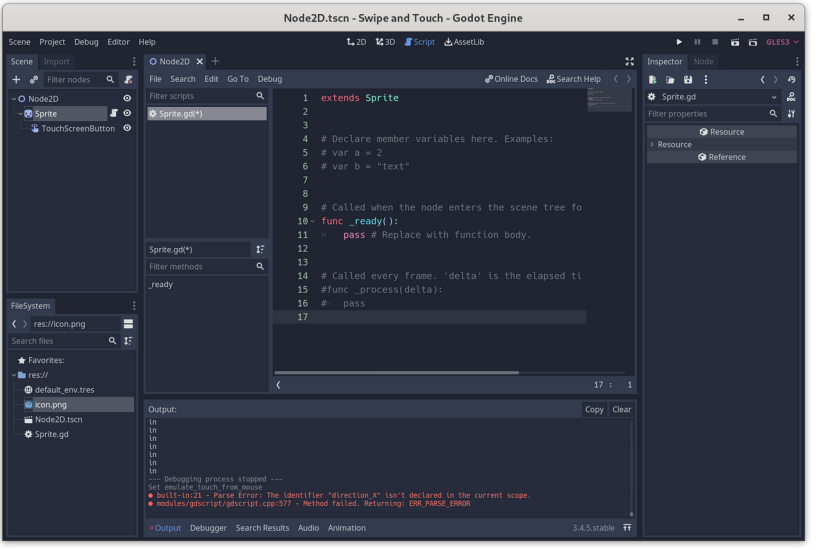
Example Output:



Step 16.Click on the "**Sprite**" and then Click on the "**Add Script**" Button. After, Click "**Create**" to create the script.

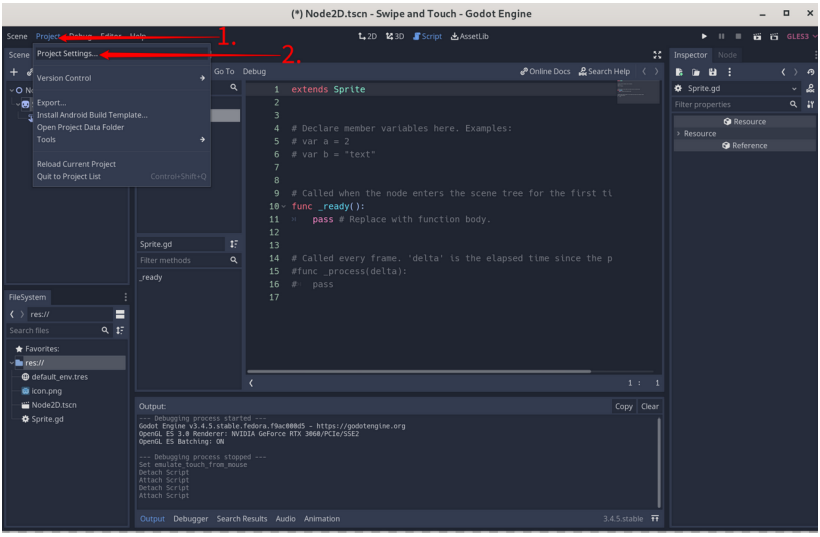
Example Output:



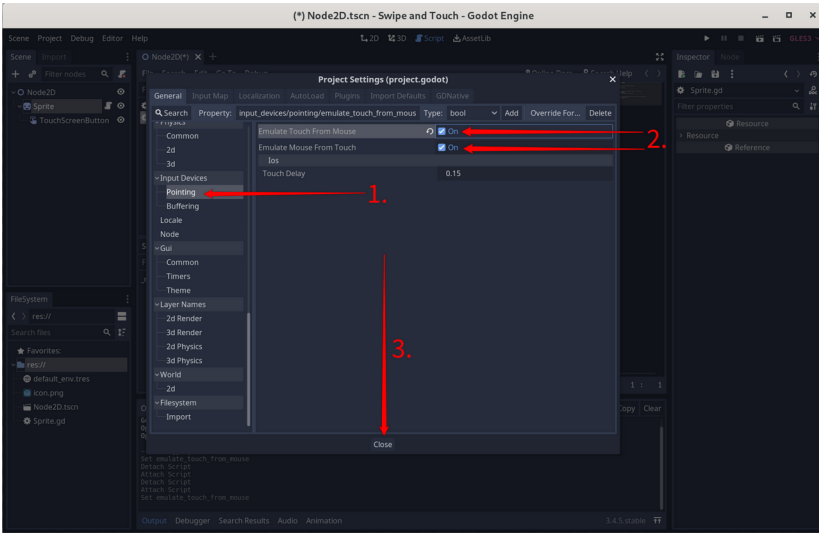


Step 17. Click On the “**Project**” Option and go to “**Project Settings**”.

Example Output:



- Find the “**Pointing**” Under “**Input Devices**”, and turn **on** the options in “**2.**”, then your mouse could work as finger touch on the screen. At last, Click “**Close**”.



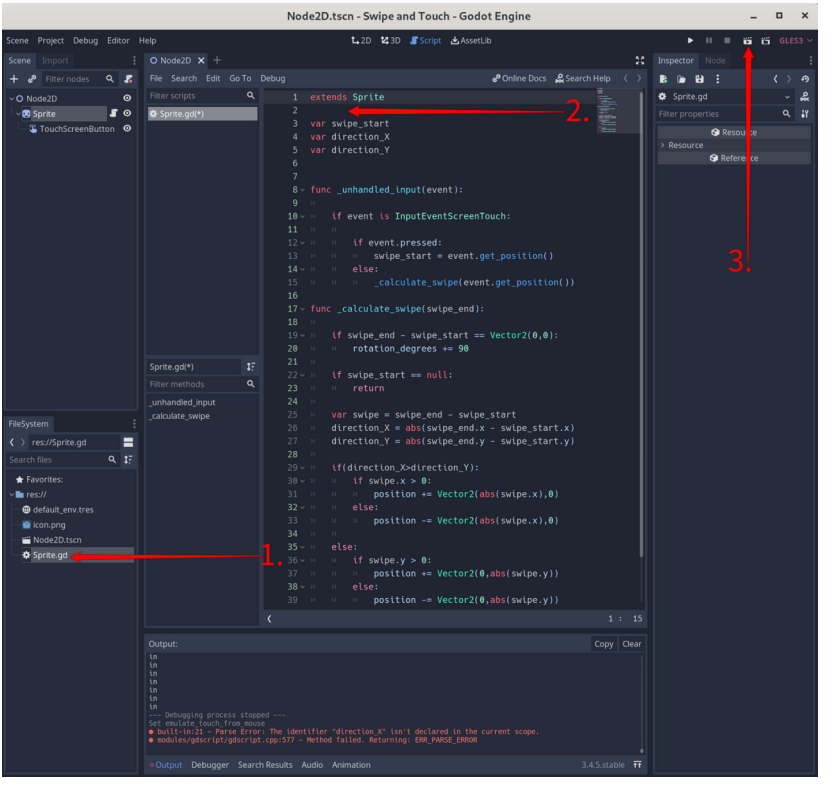
Step 18.

1. Click on the Script.gd script.

2. Type or Copy the Logic from below.

3. Play this Scene.

Example Output:



Codes:

```
extends Sprite

var swipe_start
var direction_X
var direction_Y


func _unhandled_input(event):

    if event is InputEventScreenTouch:

        if event.pressed:
            swipe_start = event.get_position()
        else:
            _calculate_swipe(event.get_position())

func _calculate_swipe(swipe_end):

    if swipe_end - swipe_start == Vector2(0,0):
        rotation_degrees += 90

    if swipe_start == null:
        return

    var swipe = swipe_end - swipe_start
    direction_X = abs(swipe_end.x - swipe_start.x)
    direction_Y = abs(swipe_end.y - swipe_start.y)

    if(direction_X>direction_Y):
        if swipe.x > 0:
            position += Vector2(abs(swipe.x),0)
        else:
            position -= Vector2(abs(swipe.x),0)

    else:
        if swipe.y > 0:
```

```
            position += Vector2(0,abs(swipe.y))
        else:
            position -= Vector2(0,abs(swipe.y))
```

Logics:

- There are three variables;

1) “swipe_start” records down the start location of the screen being pressed.

2) “direction_X” stores the distance of the X-coordinate start and end point when the user swipes.

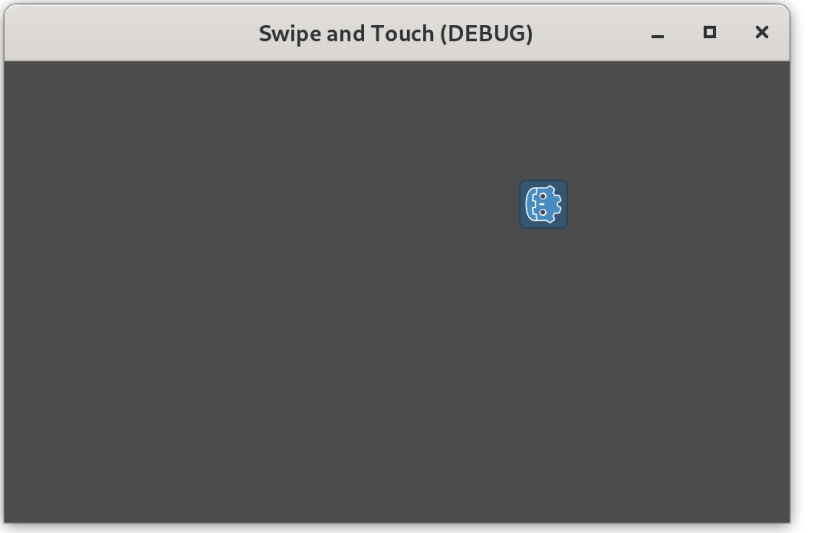
3) “direction_Y” stores the distance of the Y-coordinate start and end point when the user swipes.

- The function “_unhandled_input” checks if the input is from the Screen touch, Then it will record the start position if the event is a Screen Press. Else, it will be seen as a Press Released and trigger “_calculate_swipe” function.

- The function “_calculate_swipe” takes in the end point of the Press being released, and use the end point - start point to calculate the distance between the two point. Further, if there no different between two points, which means that they is no Swipe performed but tap. In this case, we will only rotate the Icon. Moreover, if “direction_X” is larger than “direction_Y”, it whether the user has swiped vertically or horizontally. Finally, it sets the position of the Sprite to the dragged distance.

Step 19. Try Single Click and rotate the Icon. You could also Press and Swipe the Icon anywhere you want :)

Example Output:



E.N.D

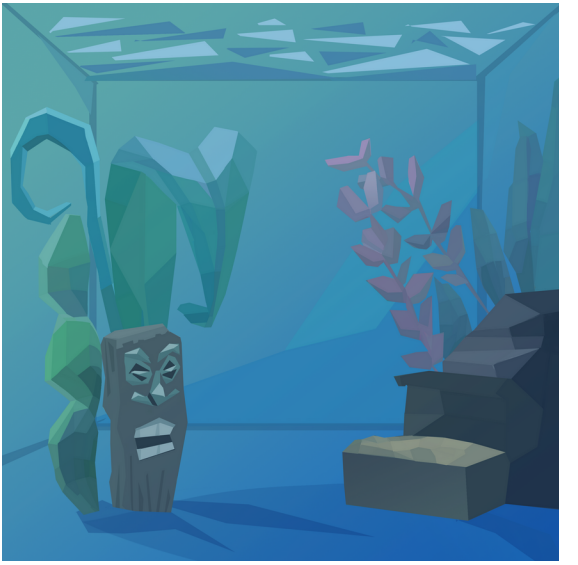
Tetris Game on Godot

Brief Tetris game Development

Things you will need

You will need Texture or artwork when creating this game. I will be showing you guys some examples.

1. Game Background.png



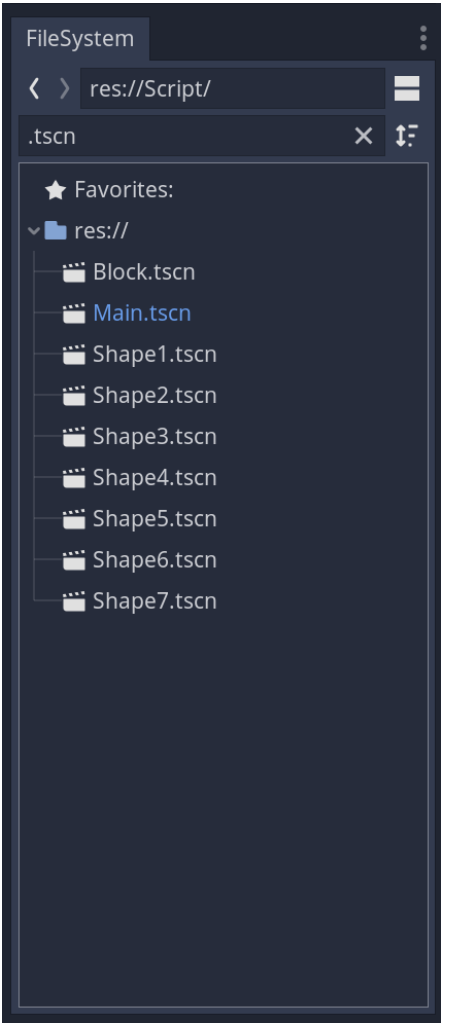
2. Game Field.png

3. Game Block.png

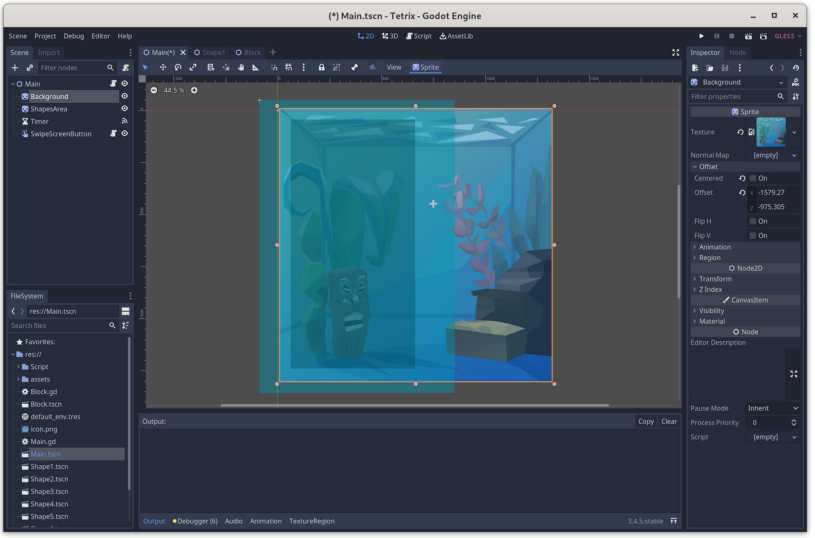
4.

Scene

- Scene Tree Overlook:

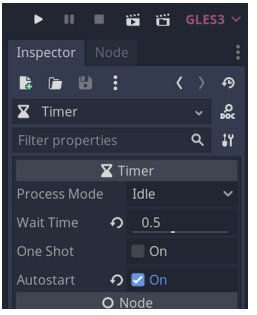


- Main Scene

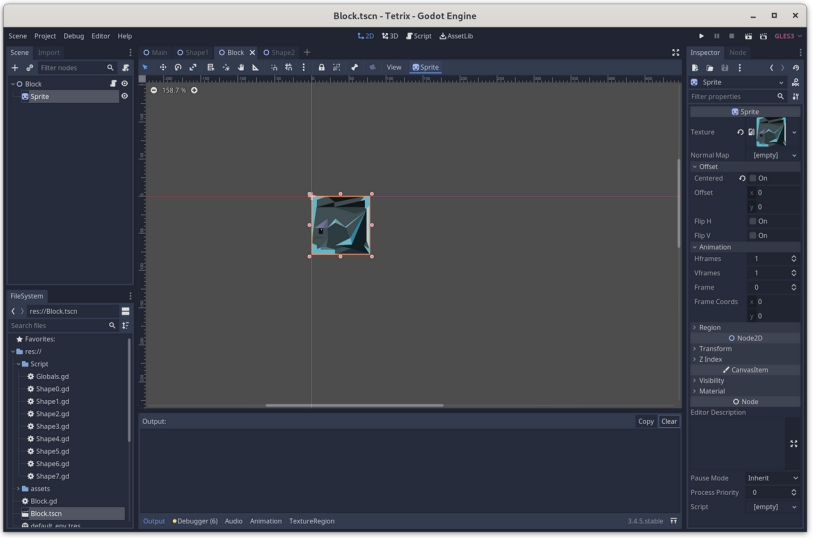


- Most of the visible elements are placed in the Main Scene, such as the background and the game field area. We also put the SwipeScreenButton in here which is the Child Class we created from the TouchScreenButton Parent Class. It's area covers the whole game field which allows us to capture all the input from the screen and pass to the it's class functions for further operation. E.g. Swipe, touch

- The Timer component in this Main Scene is to wait X seconds before letting the blocks start dropping from the top of the screen. This could be adjusted.

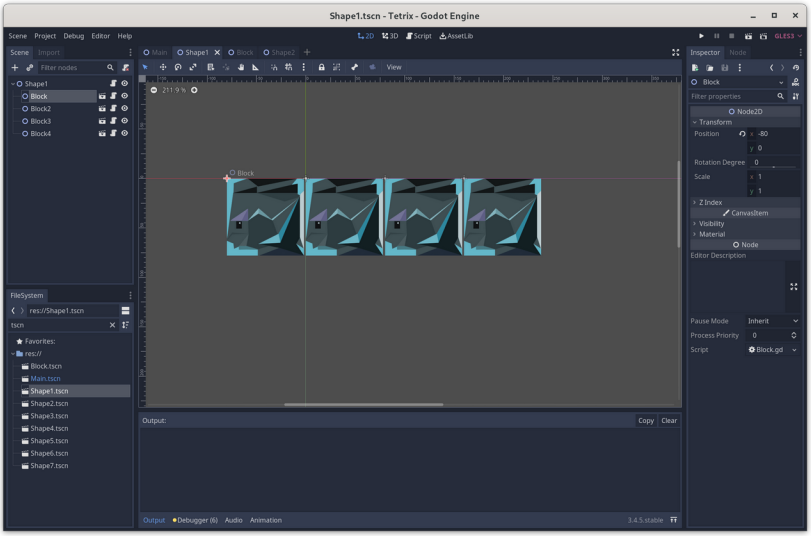


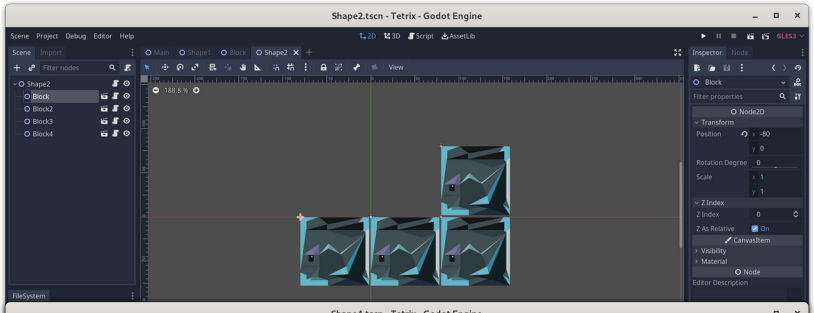
- Block Scene

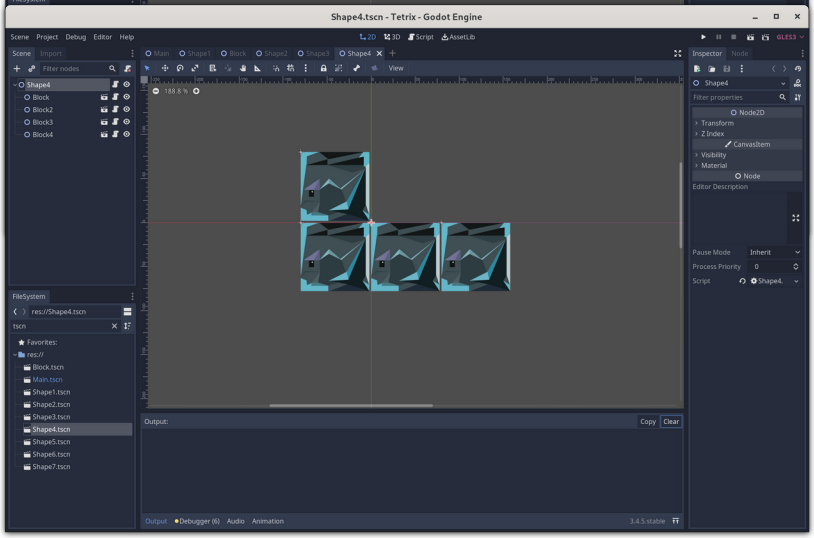


- The Block Scene only contains one Sprite with the texture = our block.png, it will be applied majorly on the dropping blocks. This block.png is scaled to 0.29 which will be fitting the game filed.

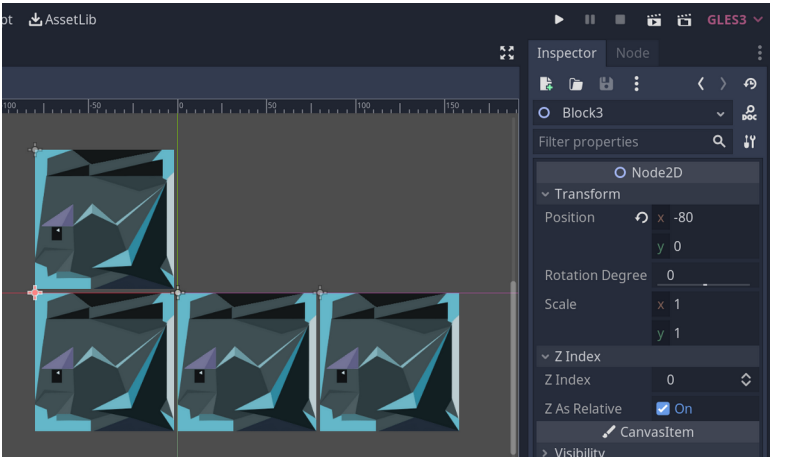
- Shape 1 - 7 Scenes (Showing 1 – 4 as examples)







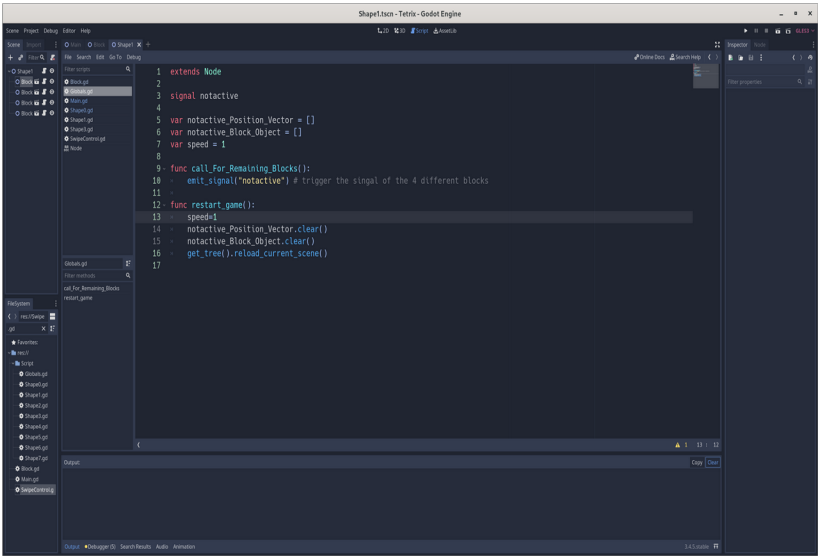
- There are in-total **7 Shapes** for this Tetris game, shown above are Shape 1 – 4 in the game.
- Each Shape Contain 4 Block Nodes, all of them have the same block and block size, the only different is x/y coordinate of specific blocks.
- In addition, we are rotating regarding the root "**Block2**", that's why we do not change the x/y coordinate of it but instead, changing "**Block1,3,4**" to fulfill the Tetris Block Shape.
- Simple Trick, if you want to change the x/y coordinate of certain block, just select the block and Go to the "**Transform**" Label in the Inspector screen.



Script

- Script Tree Overlook:

- Globals.gd



- Globals.gd is the location for us to store the global variables that we could be using in our later scripts. E.g. Block.gd, Main.gd

Codes:

```
extends Node

signal notactive

var notactive_Position_Vector = []
var notactive_Block_Object = []
var speed = 1

func call_For_Remaining_Blocks():
    emit_signal("notactive") # trigger the singal of the 4
different blocks
```

```
func restart_game():

    speed=1

    notactive_Position_Vector.clear()

    notactive_Block_Object.clear()

    get_tree().reload_current_scene()
```

Logics:

- We have Three variable and One Signal.

1) "notactive_Position_Vector" stores the X/Y Coordinates of the inactive blocks. As the game move forward, this array will be storing more and more blocks' position.

2) "notactive_Block_Object" stores the block object itself that are considered as inactive. As the game move forward, this array will be storing more and more block objects.

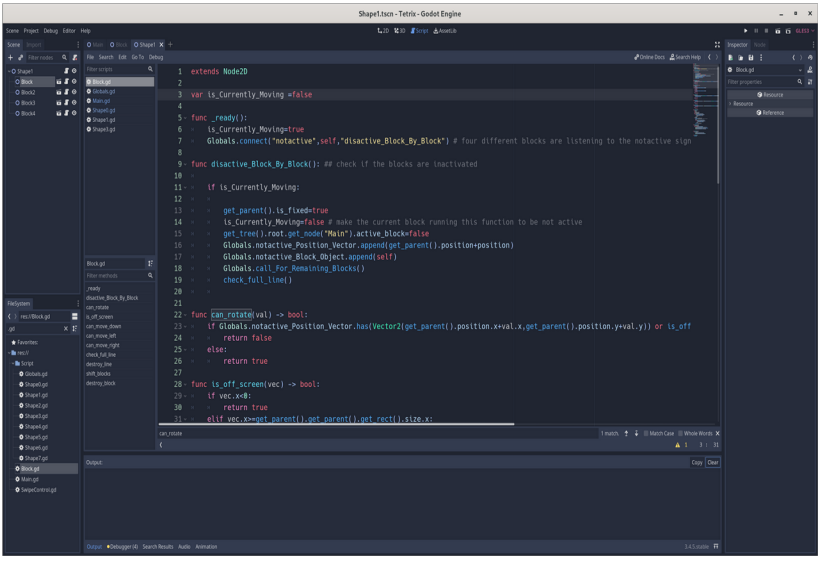
3) "speed" controls the blocks dropping speed.

4) "notactive" is a signal that will be called for event.

- The function "call_For_Remaining_Blocks" will be triggering the event listener the Block.gd script.

- The function "restart_game" resets all the variable we need for a new game.

- Block.gd



- Block.gd contains variable and method to control individual block for one specific Shape.

Codes:

```
extends Node2D

var is_Currently_Moving =false

func _ready():
   is_Currently_Moving=true
   Globals.connect("notactive",self,"disactive_Block_By_Block")
# four different blocks are listening to the notactive signal

func disactive_Block_By_Block(): ## check if the blocks are
inactivated

   if is_Currently_Moving:

      get_parent().is_fixed=true
```

```
      is_Currently_Moving=false # make the current block running
this function to be not active
      get_tree().root.get_node("Main").active_block=false

Globals.notactive_Position_Vector.append(get_parent().position+p
osition)
      Globals.notactive_Block_Object.append(self)
      Globals.call_For_Remaining_Blocks()
      check_full_line()


func can_rotate(val) -> bool:
   if
Globals.notactive_Position_Vector.has(Vector2(get_parent().posit
ion.x+val.x,get_parent().position.y+val.y)) or
is_off_screen(Vector2(get_parent().position.x+val.x,get_parent()
.position.y+val.y)):
      return false
   else:
      return true

func is_off_screen(vec) -> bool:
   if vec.x<0:
      return true
   elif vec.x>=get_parent().get_parent().get_rect().size.x:
      return true
   elif vec.y<0:
      return true
   elif vec.y>=get_parent().get_parent().get_rect().size.y:
      return true
   else:
      return false

func can_move_down():

   if
Globals.notactive_Position_Vector.has(Vector2(get_parent().posit
ion.x+position.x,get_parent().position.y+position.y+80)) or
```

```
get_parent().position.y+position.y==1520:
      disactive_Block_By_Block()
      return false
   else:
      return true

func can_move_left():
   if get_parent().position.x+position.x==0 or
(Globals.notactive_Position_Vector.has(Vector2(get_parent().posi
tion.x+position.x-80,get_parent().position.y+position.y))) or
not is_Currently_Moving:
      return false
   else:
      return true

func can_move_right():
   if get_parent().position.x+position.x==720 or
(Globals.notactive_Position_Vector.has(Vector2(get_parent().posi
tion.x+position.x+80,get_parent().position.y+position.y))) or
not is_Currently_Moving:
      return false
   else:
      return true

func check_full_line():

   var index=0
   var count=0
   var positions_to_erase=[]
   var blocks_to_shift=[]
   for i in Globals.notactive_Position_Vector:
      if i.y==get_parent().position.y+position.y:
         positions_to_erase.append(index)
         count+=1
      index+=1
   if count==10: # if 10 blocks in a row
      destroy_line(positions_to_erase) # destroy the row blocks
      index=0
```

```
      for i in Globals.notactive_Position_Vector: # i
representing the existing blocks in the array

         if i.y<get_parent().position.y+position.y:

            blocks_to_shift.append(index)
         index+=1

      shift_blocks(blocks_to_shift)

func destroy_line(indexes):
   var line_vals=indexes
   for i in range(line_vals.size()-1,-1,-1):

      Globals.notactive_Position_Vector.remove(line_vals[i])

Globals.notactive_Block_Object[line_vals[i]].destroy_block()
      Globals.notactive_Block_Object.remove(line_vals[i])


func shift_blocks(blocks):
   for i in blocks:
      Globals.notactive_Position_Vector[i].y+=80
      Globals.notactive_Block_Object[i].position.y+=80

func destroy_block():
   queue_free()
```

Logics:

- The variable “is_Currently_Moving” controls the moving state of each block.

- The function “_ready” will be loaded when the Scene starts, which will then set all the block moving state to be True and also make each block listen on the “notactive” event. If the event being triggered, the “disactive_Block_By_Block“ function will be called.

- The function “disactive_Block_By_Block” checks if the individual block is moving, if yes, it will change the current block running state to false. Also, it will append the X/Y position of every block for that Shape to the “notactive_Position_Vector”. And append the block object itself to “notactive_Block_Object”. After the recursive call and all the blocks for the Shape is not moving, “check_full_line” function will be called for checking if there are completed lines.

- The function “can_rotate” and “is_off_screen” worked together to checks if the the blocks can rotate to a different direction. They take the X/Y position of the rotation_matrix and adding up with the block current location. Finally, return if the Blocks are could be rotated.

- The function “can_move_down”, “can_move_left” and “can_move_right” helps for checking if the next moving block will be collide with the current fixed blocks in the screen. E.g. “can_move_right” checks if we add 80 to the current moving blocks, will it be the same as the existing blocks in “notactive_Position_Vector”, if so, it means the moving block cannot go further. And they we will call the “disactive_Block_By_Block” function to disable all moving blocks.

- The function “check_full_line” contains four variables including:

1) “index” holds the indexing Number of the array “notactive_Position_Vector”, helping us to know which block we should delete in the existing screen.

2) “count” aids in counting if any line has reach 10 blocks per row, if so, we will destroy that line by calling “destroy_line”

3) “positions_to_erase” is an array that store all the “index” position for the row we want to delte.

4) “blocks_to_shift” if the “count” reaches 10 and we are deleting that row, we will store the “index” of the blocks should move.

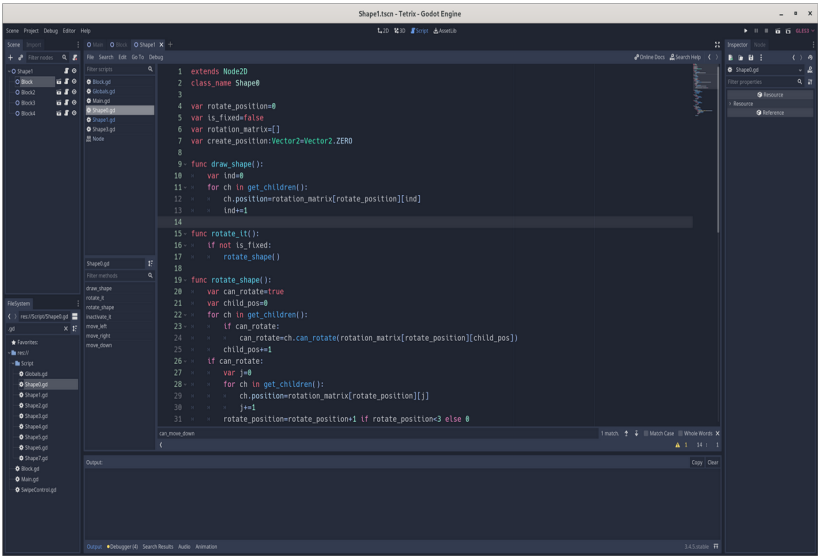
   The Flow here is just to check if the any row has 10 block in line, if so, we will destroy that line and also shift the blocks to the correct position.

- The function “destroy_line” takes in a variable “indexes” and by using “indexes”, it removes the completed row in the “notactive_Position_Vector” and “notactive_Block_Object”.

- The function “destroy_line” takes in variable “blocks” that contains of the ready to shift blocks’ index. And just modify their Y position.

- The function “destroy_block” is a system function, that delete a node at the end of the current frame.

- Shape0.gd



- Shape0.gd is programmed as the parent of the Shape1-7.gd, every Shape1-7 inherited the variables and functions from Shape0. The major logic for the Shape Control could be found here, E.g. Rotate.

Codes:

```
extends Node2D
class_name Shape0

var rotate_position=0
var is_fixed=false
var rotation_matrix=[]
var create_position:Vector2=Vector2.ZERO

func draw_shape():
    var ind=0
    for ch in get_children():
        ch.position=rotation_matrix[rotate_position][ind]
        ind+=1
```

```
func rotate_it():
    if not is_fixed:
        rotate_shape()

func rotate_shape():
    var can_rotate=true
    var child_pos=0
    for ch in get_children():
        if can_rotate:

can_rotate=ch.can_rotate(rotation_matrix[rotate_position]
[child_pos])
        child_pos+=1
    if can_rotate:
        var j=0
        for ch in get_children():
            ch.position=rotation_matrix[rotate_position][j]
            j+=1
        rotate_position=rotate_position+1 if rotate_position<3
else 0

func move_left():
    if not is_fixed:
        for ch in get_children():
            if not ch.can_move_left():
                return
        position.x-=80

func move_right():
    if not is_fixed:
        for ch in get_children():
            if not ch.can_move_right():
                return
        position.x+=80

func move_down():
    if not create_position:
```

```
        create_position=position
    if not is_fixed:
        for ch in get_children():
            if not ch.can_move_down():

                if create_position==position:
                    Globals.restart_game()

                return

        position.y+=80
```

Logics:

- There are four variables:

1) “rotate_position” controls the moving state of each block.

2) “is_fixed” controls the status of each block whether they are in a fixed position.

3) “rotation_matrix” contains all the rotating Shape for each Original Shape. This variable will be overwritten by Each Shape1-7.gd.

4) “create_position” stores the initial location when the new Shape is respawned.

- The function “draw_shape” will be called when any of the Shape got respawned. This function allows 4 blocks to

position themselves according to the “rotation_matrix”.

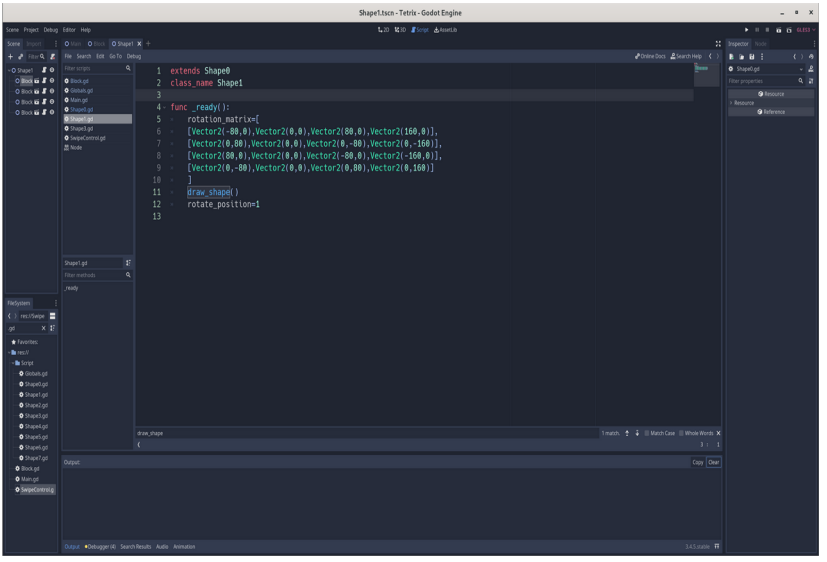
- The function “rotate_it” and “rotate_shape” worked together to check if it is feasible for the Shape to rotate. First “rotate_it” will check if current block is in the fixed position, if not it will trigger the “rotate_shape” method. Inside of the “rotate_shape” method. It will call the “can_rotate” method to see if rotation is doable. For more detail about “can_rotate”, please refer to Block.gd. If rotation is possible for the current Shape, we will set the X/Y position of every moving blocks to be the X/Y position in the “rotation_matrix”. Finally, change the “rotate_position” to make sure we are rotated.

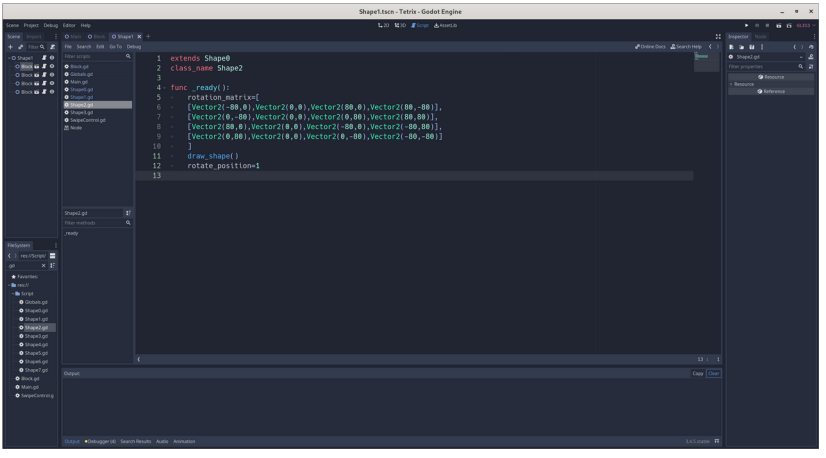
- The function “move_left” and “move_right” first check if the current blocks are in a fixed location, if not, then check with each block using “can_move_left” and “can_move_right” see if they are capable to move to left/right. If yes, add/minus the whole Shape Node’s y-position by 80.

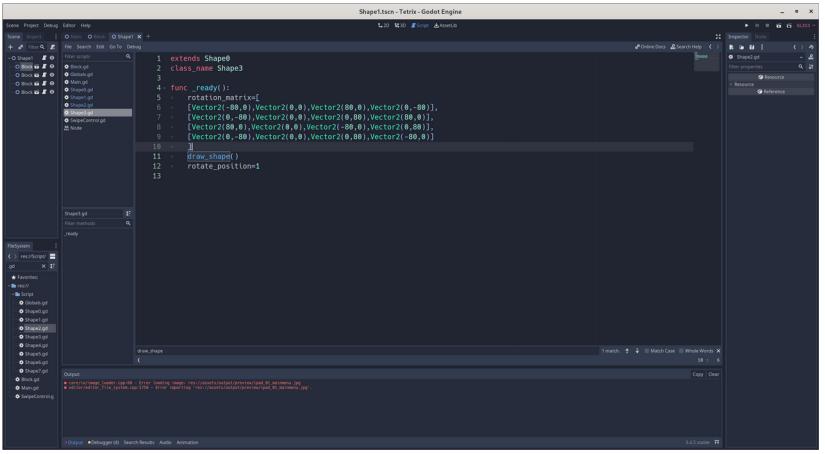
- The function “move_down” is a little different from the above two directional control function. It helps to configure the “create_position” of the Shape Node when it is created. Further, if the blocks cannot move down anymore and our current position has reached the create_position. This implies the user has failed and the game will be restart by the “restart_game” function in the Globals.gd. On the other hand, if the blocks can still go down, we will add the Y-position of the blocks by 80.

- Shape1-7.gd

- Screen Capture only showed Shape1-3.gd as examples, since they are almost identical to each other







- Block1-7.gd contains rotation_matrix that we predefined when when Block1-7.gd is initialized. At the same time, they will trigger the draw_shape function from their parent class from Shape0.gd and define their next rotate_position to the second one in their matrix.

Codes For Block1.gd:

```
extends Shape0
class_name Shape1

func _ready():
  rotation_matrix=[
  [Vector2(-80,0),Vector2(0,0),Vector2(80,0),Vector2(160,0)],
  [Vector2(0,80),Vector2(0,0),Vector2(0,-80),Vector2(0,-160)],
  [Vector2(80,0),Vector2(0,0),Vector2(-80,0),Vector2(-160,0)],
  [Vector2(0,-80),Vector2(0,0),Vector2(0,80),Vector2(0,160)]
  ]
  draw_shape()
  rotate_position=1
```

Codes For Block2.gd:

```
extends Shape0
class_name Shape2

func _ready():
    rotation_matrix=[
    [Vector2(-80,0),Vector2(0,0),Vector2(80,0),Vector2(80,-80)],
    [Vector2(0,-80),Vector2(0,0),Vector2(0,80),Vector2(80,80)],
    [Vector2(80,0),Vector2(0,0),Vector2(-80,0),Vector2(-80,80)],
    [Vector2(0,80),Vector2(0,0),Vector2(0,-80),Vector2(-80,-80)]
    ]
    draw_shape()
    rotate_position=1
```

Codes For Block3.gd:

```
extends Shape0
class_name Shape3

func _ready():
    rotation_matrix=[
    [Vector2(-80,0),Vector2(0,0),Vector2(80,0),Vector2(0,-80)],
    [Vector2(0,-80),Vector2(0,0),Vector2(0,80),Vector2(80,0)],
    [Vector2(80,0),Vector2(0,0),Vector2(-80,0),Vector2(0,80)],
    [Vector2(0,-80),Vector2(0,0),Vector2(0,80),Vector2(-80,0)]
    ]
    draw_shape()
    rotate_position=1
```

Codes For Block4.gd:

```
extends Shape0
class_name Shape4

func _ready():
    rotation_matrix=[
    [Vector2(80,0),Vector2(0,0),Vector2(-80,0),Vector2(-80,-
80)],
    [Vector2(0,80),Vector2(0,0),Vector2(0,-80),Vector2(80,-80)],
    [Vector2(-80,0),Vector2(0,0),Vector2(80,0),Vector2(80,80)],
    [Vector2(0,-80),Vector2(0,0),Vector2(0,80),Vector2(-80,80)]
    ]
    draw_shape()
    rotate_position=1
```

Codes For Block5.gd:

```
extends Shape0
class_name Shape5

func _ready():
    rotation_matrix=[
    [Vector2(-80,0),Vector2(0,0),Vector2(-80,80),Vector2(0,80)],
    [Vector2(-80,0),Vector2(0,0),Vector2(-80,80),Vector2(0,80)],
    [Vector2(-80,0),Vector2(0,0),Vector2(-80,80),Vector2(0,80)],
    [Vector2(-80,0),Vector2(0,0),Vector2(-80,80),Vector2(0,80)]
    ]
    draw_shape()
    rotate_position=1
```

Codes For Block6.gd:

```
extends Shape0
class_name Shape6

func _ready():
    rotation_matrix=[
    [Vector2(80,0),Vector2(0,0),Vector2(0,-80),Vector2(-80,-
80)],
    [Vector2(0,80),Vector2(0,0),Vector2(80,0),Vector2(80,-80)],
    [Vector2(80,0),Vector2(0,0),Vector2(0,-80),Vector2(-80,-
80)],
    [Vector2(0,80),Vector2(0,0),Vector2(80,0),Vector2(80,-80)]
    ]
    draw_shape()
    rotate_position=1
```

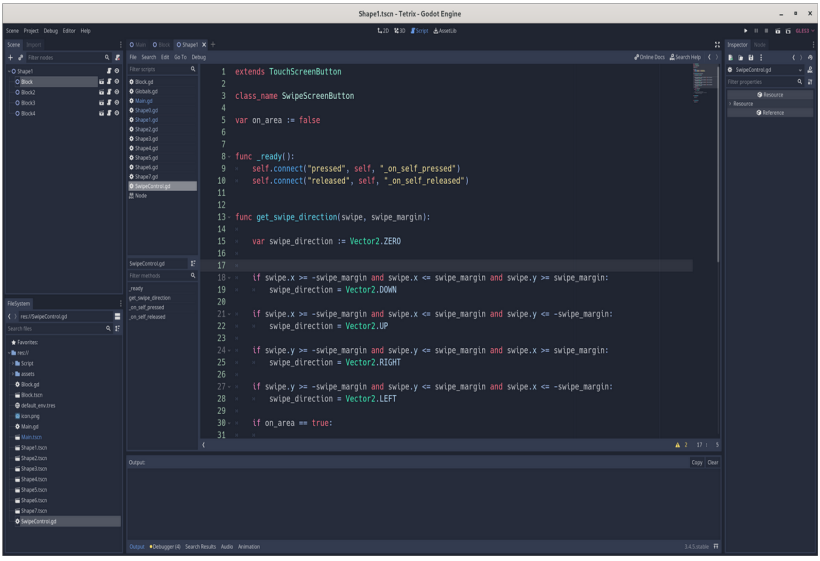
Codes For Block7.gd:

```
extends Shape0
class_name Shape7

func _ready():
    rotation_matrix=[
    [Vector2(-80,0),Vector2(0,0),Vector2(0,-80),Vector2(80,-
80)],
    [Vector2(0,-80),Vector2(0,0),Vector2(80,0),Vector2(80,80)],
    [Vector2(80,0),Vector2(0,0),Vector2(0,80),Vector2(-80,80)],
    [Vector2(0,80),Vector2(0,0),Vector2(-80,0),Vector2(-80,-80)]
    ]
    draw_shape()
    rotate_position=1
```

Logics:

- The function “_ready” will be pre-loaded when any Shape1-7 is created. Then inside “_ready”, it sets the empty “rotation_matrix“ in Shape0 to be a pre-set array (rotational shapes in X&Y position). Then call the“draw_shape” function to position the Shape to the first X&Y position in the “rotation_matrix“. At last, this function set the next rotate position to 1 and we could check whether rotation could be performed.

- SwipeControl.gd



- SwipeControl.gd extends the TouchScreenButton Class and inherited the variables and function in that class. Is is made for the screen control, E.g. Swipe,touch.

Codes:

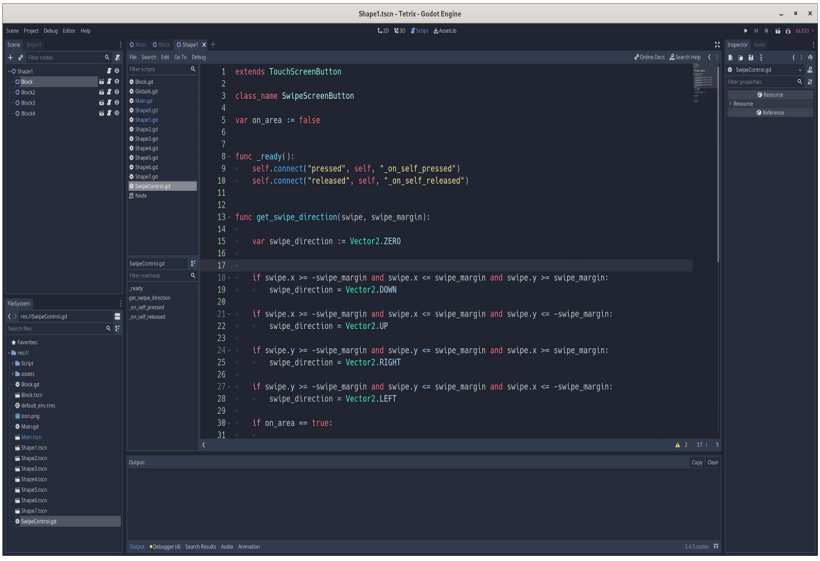
```
extends TouchScreenButton

class_name SwipeScreenButton

var on_area := false

func _ready():
    self.connect("pressed", self, "_on_self_pressed")
    self.connect("released", self, "_on_self_released")


func get_swipe_direction(swipe, swipe_margin):
```

```
    var swipe_direction := Vector2.ZERO


    if swipe.x >= -swipe_margin and swipe.x <= swipe_margin and
swipe.y >= swipe_margin:
        swipe_direction = Vector2.DOWN

    if swipe.x >= -swipe_margin and swipe.x <= swipe_margin and
swipe.y <= -swipe_margin:
        swipe_direction = Vector2.UP

    if swipe.y >= -swipe_margin and swipe.y <= swipe_margin and
swipe.x >= swipe_margin:
        swipe_direction = Vector2.RIGHT

    if swipe.y >= -swipe_margin and swipe.y <= swipe_margin and
swipe.x <= -swipe_margin:
        swipe_direction = Vector2.LEFT

    if on_area == true:

        return swipe_direction
    else:
        return Vector2.ZERO


func _on_self_pressed():

    on_area = true



func _on_self_released():

    on_area = false
```

Logics:

- The variable “on_area” controls the state of the Press control whether it is pressed or not. It is like a lock.
- The function “_ready” listen on the “pressed” and “released” event which will trigger the “_on_self_pressed” & “_on_self_released” function.
- The function “get_swipe_direction” takes in the “swipe” and “swipe_margin” do some simple check to see if the user is swiping up/down/left/right. At last, if “on_area” is True, it will return “swipe_direction” we created for storing Vector2.Directions.

- Main.gd



- Main.gd is the Core Script of this game. It controls the Swipe,Touch,Rotation,Respawn.

Codes:

```
extends Node2D

onready var shape1=preload("res://Shape1.tscn")
onready var shape2=preload("res://Shape2.tscn")
onready var shape3=preload("res://Shape3.tscn")
onready var shape4=preload("res://Shape4.tscn")
onready var shape5=preload("res://Shape5.tscn")
onready var shape6=preload("res://Shape6.tscn")
onready var shape7=preload("res://Shape7.tscn")

# check the screen control
onready var Swipe = $SwipeScreenButton

var shapes=[]
```

```
var currentMovingShape
var active_block=false
var rnd=RandomNumberGenerator.new()
var num:int=-1
var next_num:int=0

func _ready():
    shapes=[shape1,shape2,shape3,shape4,shape5,shape6,shape7]
    rnd.randomize()

func _on_Timer_timeout():
    $Timer.wait_time=Globals.speed

    if not active_block: ## Generate new blocks if there is no
more active block
        num=rnd.randi()%7 if num==-1 else next_num
        next_num=rnd.randi()%7

        currentMovingShape=shapes[num].instance()
        $ShapesArea.add_child(currentMovingShape)

        currentMovingShape.position=Vector2(320,80)
        active_block=true
    else:
        move_down()

func move_left():
    if active_block:
        currentMovingShape.move_left()

func move_right():
    if active_block:
        currentMovingShape.move_right()

func move_down():
    if active_block:

        currentMovingShape.move_down()
```

```
        $Timer.start()


var releaseFromDrag = false

func _input(event):

    if currentMovingShape:

        if event is InputEventScreenDrag:


                if Swipe.get_swipe_direction(event.relative,4)
== Vector2.RIGHT:
                    move_right()
                    releaseFromDrag = true
                elif Swipe.get_swipe_direction(event.relative,4)
== Vector2.LEFT:
                    move_left()
                    releaseFromDrag = true
                elif
Swipe.get_swipe_direction(event.relative,15) == Vector2.DOWN:
                    for i in 50:
                        move_down()
                    releaseFromDrag = true

        if event is InputEventScreenTouch:
            if not event.is_pressed() and releaseFromDrag ==
false:
                currentMovingShape.rotate_it()
            releaseFromDrag = false
```

Logics:

- The variable “shape1-7” will be pre-loaded before the game start, we need these seven Shape Scenes to be put into our “shapes” array.

- The variable “Swipe” is loaded as a instance from SwipeScreenButton.gd for our touch screen control.

- The variable “shapes” is an array that stores the Seven Scenes.

- The variable “active_block” helps in checking if the current block is moving.

- The variable “rnd” created as a Class Object from the “RandomNumberGenerator”, just a tool to generate random numbers.

- The variable “num” responsible for storing a randomize number and make it as the applied for selecting current Shape. For instance, num = 1, select Shape 1.

- The variable “next_num” responsible for storing a randomize number and make it as the applied for selecting the next Shape. For instance, num = 5, next selected Shape 5.

- The function “_ready” will be start when Main Scene starts running, all the pre-loaded “Shape1-7” will be added to the “shapes” array. Then we will start randomizing the “rnd” variable.

- The function “_on_Timer_timeout” will be started only when our timer is time out. As mentioned in the Scene Topic of this tutorial, the timer could be configured by yourself with anytime you prefer.
  - Moreover,this function is triggered, we will set the wait_time of this timer to be our speed which we can

adjust during the game.
- If we have a short speed, we will get a shorter dropping time. But remember that we cannot have a negative speed, or else it can not keep lowering the timer's wait time.
- If the "active_block" is false, The current Shape and the next Shape's number will be generated and stored in the "num" and "next_num"
- The "currentMovingShape" will be assigned by the "shapes" with the "num". Then added to the game field. After that, the position of that Shape will be set to the position (320,80). It will also be used in Shape0.gd for "create_position".
- We then set the "active_block" to true which disable the new blocks from Re-generating.
- If the "active_block" is true, we will perform the "move_down" function to make the blocks go down by 1 frame.

- The function "move_left", "move_right" first check if the "active_block" is in the true state. If so, they will call the "move_left" function inside the "currentMovingShape" object.
  P.S. The move left function are different if it is the function for that specific object.
  E.g. ClassA.move_left() is different from ClassB.move_left(). Even they have the same function name, the content inside is different expect when they are parent&child Class.

- The variable "releaseFromDrag" and the function "_input" worked together for performing Screen Swipe & Touch. "releaseFromDrag" serves as a lock to stop Swipe and Touch got Triggered at the same time. This mean we can perform Swipe and Touch Individually.
  "_input" listen on any events on the SwipeScreenButton area. Each input would be passed as "event" to this

"_input" function.

- If the "event" is InputEventScreenDrag, we will pass the event relative position and the swipe_margin to "Swipe.get_swipe_direction". If this function return Vector2.LEFT/RIGHT/DOWN, move_left/move_right/move_down will be called accordingly. After called the move function, "releaseFromDrag" will be set to true and this will block the "currentMovingShape.rotate_it" below.

- If the "event" is InputEventScreenTouch, we will check if the press is released and is the "releaseFromDrag" false or not. If it is true instead of false, it means that this touch is release from Swipe that we don't have to rotate the direction. However, if the "releaseFromDrag" is true, we will know it is just a normal Screen Touch, we will then perform rotation on the current Shape by calling "currentMovingShape.rotate_it".

E.N.D