How to Make a Strategy Game in Godot – Part 2

Last Updated November 10, 2023 by Daniel Buckley

Table of contents

Introduction

Welcome back to Part 2 of creating a strategy game in the Godot game engine!

In Part 1, we began showing you how to build your first strategy game in Godot by setting up highlight-able tiles, buildings, and the basic UI which will tell the players crucial information about their resource management. All in all, we have a great foundation to work with so we can finish our project and add it to our portfolio!

Of course, with this being Part 2, there is still more go. We need to finish up our map system, set up our UI properly, give ourselves the ability to place buildings, and even implement the overall turn-based gameplay flow. So, if you're prepared, and let's finish this resource management game and become master Godot developers!

Project Files

In this tutorial, we'll be using some sprites from the kenney.nl website (an open domain game asset website) and fonts from Google Fonts. You can of course choose to use your own assets, but we'll be designing the game around these:

- Download the sprite and font assets we'll be using for this tutorial here.
- Download the complete strategy game Godot project here.

Did you come across any errors in this tutorial? Please let us know by completing this form and we'll look into it!

Finishing the Map Script

To begin, let's go back to our **Map** script. The **place_building** function gets called when we want to place down a building on a tile.

```
# places down a building on the map
func place_building (tile, texture):

    tilesWithBuildings.append(tile)
    tile.place_building(texture)

    disable_tile_highlights()
```

Finally, the **_ready** function gets called when the node is initialized. Here, we want to get all of the tiles in the "Tiles" group and setup the initial base building.

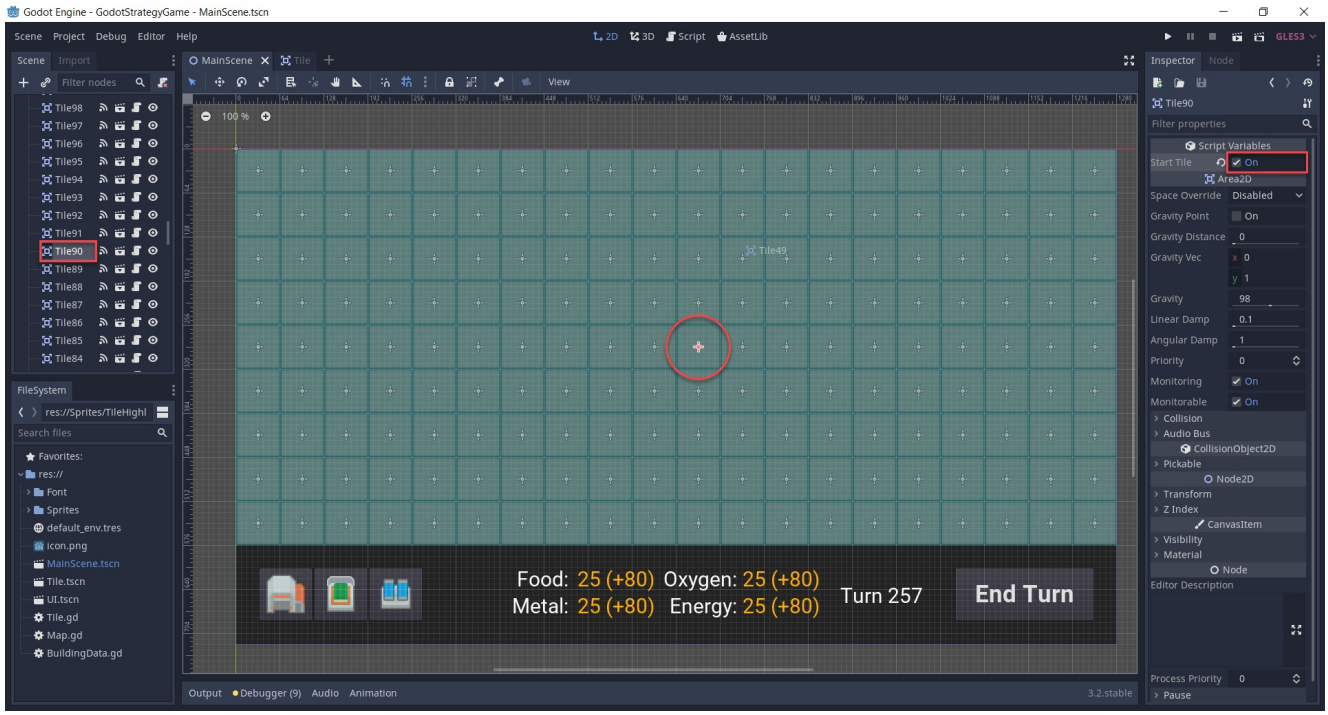
```
func _ready ():

```

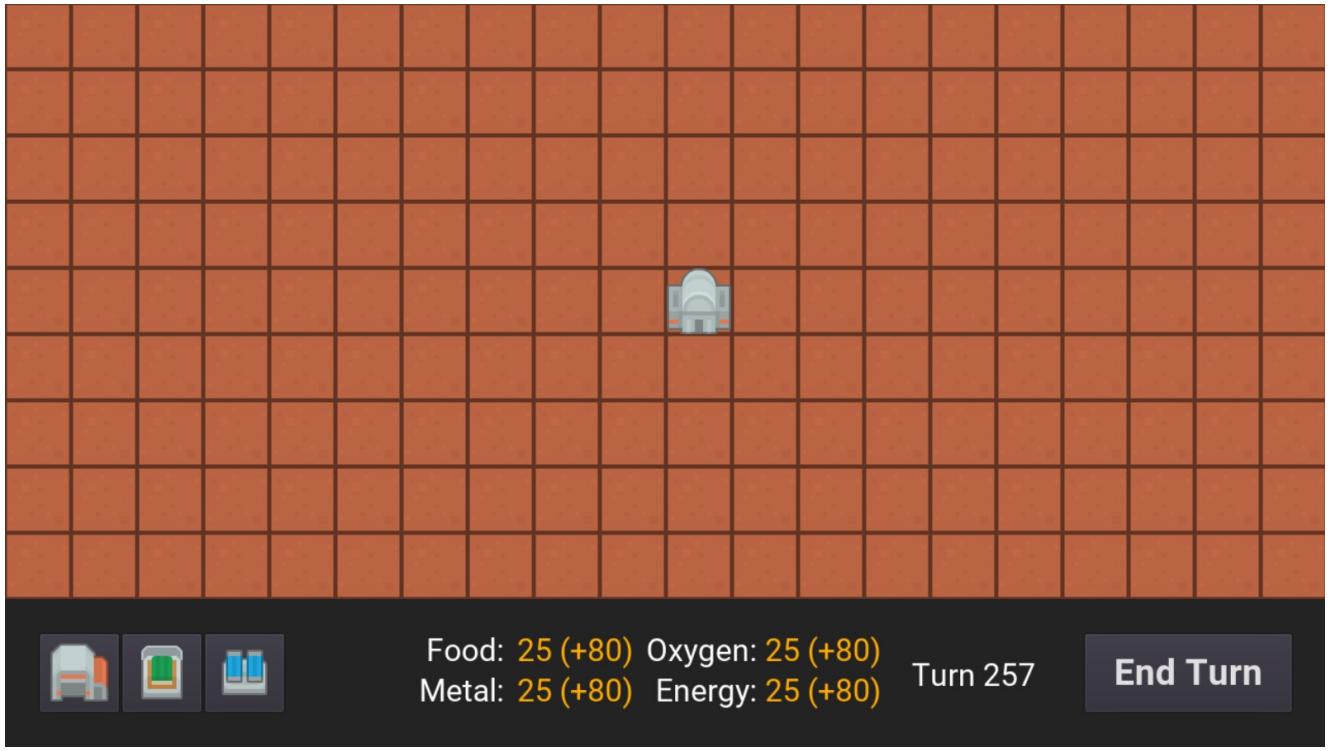
```
    # when we're initialized, get all of the tiles
    allTiles = get_tree().get_nodes_in_group("Tiles")

    # find the start tile and place the Base building
    for x in range(allTiles.size()):
        if allTiles[x].startTile == true:
            place_building(allTiles[x], BuildingData.base.iconI
```

Back in the **MainScene**, let's select the center tile and enable **Start Tile**.



Now if we press play, you should see that the center tile has a Base building on it.



GameManager Script

The **GameManager** script is what's going to manage our resources and states. Go to the **MainScene** and select the **MainScene** node. Create a new script attached to it called **GameManager**. We can start with our variables.

```
1.  # current amount of each resource we have
2.  var curFood : int = 0
3.  var curMetal : int = 0
4.  var curOxygen : int = 0
5.  var curEnergy : int = 0
6.
7.  # amount of each resource we get each turn
8.  var foodPerTurn : int = 0
9.  var metalPerTurn : int = 0
10. var oxygenPerTurn : int = 0
11. var energyPerTurn : int = 0
12.
13. var curTurn : int = 1
14.
15. # are we currently placing down a building?
```

```
var currentlyPlacingBuilding : bool = false

# type of building we're currently placing
var buildingToPlace : int

# components
onready var ui : Node = get_node("UI")
onready var map : Node = get_node("Tiles")
```

The **on_select_building** function gets called when we press one of the three building UI buttons. This will be hooked up later on when we create the **UI** script.

```
# called when we've selected a building to place
func on_select_building (buildingType):

    currentlyPlacingBuilding = true
    buildingToPlace = buildingType

    # highlight the tiles we can place a building on
    map.highlight_available_tiles()
```

The **add_to_resource_per_turn** function adds the given *amount* to the given *resource* per turn.

```
# adds an amount to a certain resource per turn
func add_to_resource_per_turn (resource, amount):

    # resource 0 means none, so return
    if resource == 0:
        return
    elif resource == 1:
        foodPerTurn += amount
    elif resource == 2:
        metalPerTurn += amount
    elif resource == 3:
        oxygenPerTurn += amount
    elif resource == 4:
        energyPerTurn += amount
```

The **place_building** function will be called when we place down a tile on the grid.

```
# called when we place a building down on the grid
func place_building (tileToPlaceOn):

    currentlyPlacingBuilding = false

    var texture : Texture

    # are we placing down a Mine?
    if buildingToPlace == 1:
        texture = BuildingData.mine.iconTexture

        add_to_resource_per_turn(BuildingData.mine.prodResource
        add_to_resource_per_turn(BuildingData.mine.upkeepResour

    # are we placing down a Greenhouse?
    if buildingToPlace == 2:
        texture = BuildingData.greenhouse.iconTexture

        add_to_resource_per_turn(BuildingData.greenhouse.prodRe
        add_to_resource_per_turn(BuildingData.greenhouse.upkeep

    # are we placing down a Solar Panel?
    if buildingToPlace == 3:
        texture = BuildingData.solarpanel.iconTexture

        add_to_resource_per_turn(BuildingData.solarpanel.prodRe
        add_to_resource_per_turn(BuildingData.solarpanel.upkeep

    # place the building on the map
    map.place_building(tileToPlaceOn, texture)
```

Finally, we have the **end_turn** function which gets called when we press the end turn button.

```
# called when the player ends the turn
func end_turn ():

```

```
    # update our current resource amounts
    curFood += foodPerTurn
    curMetal += metalPerTurn
    curOxygen += oxygenPerTurn
    curEnergy += energyPerTurn

    # increase current turn
    curTurn += 1
```

Okay so we've got our GameManager class all setup but there's no real way for it to function. In order to connect everything together, we need to create a **UI** script.

UI Script

In the **UI** scene, select the UI node and create a new script called **UI**. Let's start with our variables.

```
# container holding the building buttons
onready var buildingButtons : Node = get_node("BuildingButtons"

# text displaying the food and metal resources
onready var foodMetalText : Label = get_node("FoodMetalText")

# text displaying the oxygen and energy resources
onready var oxygenEnergyText : Label = get_node("OxygenEnergyTe

# text showing our current turn
onready var curTurnText : Label = get_node("TurnText")

# game manager object in order to access those functions and va
onready var gameManager : Node = get_node("/root/MainScene")
```

First, we have the **on_end_turn** function. This gets called when a turn is over, so we're going to reset the UI.

```
# called when a turn is over - resets the UI
func on_end_turn ():

```

```
    # updates the cur turn text and enable the building buttons
    curTurnText.text = "Turn: " + str(gameManager.curTurn)
    buildingButtons.visible = true
```

The we have the **update_resource_text** function which updates the two resource labels to show the player's current resource values.

```
# updates the resource text to show the current values
func update_resource_text ():

    # set the food and metal text
    var foodMetal = ""

    # sets the text, e.g. "13 (+5)"
    foodMetal += str(gameManager.curFood) + " (" + ("+" if game
    foodMetal += "\n"
    foodMetal += str(gameManager.curMetal) + " (" + ("+" if gam

    foodMetalText.text = foodMetal

    # set the oxygen and energy text
    var oxygenEnergy = ""

    # set the text, e.g. "13 (+5)"
    oxygenEnergy += str(gameManager.curOxygen) + " (" + ("+" if
    oxygenEnergy += "\n"
    oxygenEnergy += str(gameManager.curEnergy) + " (" + ("+" if

    oxygenEnergyText.text = oxygenEnergy
```

Now we need to connect the buttons. In the **UI** scene, do the following for the EndTurnButton, MineButton, GreenhouseButton and SolarPanelButton...

1. Select the button node
2. Double click the **pressed** signal (called when we press the button)
3. Connect that to the UI script

So back in our script, we'll have 4 new functions. Let's start with the three building buttons.

```
# called when the Mine building button is pressed
func _on_MineButton_pressed ():

    buildingButtons.visible = false
    gameManager.on_select_building(1)

# called when the Greenhouse building button is pressed
func _on_GreenhouseButton_pressed ():

    buildingbuttons.visible = false
    gameManager.on_select_building(2)

# called when the Solar Panel building button is pressed
func _on_SolarPanelButton_pressed

    buildingButtons.visible = false
    gameManager.on_select_building(3)
```

Then we have the end turn button function.

```
# called when the "End Turn" button is pressed
func _on_EndTurnButton_pressed ():

    gameManager.end_turn()
```

Connecting Everything Together

Now that we have our UI script, let's go back to the **Tile** script and fill in the **_on_Tile_input_event** function.

```
# called when an input event takes place on the tile
func _on_Tile_input_event (viewport, event, shape_idx):

    # did we click on this tile with our mouse?
    if event is InputEventMouseButton and event.pressed:
        var gameManager = get_node("/root/MainScene")

        # if we can place a building down on this tile, then do
        if gameManager.currentlyPlacingBuilding and canPlaceBui
            gameManager.place_building(self)
```

Next, let's hop into the **GameManager** script and create the **_ready** function. Here, we're going to initialize the UI.

```
func _ready ():

    # updates the UI when the game starts
    ui.update_resource_text()
    ui.on_end_turn()
```

At the end of the **end_turn** function, let's also update the UI.

```
# update the UI
ui.update_resource_text()
ui.on_end_turn()
```

Finally, at the bottom of the **place_building** function, we can update the resource text UI.

```
# update the UI to show changes immediately
ui.update_resource_text()
```

Now we can press play and test out the game!

Conclusion

Congratulations on completing the tutorial!

You just created a 2D, turn-based strategy game in Godot. Through this journey, we've covered a wide array of topics, from setting up objects that give and take resources, to creating a tile-based map that provides visual clues about where buildings can be placed. Further, with turn-based gameplay mechanics also introduced, we've tackled a key component for many other sorts of strategy games as well!

From here, you can expand upon what you've learned to add more systems, work on existing ones we touched on here, or even start a new strategy game project with Godot. Regardless, thank you very much for following along with the tutorial, and we wish you the best of luck with your future Godot games.

Continue Learning

A Guide to Third-Person Controllers

Free eBook – Godot Game Development for Beginners

Create a Top-Down 2D Character

How to Create an FPS Player and Camera

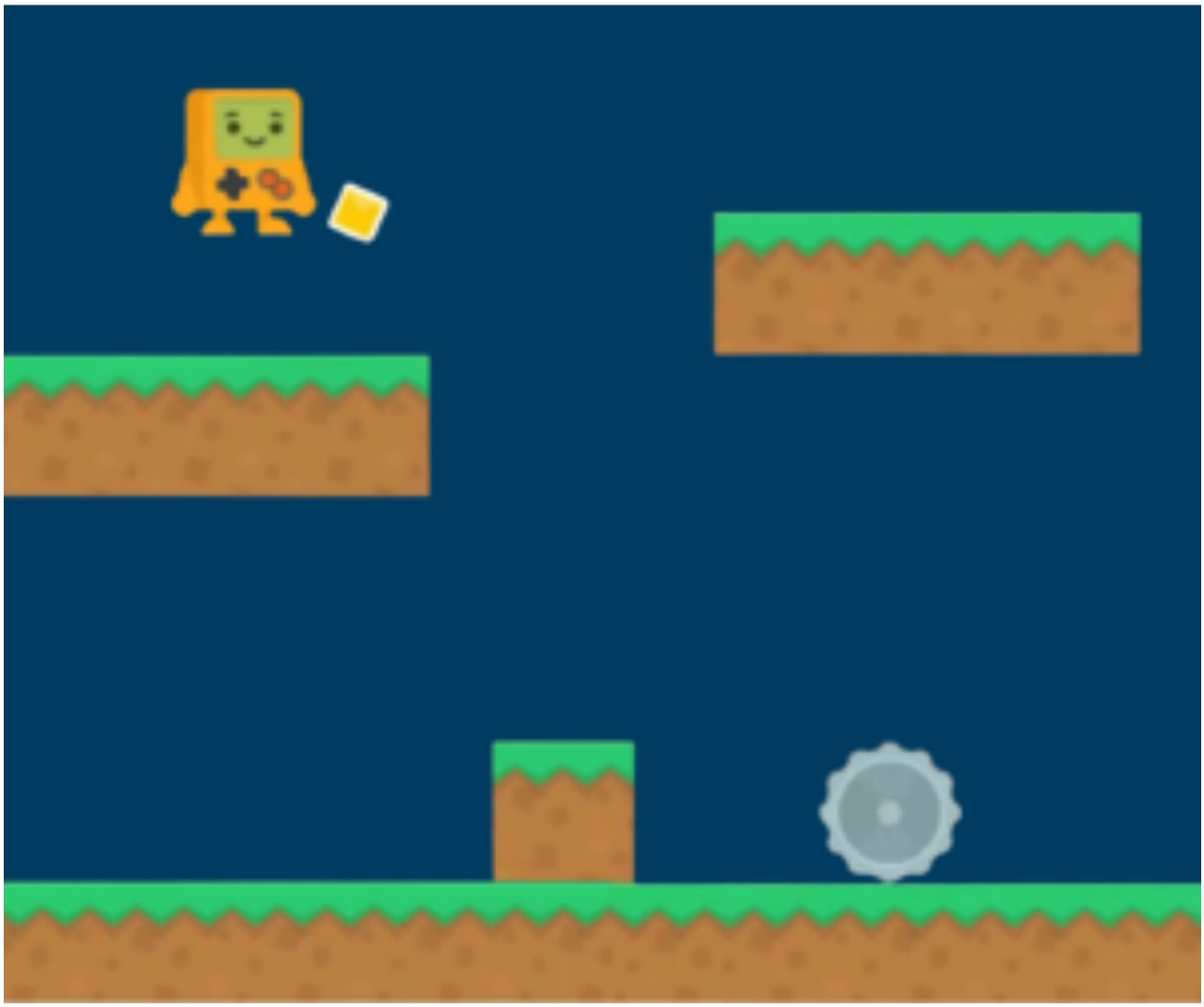
How to Track Players with Godot's Camera

An Introduction to the Godot Editor

Godot, Godot 3, Uncategorized

How to Make a Strategy Game in Godot – Part 1

Create 2D Lights with Unity's Universal Render Pipeline



How to Code a Game

Learn Coding

Python Programming

Game Development

Game Design

Game Engines

Unity Journey

What is Unity?

Platformer in Unity

RPG in Unity

Multiplayer Game

Mobile Game in Unity

Procedural Generation

Godot Journey

What is Godot?

Your First Godot Game

RPG in Godot

Survival Game in Godot

Strategy Game in Godot

Collision Detection

Python Journey

- What is Python?
- How to Learn Python?
- Python Basics
- Python Turtle
- Python Loops
- Pygame

Career Advice
- Coding Education
- Coding Jobs
- Zenva Success Stories

Featured Content

Godot
- GDScript
- Godot 3
- Godot 4

Metaverse (VR & AR)

Other Game Engines

- BabylonJS
- Blender
- Game Design
- Phaser
  - Phaser 2
  - Phaser 3
- Roblox
  - Lua

Python

- Computer Vision
- Data Science
- Machine Learning
- Pygame
- Python Basics

Software Development

- Android Development
- C#
- C++
- iOS Development
- Java

Uncategorized

Unity

- 2D
- 3D
- Intermediate

- Multiplayer
- Storytelling
- Unreal Engine
- Web Development
  - Frontend Development
    - CSS & CSS Frameworks
    - HTML
    - JavaScript
    - React
  - Server-Side Development
  - Web Development in Spanish

Mini-Degrees™

- Augmented Reality
- Godot
- Machine Learning
- Phaser
- Python
- RPGs
- Unity
- Unreal Engine
- Virtual Reality

Zenva

Course Catalog

Free Courses

Success Stories

Bulk Purchases

Help Center

Terms and Conditions

Privacy Policy

Our Network

Zenva Academy

Zenva Schools

GameDev Academy

Zenva Pty Ltd

138 Juliette Street
Greenslopes, QLD, 4120
Australia
ABN 83 606 402 199