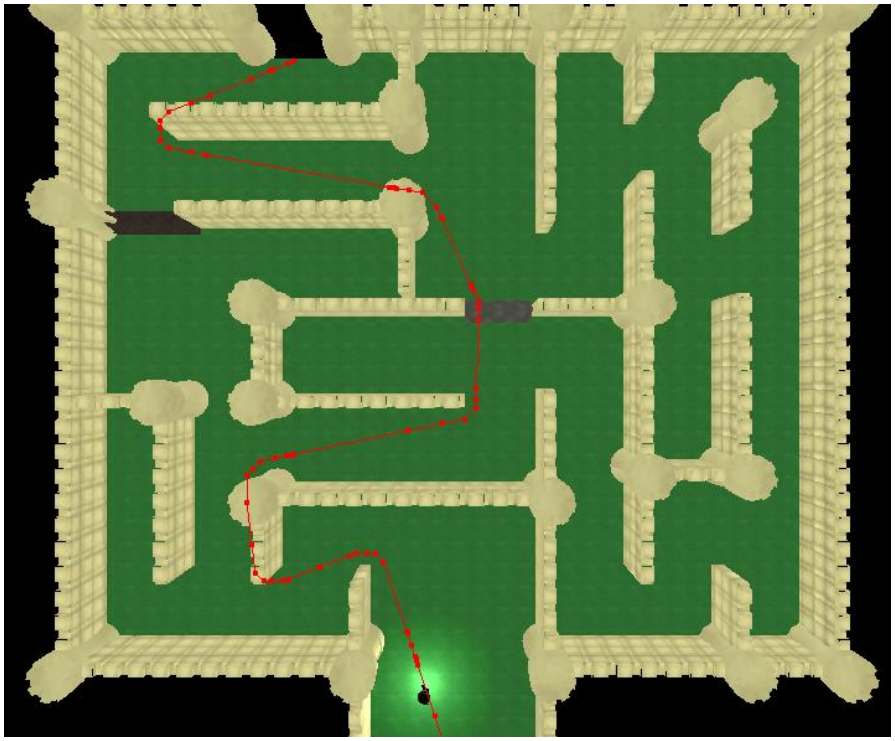
CODE NINJAS®

Silver Belt Ninja Guide

Activity 11: Labyrinth

Navigation and Pathfinding

Many game systems have all sort of enemies, followers, and NPCs (non-player characters) that need to navigate through the world. While game developers *could* manually program characters to move from point to point, it becomes very complicated when NPCs need to avoid walls, obstacles, climb or jump! To help make this process easier, many game engines use a **pathfinding** system.

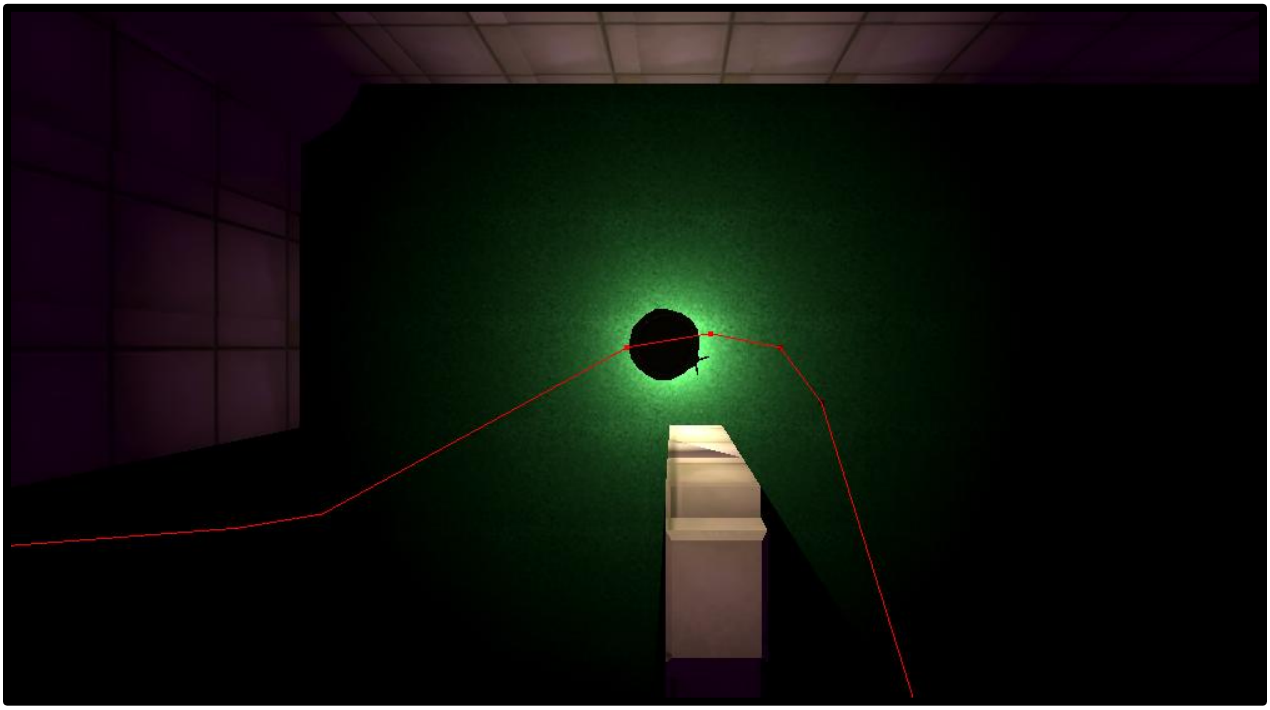


These systems allow game developers to create an **agent** that can navigate a **region**, avoiding obstacles along the way! **Agents** are given a destination, and then a path is calculated using a series of points. **Agents** then move from point along the path, until the reach the destination. Using these **pathfinding** systems, NPCs can act a lot smarter in games!

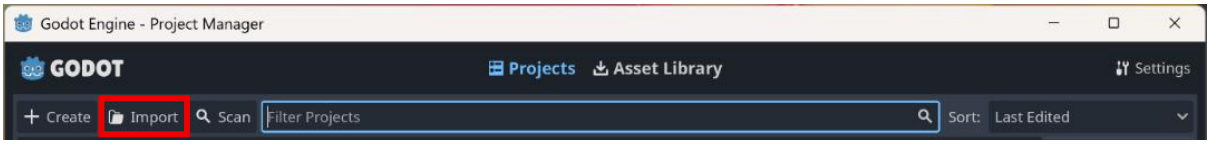
Activity 11: Labyrinth

In this activity, you will set up Codey to find the shortest path to the maze exit. This project will show you how to use Godot's navigation system.

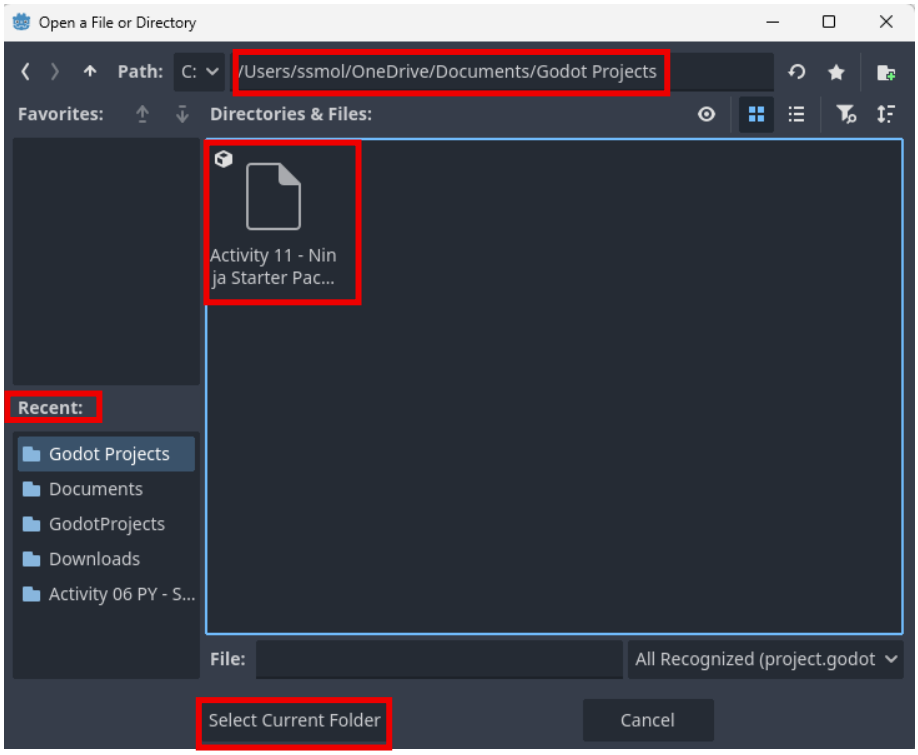
By the end of this activity, you will have explored navigation regions, programming a navigation agent to move to a destination, and how to avoid obstacles.



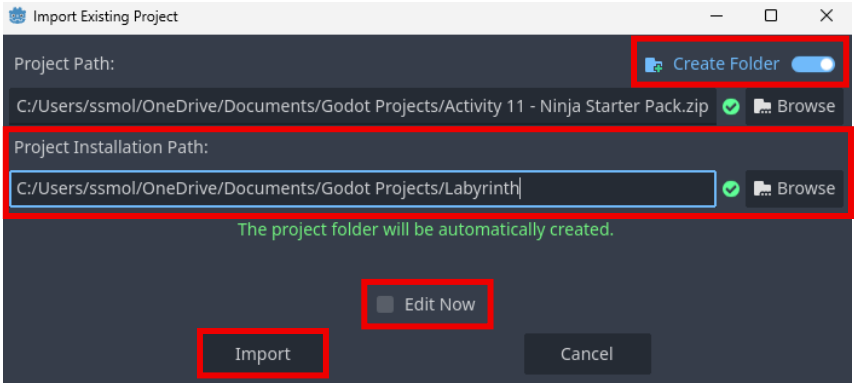
1 Open Godot and click **Import.**



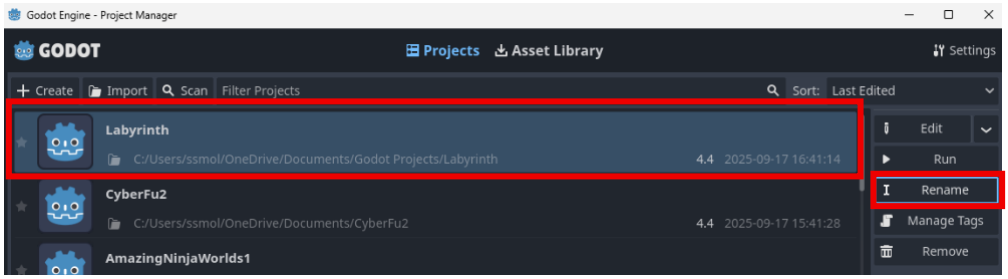
In the File Directory, navigate to the correct file path. Select **SB Activity 11 – Ninja Starter Pack.zip** and click **Open**.



2 Update the **Project Installation Path** and make sure **Create Folder** is enabled. **Uncheck Edit Now** and click **Import**.

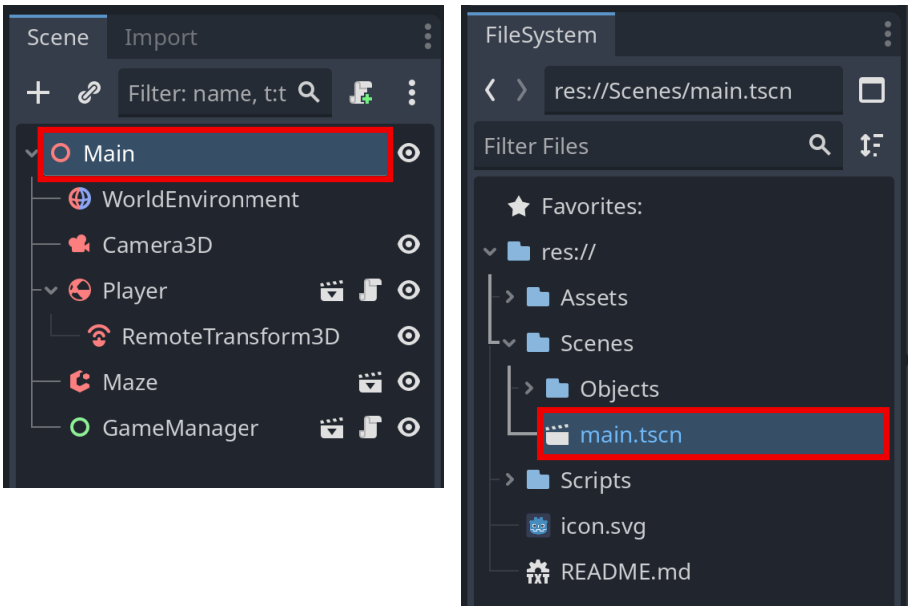


The project will appear at the top. Click on the project and select **Rename** on the right.

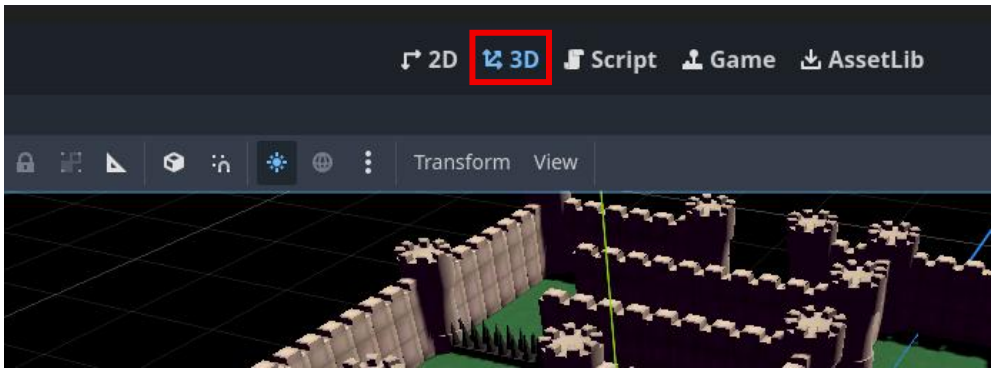


Update the Project Name to **[YourInitials]Labyrinth**. Once renamed, select the project and click **Edit** to open the starter code.

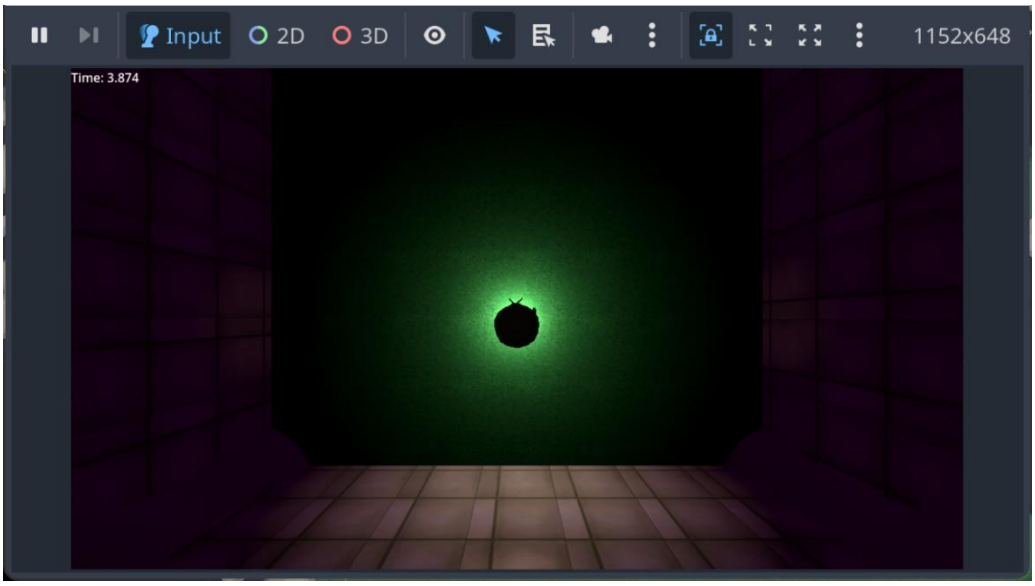
3 After importing the project, the main scene should open. If it does not open automatically, in **FileSystem**, navigate to **main.tscn** and double click to open it.



Then, above the Godot editor, check that then **3D workspace** is selected.

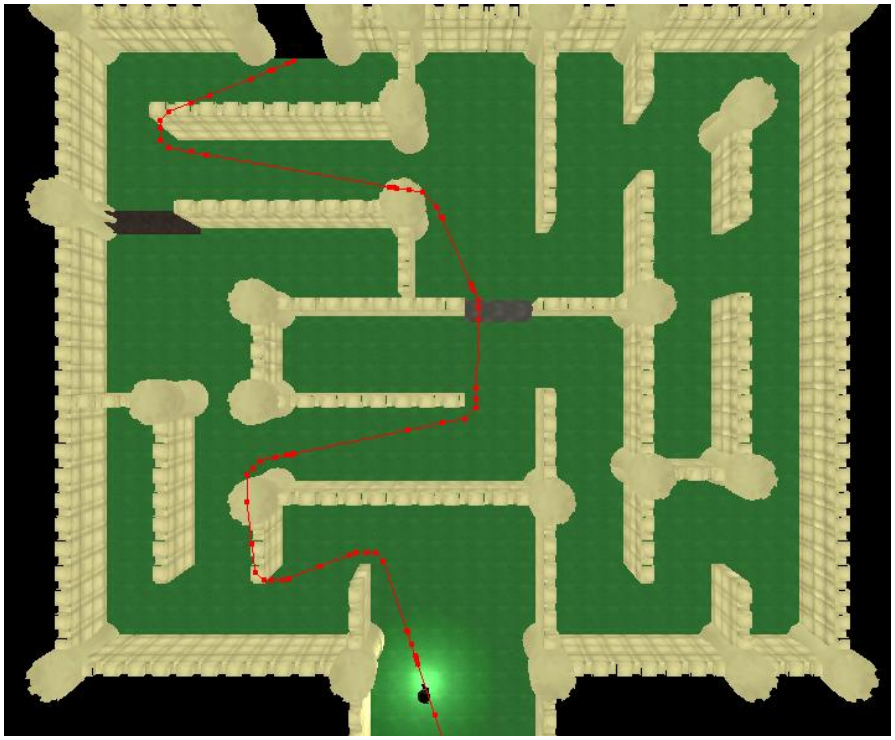


4 Playtest the project. What happens when the direction buttons are used to move Codey?



5 Codey does not move with the direction buttons.

Instead of manually moving Codey using the keyboard, Codey will be set up to move on its own using Godot's **Navigation** nodes. The Navigation nodes allow game builders to create AI agents that find the shortest path to a target destination, while avoiding walls and obstacles!



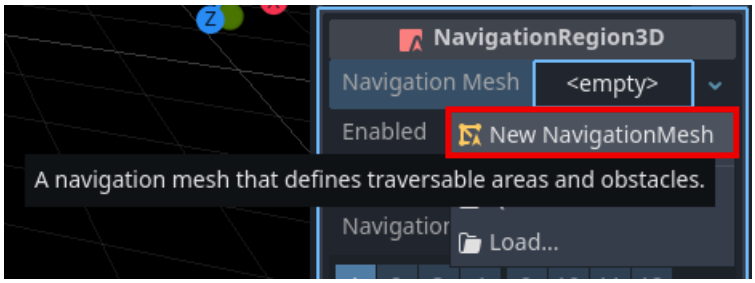
6 Add a **NavigationRegion3D** node as a child to **Main**.

The warning symbol will appear next to NavigationRegion3D. This is because the node is missing a mesh!

New Concept: NavigationRegion3D

A NavigationRegion3D is a traversable 3D region based on a NavigationMesh that NavigationAgents3Ds can use for pathfinding.

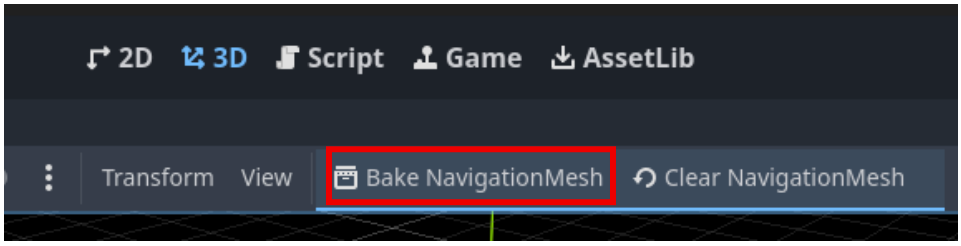
7 In the **Inspector** for **NavigationRegion3D**, click **<empty>** to add a **New NavigationMesh** to the region.



8 A navigation region needs to be **baked** so the **NavigationRegion3D** node can look at the walls, floors, and ceilings in the region and calculate which areas are walkable, and which ones are not.

After baking, the walkable areas will turn blue!

Click **Bake NavigationMesh** at the top of the 3D viewport.

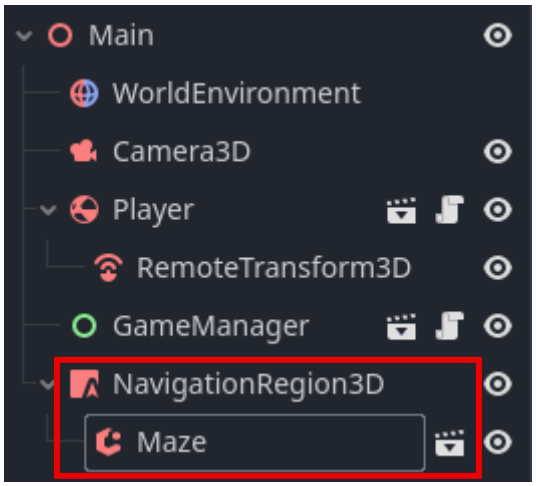


New Concept: Baking

Like baking a cake, baking in games means solidifying. When something is "baked", it means it won't be changed during the game. In this case, the walkable area won't change, so it will be baked!

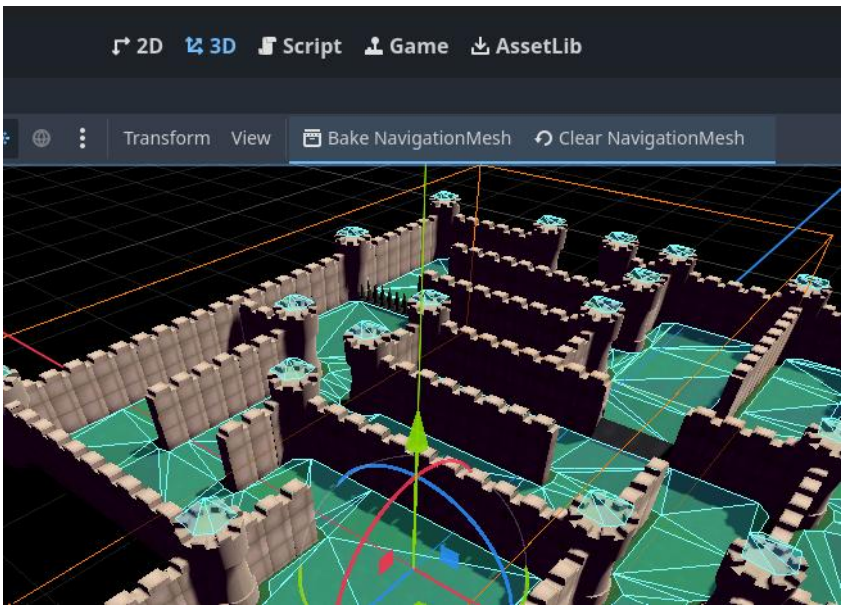
9 Wait a second! None of the floor turned blue! This means the area is not walkable yet. The **NavigationRegion3D** node only looks at objects that are children of it.

In **Scene**, notice that both **Maze** and **NavigationRegion3D** are children of **Main**. Drag **Maze** and reparent the node so it's a child of **NavigationRegion3D**.



10 With **NavigationRegion3D** selected, click **Bake NavigationMesh** again.

Notice how the map turns blue!



Pause for **Sensei Stop #1**!

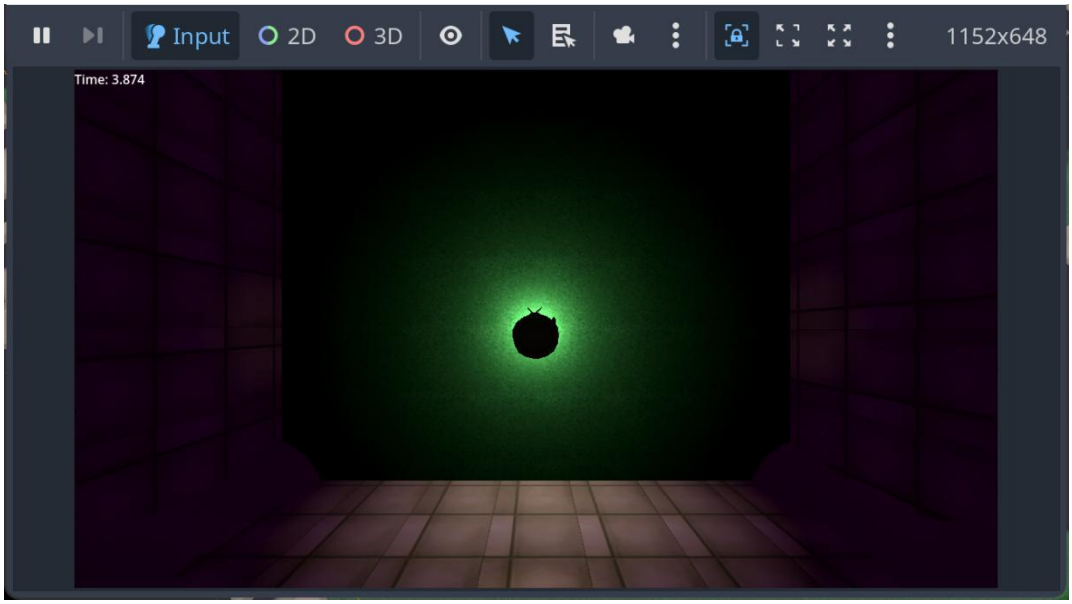
Before moving on, explain to a Code Sensei what a **navigation region** is. What does baking the region do?

Reminder: Save your work!

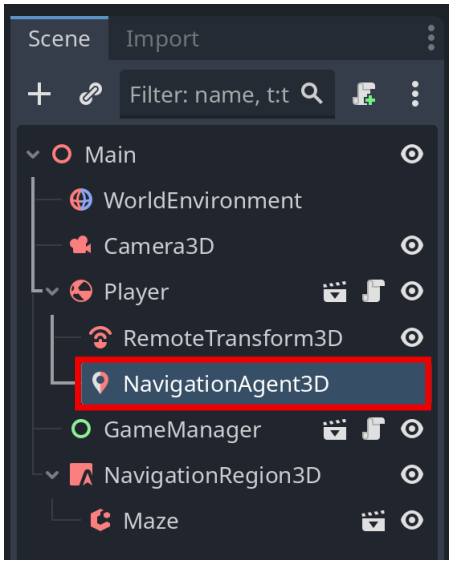
11 Playtest the project.

Codey isn't moving still! This is because Codey isn't set up to be a **Navigation agent**.

Navigation agents are the characters that move through the region. **Agents** first need a destination, then they can calculate a path of points to get to the exit. The points can't go through walls, so the **agent** will automatically find the shortest path to the destination, while avoiding walls!



12 Add a **NavigationAgent3D** node as a child to **Player**.



Notice how adding nodes as children to a scene does not add them into the specified scene. Even though **RemoteTransform3D** and **Navigation3D** have been added as children to the **Player**, they do not appear in **player.tscn**.

New Concept: NavigationAgent3D

A NavigationAgent3D is a 3D agent used to pathfind a position while avoiding obstacles. The calculation can be used by the parent node to move it along the path.

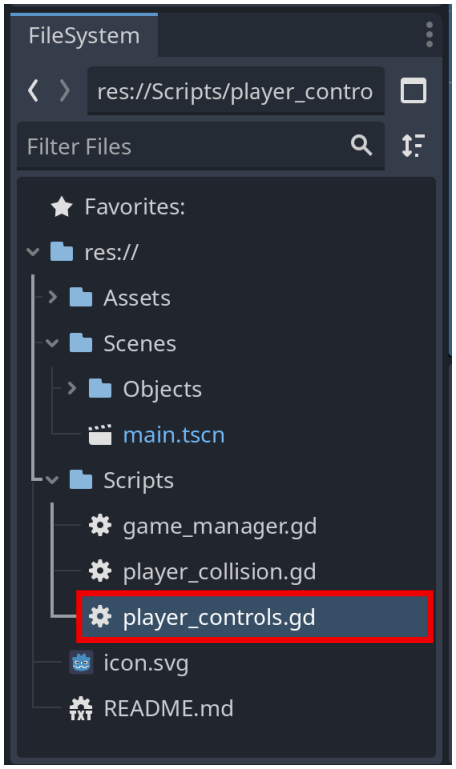
13

The agent must be programmed to move towards the exit!

This will be done in three stages:

1. Store the agent and the finish node as variables.
2. Set the destination of the agent to the finish when the game starts.
3. During the game, move the agent from point to point along the path until it reaches its destination.

In **FileSystem**, open the **player_controls.gd** script.



Notice the two variables at the top of the script:

- `anim_player` – the animator for the player
- `move_speed` – a float that will control how fast the player moves later

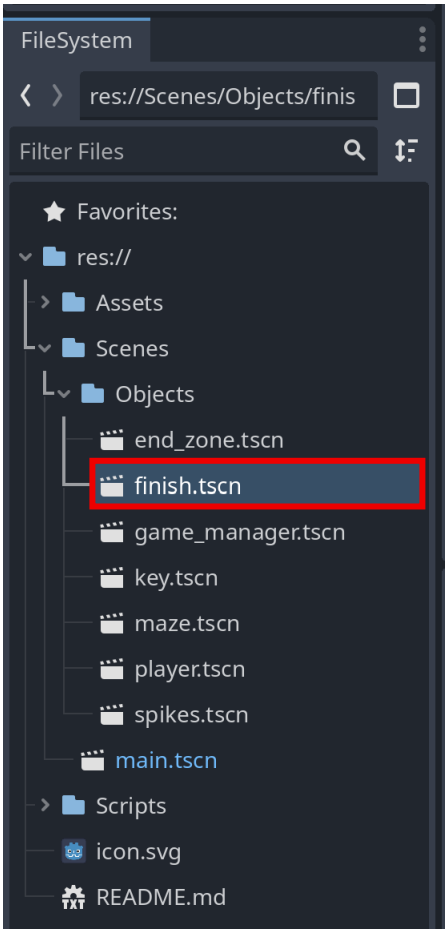
```
1   extends RigidBody3D
2
3
4   @onready var anim_player : AnimationPlayer = $AnimationPlayer
5   @export var move_speed : float = 10.0
6
```

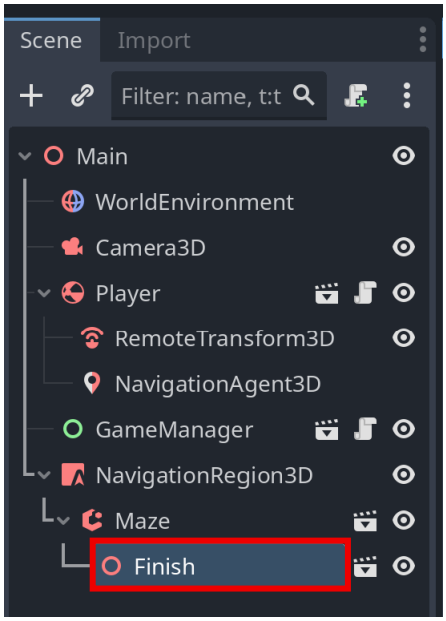
14 Under **TODO 1**, declare two variables:

- `@onready nav_agent` – of type **NavigationAgent3D** and stores the **NavigationAgent3D** node
- `@export finish` – of type **Node3D**, which will store the location of the exit node

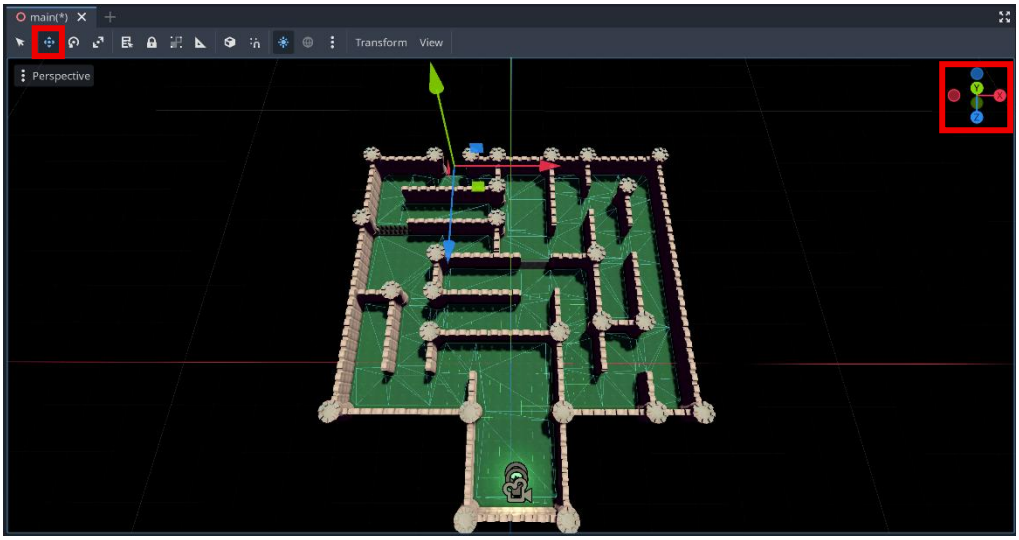
```
# --------------------
# TODO  1
# Create the navigation variables
# --------------------
@onready var nav_agent: NavigationAgent3D = $NavigationAgent3D
@export var finish: Node3D
```

15 In the **FileSystem**, under **Scenes > Objects**, add the **finish.tscn** scene to Main as a child of **Maze**.

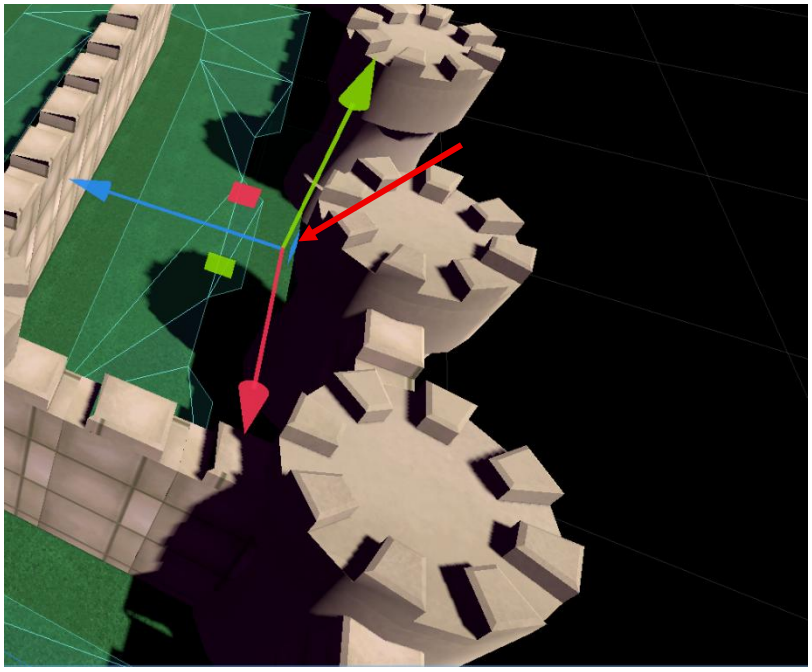




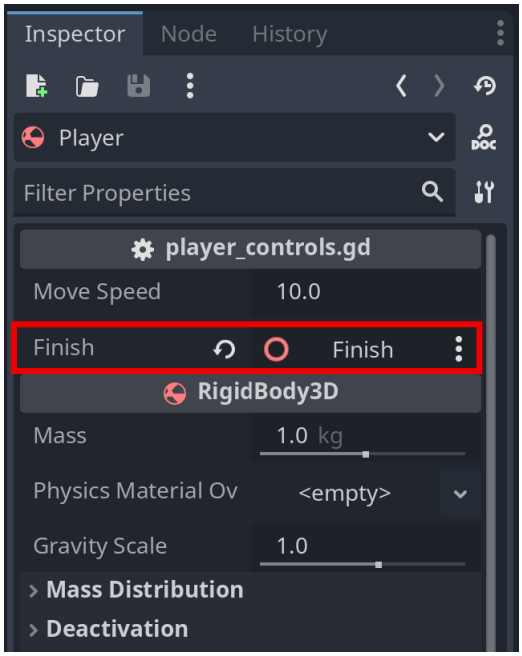
16 In the **3D workspace**, use move mode and the mouse or the manipulator gizmo to position the **Finish** node at the maze's exit.



When placing the Finish node, be sure to place the node fully inside the maze by making sure the blue move box is inside the maze exit.



17 In the **Inspector** for **Player**, assign the **Finish** node to the **Finish** variable.



18 Return the **player_controls.gd** script.

Underneath **TODO 2**, add a `_ready()` method. Inside, set the `nav_agent target_position` to `finish` node's position using the `position` property.

```
14  # --------------------
15  # TODO  2
16  # Create the navigation agent target
17  # --------------------
18  # _ready()
19      # nav_agent.target_position = ???
20
21
```

19 Now the agent must move along the path. The agent's movement code should be placed inside a `_physics_process()` method.

Under **TODO 3**, declare the `_physics_process()` method .
Inside `_physics_process(),` use the `look_at()` method to point the player towards the nav_agent's next path position using `nav_agent.get_next_path_position()`. When the point is reached by the agent, this will automatically update to the next point in the list. Remember to use `Vector3.UP` as the up direction and `true` to have the rotation relative to the player.

```
20
21  # --------------------
22  # TODO  3
23  # Move the navigation agent towards the target along the path
24  # --------------------
25  func _physics_process(delta: float) -> void:
26      look_at(nav_agent.get_next_path_position(),Vector3.UP, true)
```

`get_next_path_position()`: Returns the next position in global coordinates that can be moved to, making sure that there are no static objects in the way. If the agent does not have a navigation path, it will return the position of the agent's parent.

The use of this function once every physics frame is required to update the internal path logic of the NavigationAgent.

Parameters: *None*

Returns: Vector3D

20 On the following lines, set `linear_velocity` to `basis.z` (which will be towards the next point on the path!) multiplied by the `move_speed`, then use the `play()` method to run the `anim_player`'s `Run` animation.

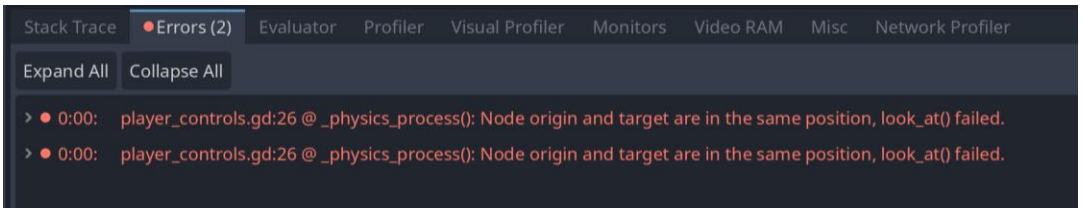
```
21  # --------------------
22  # TODO  3
23  # Move the navigation agent towards the target along the path
24  # --------------------
25  func _physics_process(delta: float) -> void:
26      look_at(nav_agent.get_next_path_position(),Vector3.UP, true)
27      # linear_velocity = ??? * ???
28      # anim_player
29
```

21 Check the code for TODO 2 and TODO 3 then update the script as needed. Save the script.

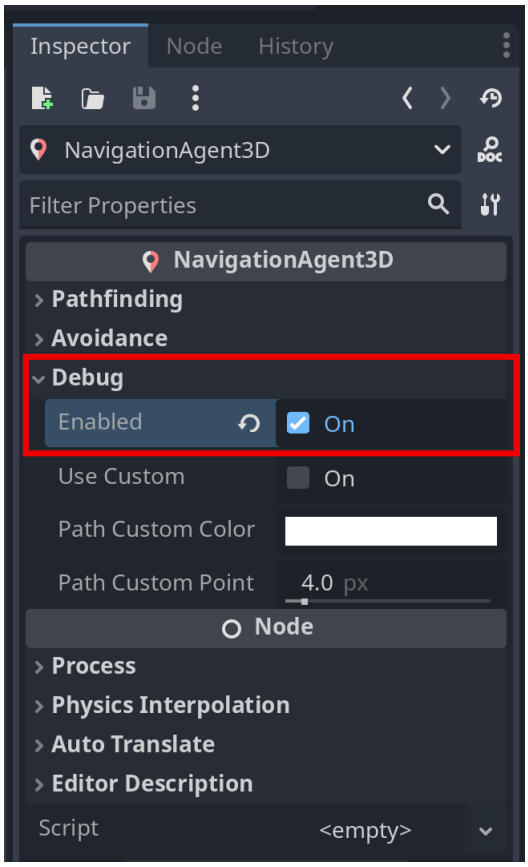
```
14  # --------------------
15  # TODO  2
16  # Create the navigation agent target
17  # --------------------
18  func _ready() -> void:
19      nav_agent.target_position = finish.position
20
```

```
21  # --------------------
22  # TODO  3
23  # Move the navigation agent towards the target along the path
24  # --------------------
25  func _physics_process(_delta: float) -> void:
26      look_at(nav_agent.get_next_path_position(),Vector3.UP, true)
27      linear_velocity = basis.z * move_speed
28      anim_player.play("Run")
29
```

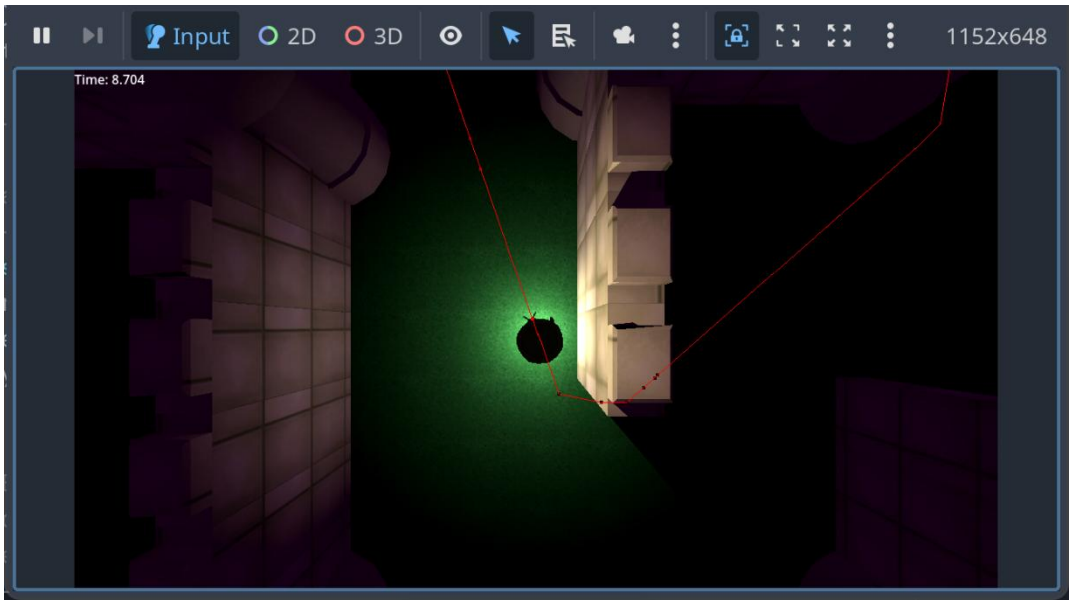
When playtesting the project in the following steps, some errors may appear. This is because when the game first starts, the first point of navigation is at the same position as the **`look_at()`** observer (Codey). This error can be ignored as it will not cause any issues in the game.



22 Before testing, go to the **Inspector** for **NavigationAgent3D**. Under **Debug**, check the box beside **Enabled** to enable debugging mode to view the path Codey takes as a line.



23 Playtest the game. What do you notice about Codey's movement now?



Pause for **Sensei Stop #2**!

Before continuing, check with a Code Sensei and explain the importance of **NavigationAgent3D**. Discuss:

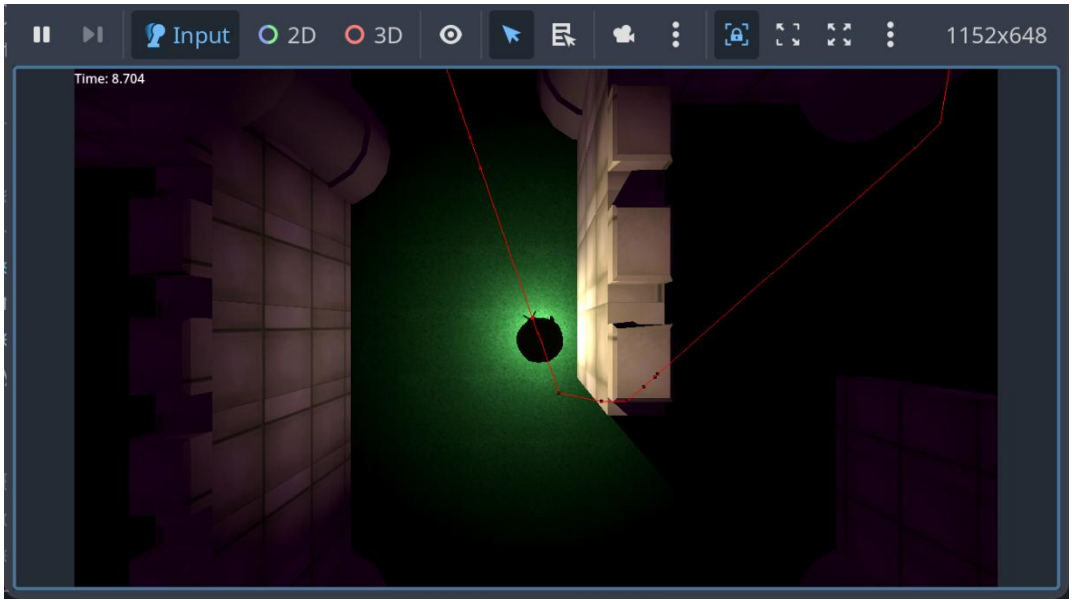
- *How does the agent know where the exit is?*
- *How does the agent move to the exit and avoid walls?*

Reminder: Save your work!

24

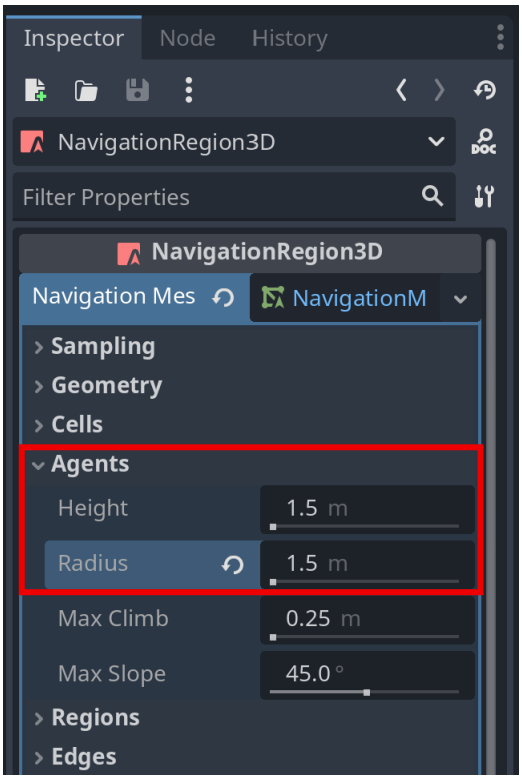
While Codey can reach the exit, it's still not the smoothest.

Firstly, Codey is still hitting some of the walls! That's because Codey's collider is larger than the current agent radius property.

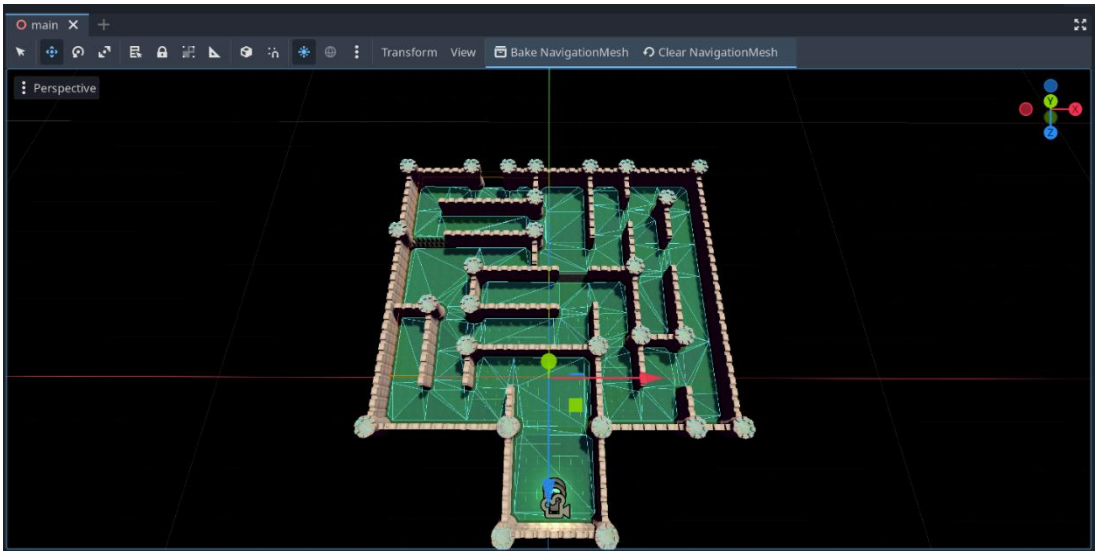


25 In the **Inspector** for **NavigationRegion3D**, select the **NavigationMesh** to open its properties.

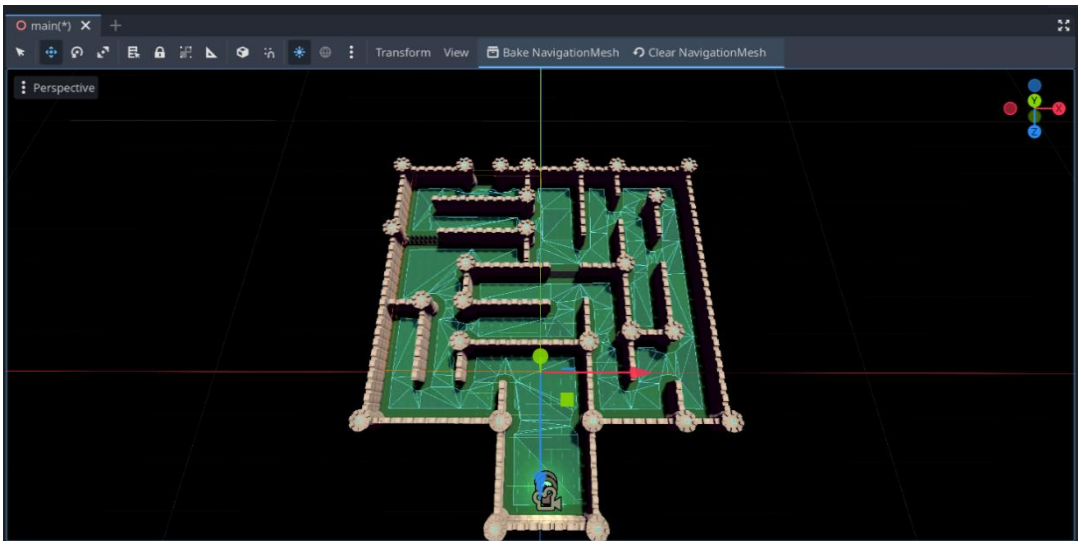
Under **Agents**, change the **Radius** to a larger number, like **1.5**. Now the agent's radius property is the same size as Codey's collider!



26 **Bake** the navigation mesh.

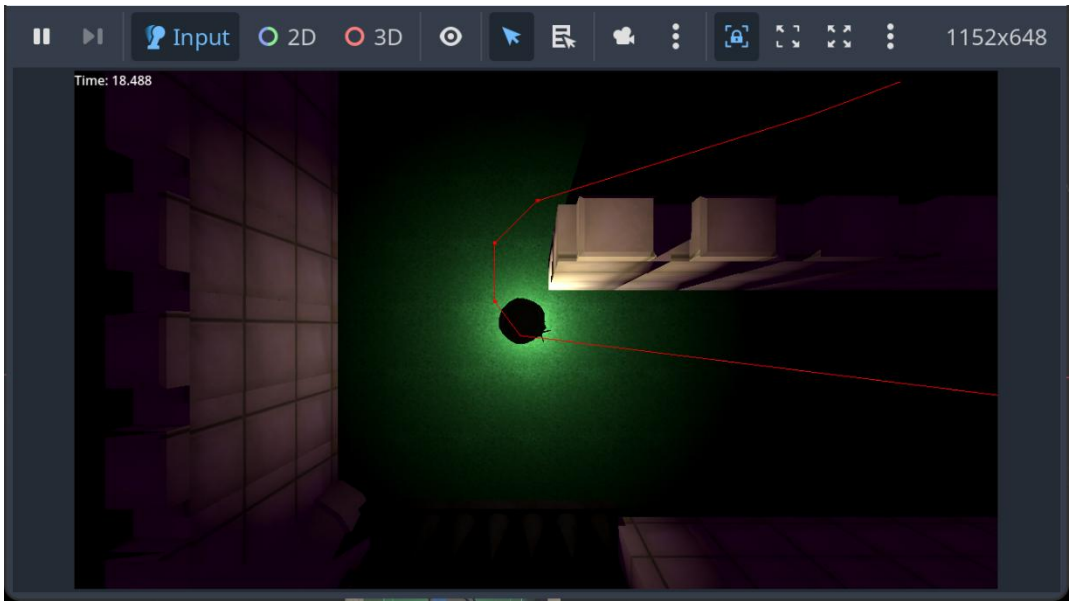


Notice how the blue area is further away from the walls now!



27 Save and playtest the game.

Notice how Codey now avoids the walls easier!

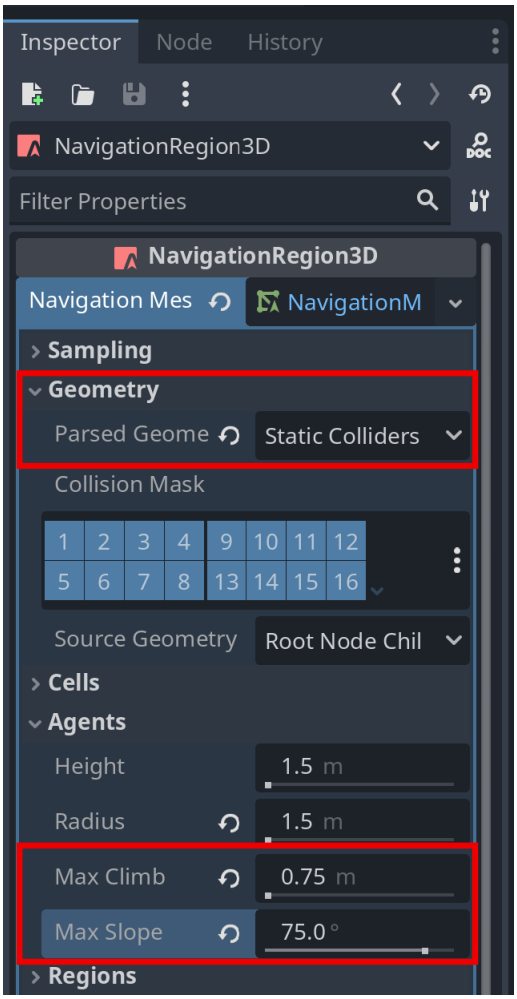


28 It takes Codey quite a long time to reach the exit.

However, on the map there is a small slope. Codey is currently avoiding this because the slope is too high to climb. This can be changed by modifying the **agent** settings!

29 In the Inspector for NavigationRegion3D, update the following settings:

- Under **Geometry** set **Parsed Geometry** to **Static Colliders**.
- Under **Agents** set **Max Climb** to **0.75** and **Max Slope** to **75.0** degrees.



30

Bake the navigation mesh.

Find the ramp in the 3D workspace and notice that the ramp is now blue and can be part of Codey's path!

Pause for **Sensei Stop #3**!

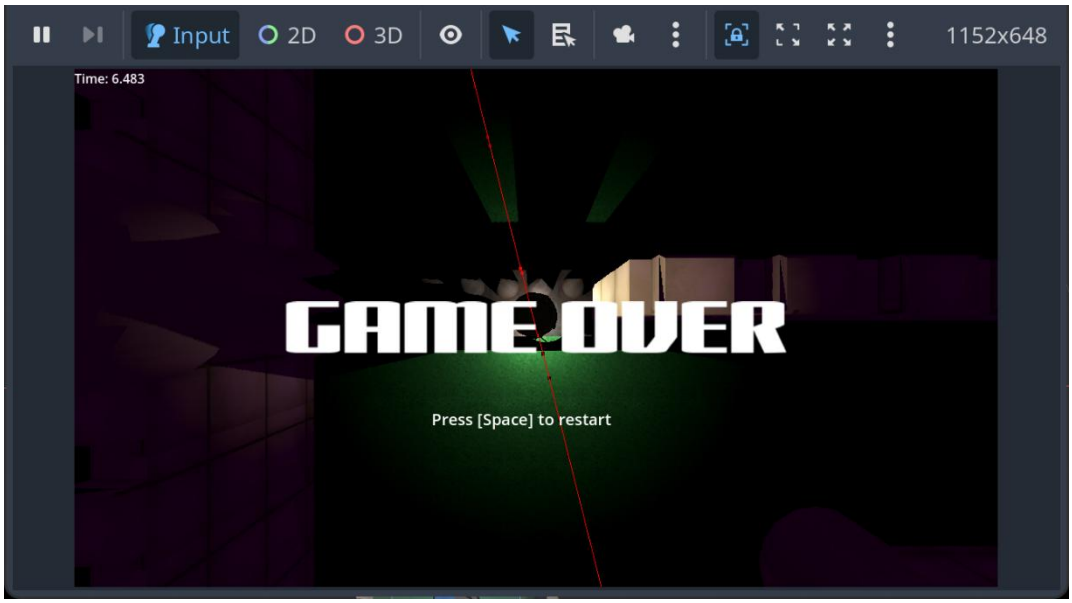
Before continuing, check with a Code Sensei and explain how changing agent settings can be useful for determining behavior. Discuss:

- *What are some of the agent's properties?*
- *What do these properties change?*

Reminder: Save your work!

31 Playtest the game.

Uh oh! What happens when Codey encounters the spikes?



32 When updating the agent settings to avoid the ramp, the spike obstacles also became walkable!

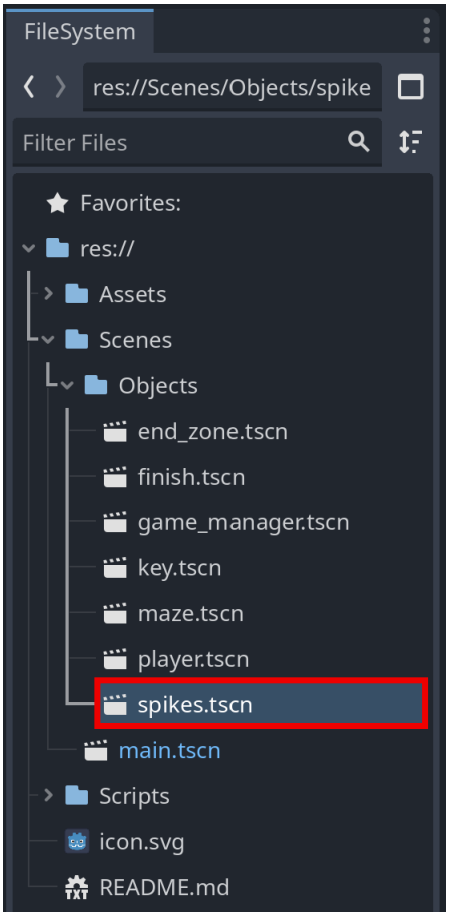
Codey needs to be set to avoid certain obstacles. This can be done with the **NavigationObstacle3D** node.

New Concept: NavigationObstacle3D

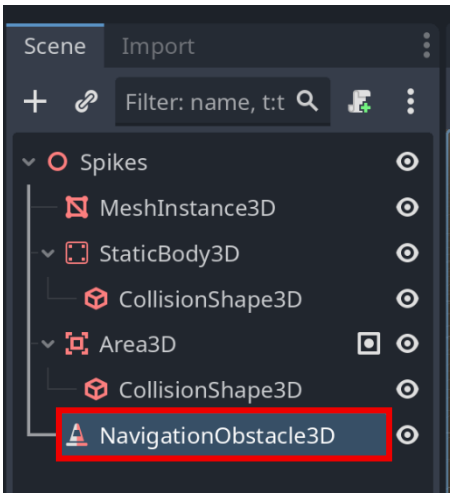
The **NavigationObstacle3D** node tells the agent to avoid certain regions of the map, even if they might be walkable.

.

33 In **FileSystem**, open the **spikes.tscn** packed scene from the **FileSystem** under **Scenes > Objects**.



34 Add a **NavigationObstacle3D** node as a child to **Spikes**.

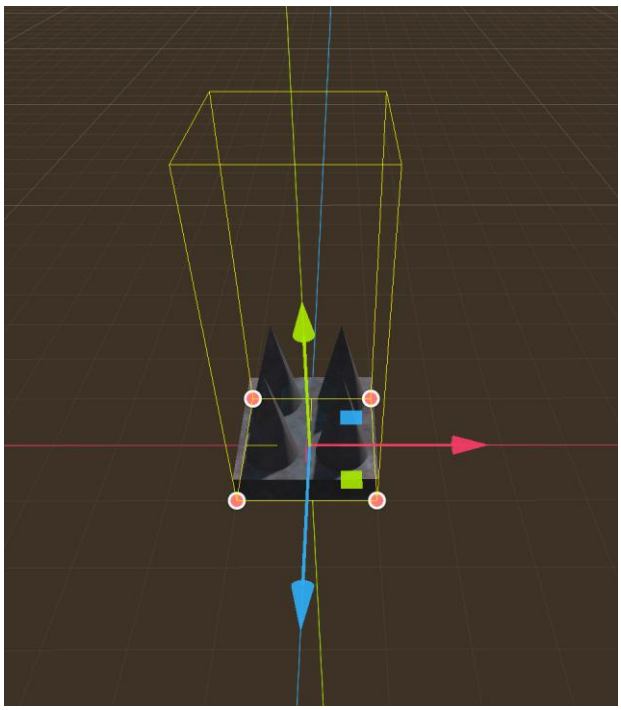


35 In the **Inspector** for **NavigationObstacle3D**, set the **height** setting to **5**.

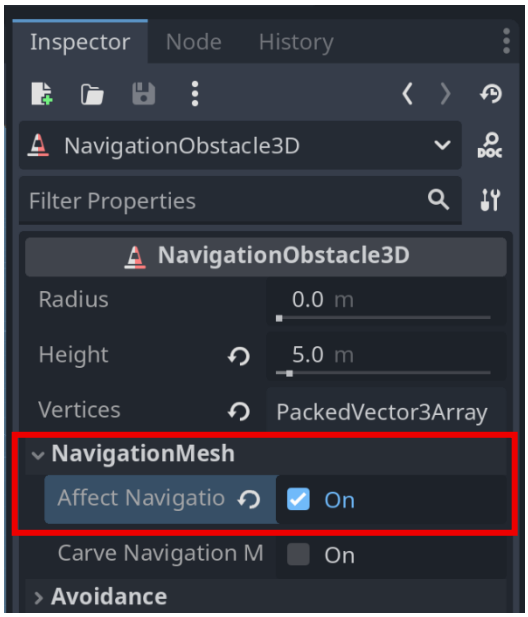
Next to **Vertices**, click **PackedVector3Array** and set **Size** to **4**. Add the following 4 vertices to the array:

- (-1, 0, 1)
- (1, 0, 1)
- (1, 0, -1)
- (-1, 0, -1)

This will set the size of the navigation obstacle box to cover the spikes.



36 Under **NavigationMesh**, check the box to enable **Affect Navigation Mesh**.

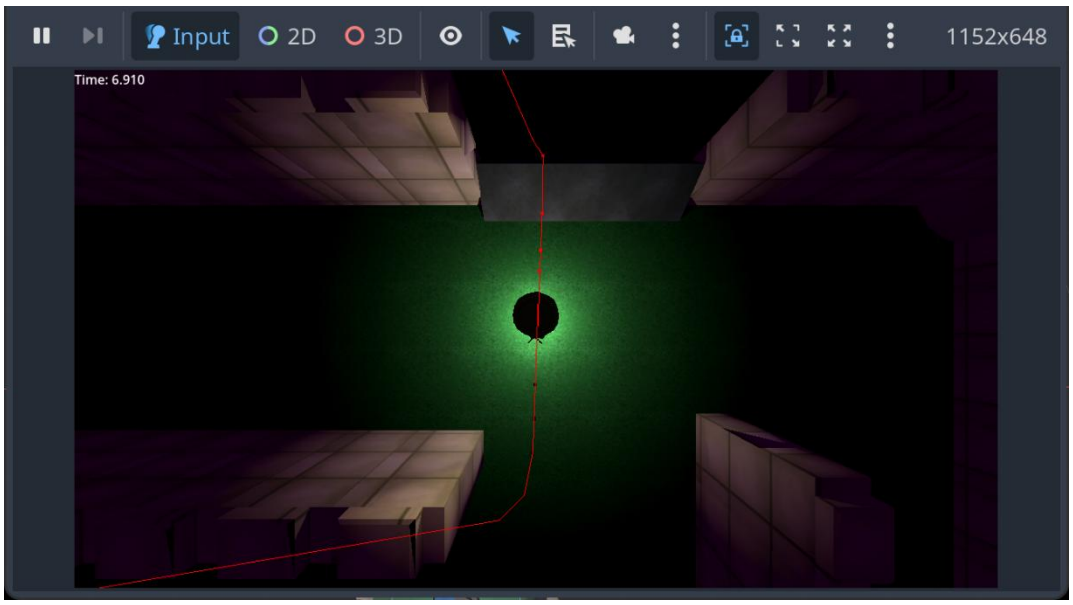


37 Save the **spikes** scene, then return to the **main** scene.

Select **NavigationRegion3D**, then **bake** the mesh. Now the spikes aren't blue!

38 Playtest the game.

Codey now gets to the exit much faster by going over the slope, but doesn't try to move through the spikes!



Pause for **Sensei Stop #4**!

Before continuing, check with a Code Sensei explain how navigation obstacles can be used to avoid certain obstacles. Discuss:

- *What is the purpose of **NavigationObstacle3Ds**?*
- *When are some good times to use them in games?*

Reminder: Save your work!