CODE NINJAS®

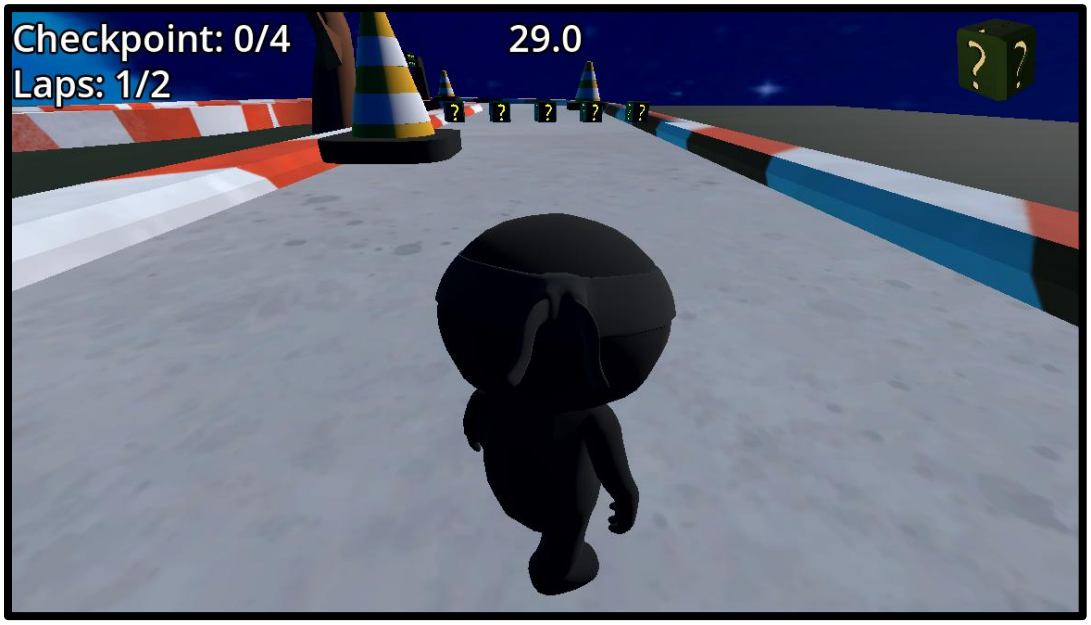
Platinum Belt Ninja Guide

PB Activity 02: Codey Raceway

Contents

PB Activity 02: Codey Raceway

In this project, you will design a racing game complete with obstacles, powerups, timers and a game over user interface (UI). You will work with signals, collisions, navigation agents and create different UI components.



Pro Tips

Now that you've worked through two Platinum Belt projects, it's time to ramp things up in PB Activity 02: Codey Raceway! This project contains many requirements and fewer instructional steps, which may not feel as detailed as you are used to. The requirements should feel familiar and most of the instructional steps contain familiar concepts applied in a slightly new way. This project is set up to feel more like a Quest in IMPACT.

Here are a few problem-solving strategies that you can use if you feel stuck while working through the requirements and instructions:

- Take things one step at a time and break down each instruction or checkbox into manageable chunks.
- Write pseudo code, either with a pen and paper or with code comments in the script.
- Use lots of print statements to see which functions are being called when, and which aren't, or if a variable is updating as expected.
- Tinker with values and playtest often. Don't be afraid to test out code, see if it works and adjust if it doesn't.
- Use the hints as needed, whether you're feeling stuck or to confirm you're on the right track.
- Refer to previous projects or documentation. This will be included in the Resources section for each requirement, but you can refer to any additional projects or documentation that you feel may be helpful.
- Check in with Code Sensei if you're still feeling unsure about a step or a requirement.

Requirement #1: Project Set Up

Set up the project and the main scene.

- ☐ Import the Ninja Starter Pack as a new project in the correct installation path.
- ☐ Rename the project to **[YourInitials]CodeyRaceway** and open the project.
- ☐ Create a new **3D** scene named **racetrack.tscn**, save the new scene in the **Scenes** folder, and set the main scene to racetrack.tscn.

Requirement #1: Hints

- Which button imports files as a new project in the Project Manager?
- Where is the starter code stored? What file type is the starter pack?

Requirement #1: Resources

- Refer to Activities 06 – 17 in SB for help with importing a project.

Requirement #2 (Instructions): Camera Setup

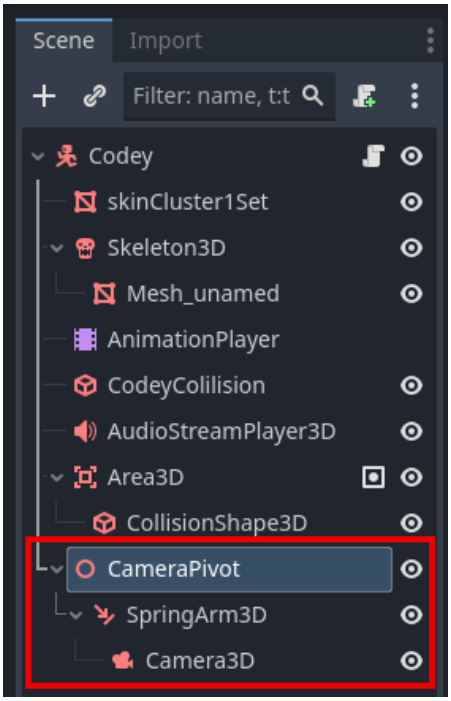
Codey is responsible for... • ~~User input & movement~~ • Camera movement • Resetting after colliding with a hazard • Spawning & using powerups • Updating and displaying the powerup UI
The finish line is responsible for... • Letting the game know when Codey has crossed the finish line
The checkpoint is responsible for... • Letting the game manager know when Codey has passed a checkpoint • Ensuring a checkpoint can't be "passed" more than once per lap • Resetting
The game manager is responsible for... • Counting checkpoints as they're passed by Codey • Checking if all the checkpoints have been passed when Codey crosses the finish line • Telling checkpoints when to reset • The time limit and race countdown • Checking if Codey has completed all the necessary laps • Updating and displaying the game UI • Displaying the game over UI
BoxSpawner is responsible for... • Spawning and placing the Mystery Boxes along the track
The Mystery Boxes are responsible for... • Their rotation • Disappearing when Codey collides with a box • Respawning after a set amount of time
The Powerups are responsible for... • Their movement • Destroying themselves after colliding with a Hazard
The Hazards are responsible for... • Destroying themselves

Requirement #2 (Instructions): Camera Setup

1 Most of the user input and player movement is included in the starter code. Set up the camera to follow behind Codey.

Find and open the **codey.tscn** scene in the Scenes folder. Add a **Node3D** named **CameraPivot** as a **child** to **Codey**. This node will control the camera's movement.

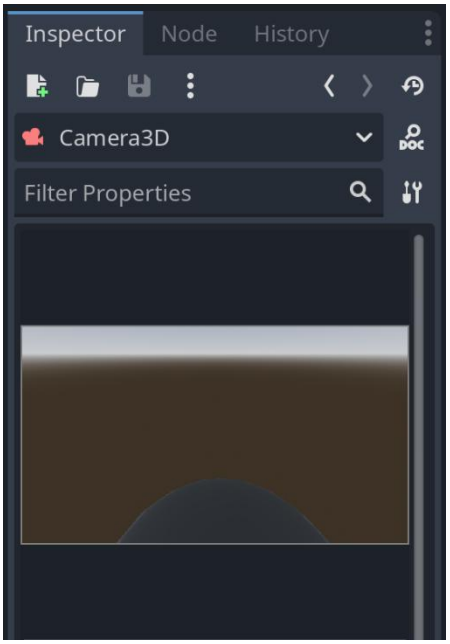
Then, add a **SpringArm3D** as a **child** to **CameraPivot**, and a **Camera3D** as a **child** to **SpringArm3D**.



New Node: SpringArm3D

A 3D raycast that moves its children near the collision point. This is helpful for 3rd person camera movement, especially when the camera needs to remain close to the player when in a tight space.

2 In the Inspector for **Camera3D**, adjust the **Rotation x** so the camera looks down on Codey. Then, in the Inspector for **CameraPivot**, adjust the **Position y** so the node is positioned slightly above Codey, then set **Rotation y** so the Camera looks at Codey from behind.



Note: The preview window in the Inspector for Camera3D may be helpful here.

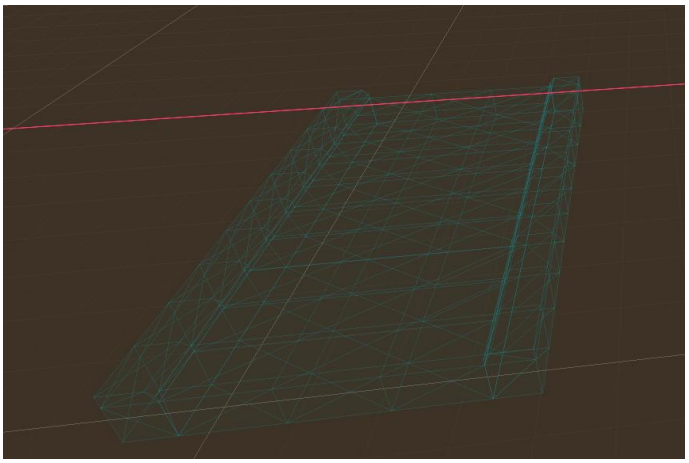
3 In the Inspector for **SpringArm3D**, set a value for **Spring Length**. The length of the spring arm is how far from its global position to check for collisions. This is also how far behind Codey the camera will appear.

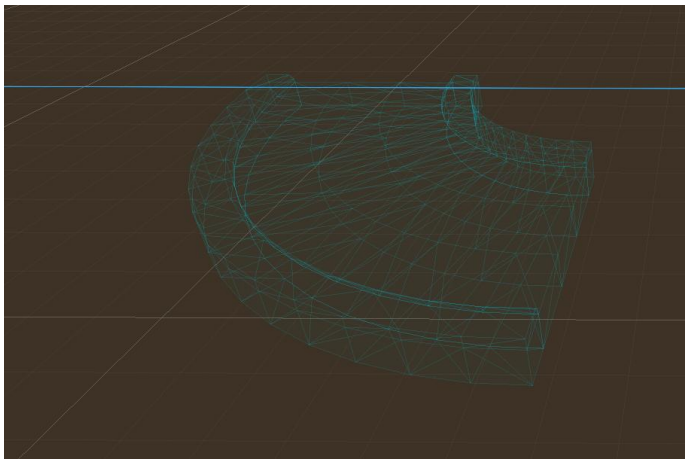
Playtest the **codey.tscn** scene to test the camera view.

Requirement #3: Set Up the Track Pieces

Set up the track pieces so Codey can't fall through the track.

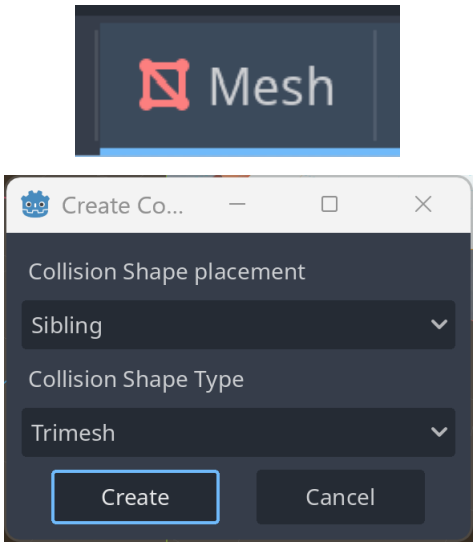
- [ ] Find and open the **straight_track.tscn** and **turn_track.tscn** scenes in the **Scenes > Track** folder. The track scenes are missing the two types of nodes which prevent Codey and other objects from falling through the track. Complete the two track scenes.





Requirement #3: Hints

- ☐ Do both scenes have a **StaticBody3D**?
- ☐ Were the **CollisionShape3D** nodes created as a **Trimesh Collision Shape** from the **MeshInstance3D** nodes in the scene, then **reparented** the CollisionShape3D as needed?



Requirement #3: Resources

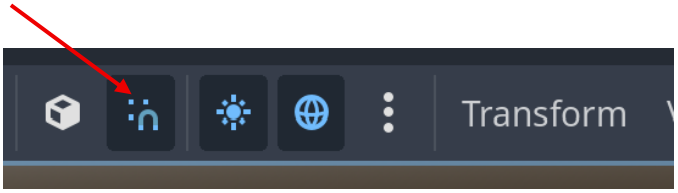
- Refer to platform creation in BB Activity 04: Sketch Head
- [Godot 4.4 StaticBody3D Documentation]

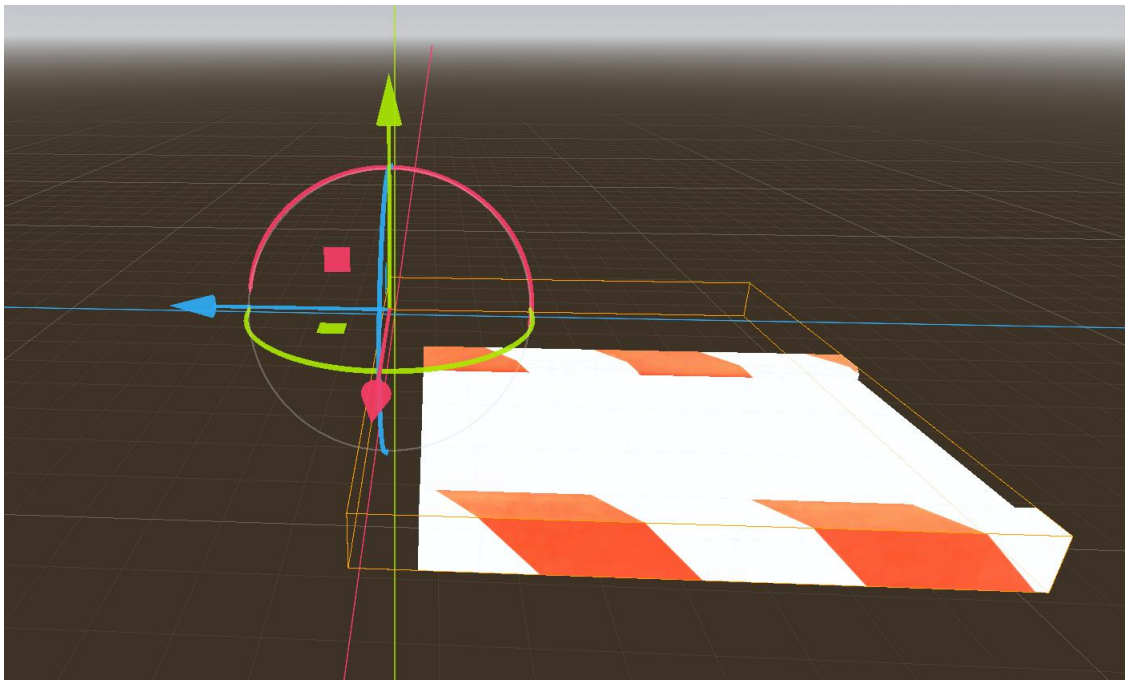
Requirement #4 (Instructions): Assemble the Racetrack

Create the racetrack for Codey.

4. Add a **Node3D** named **Track** as a **child** to **Racetrack**. This node will hold all the track pieces.

5. Use the **straight_track.tscn** and **turn_track.tscn** scenes to build out the racetrack. Turn on **Snaps** and use **Move Mode** by **dragging the direction arrows** to make aligning the track pieces easier. This will transform the track pieces one unit at a time.





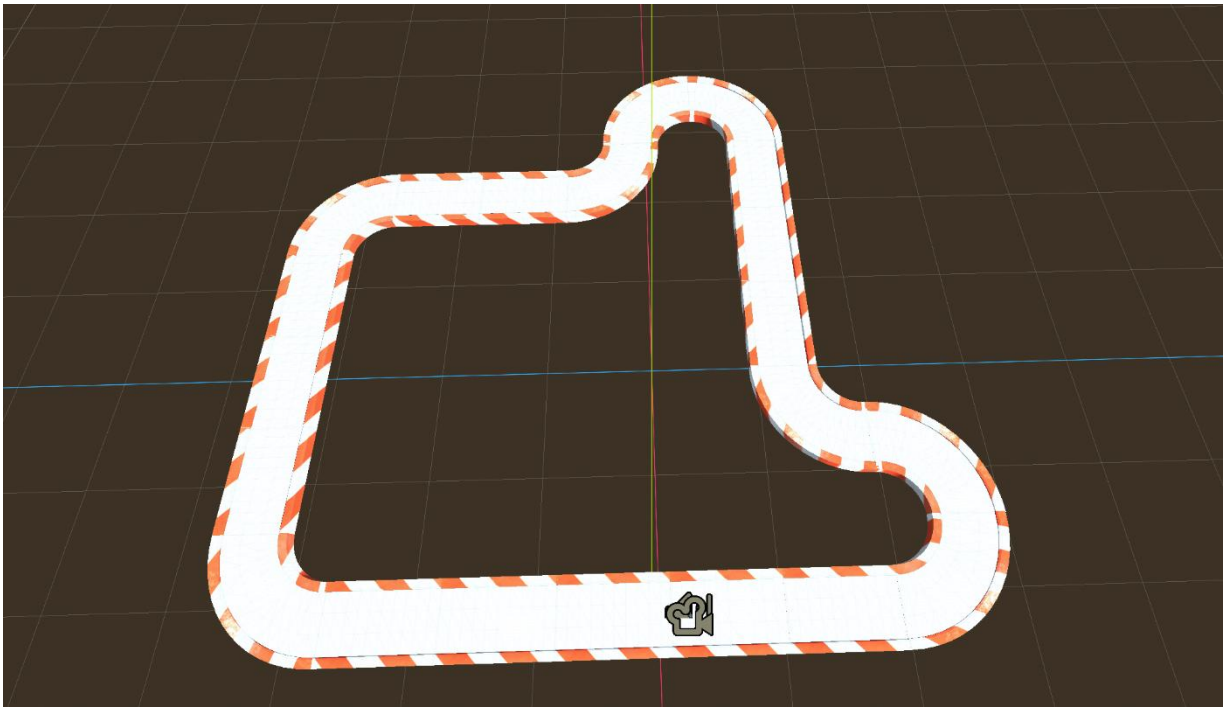
6 Adjust the track pieces as needed by updating the **x** and **z rotation** and **position** values in the **Inspector**.

Don't forget to use the transformation gizmo to adjust the view in the 3D workspace, and the mouse wheel to zoom in and out.

Pro Tip:

All the track pieces will have a y position of 0 when added to the scene. Only the x and z position need to be adjusted when building the track!

7 Add **Codey** to the scene as a **child** of **Racetrack**, then place Codey at a starting position on the Track.



Pause for **Ninja Stop #1**!

When playtesting...

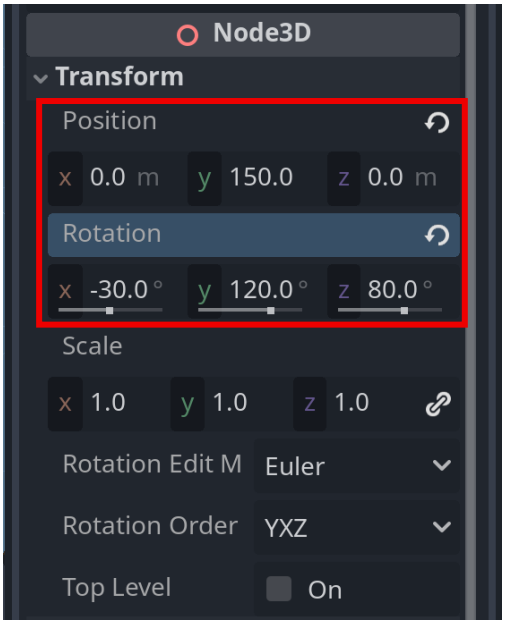
- Can Codey run along the track instead of falling through the pieces?
- Does the game appear very dark?

Reminder: Save your work!

Requirement #5 (Instructions): Environment Set up

8 When playtesting the main scene, Codey and the racetrack can't really be seen. Directional lighting and a world environment need to be added. Add a **DirectionalLight3D** as a child to **Racetrack**.

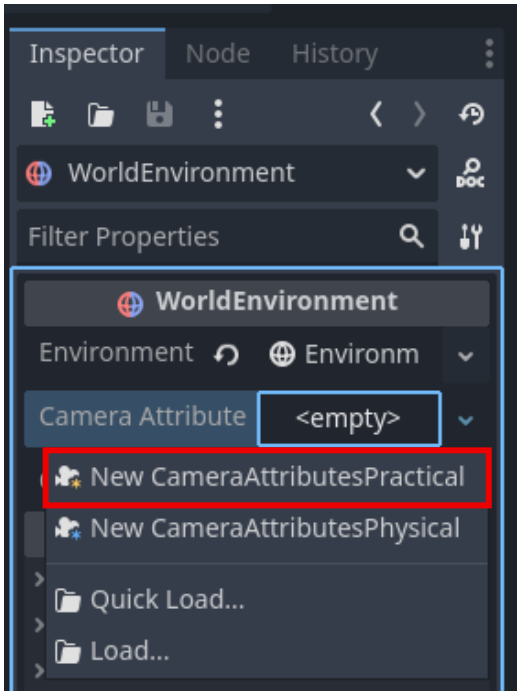
In the **Inspector**, adjust the **y position** and **rotation** so the track is lit from above.



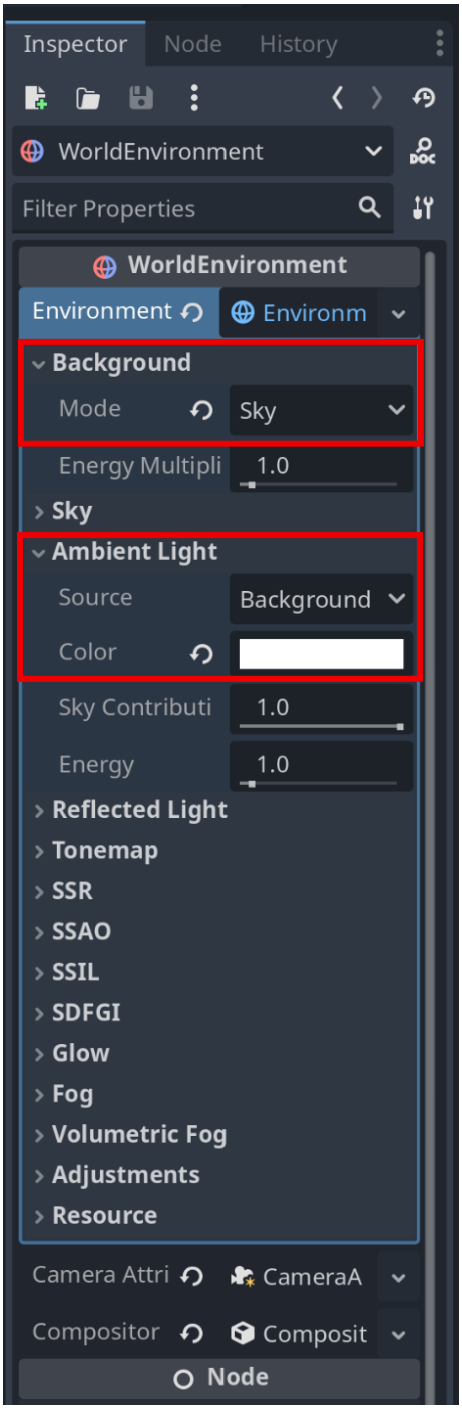
Reminder:

A **DirectionalLight3D** is a light emitting node that casts an infinite sheet of light rays across the entire scene based on the DirectionalLight3D's rotation values.

9 Add a **WorldEnvironment** as a **child** to **Racetrack** and set **Environment** to **New Environment**, **Camera Attributes** to **New CameraAttributesPractical** and **Compositor** to **New Compositor** in the **Inspector.**



10 Click on **Environment > Background** and set **Mode** to **Sky** and choose a **Color** under **Ambient Light.** This will be the color the light casts in the game scene. It could be set up to mimic sunlight, moonlight, etc.

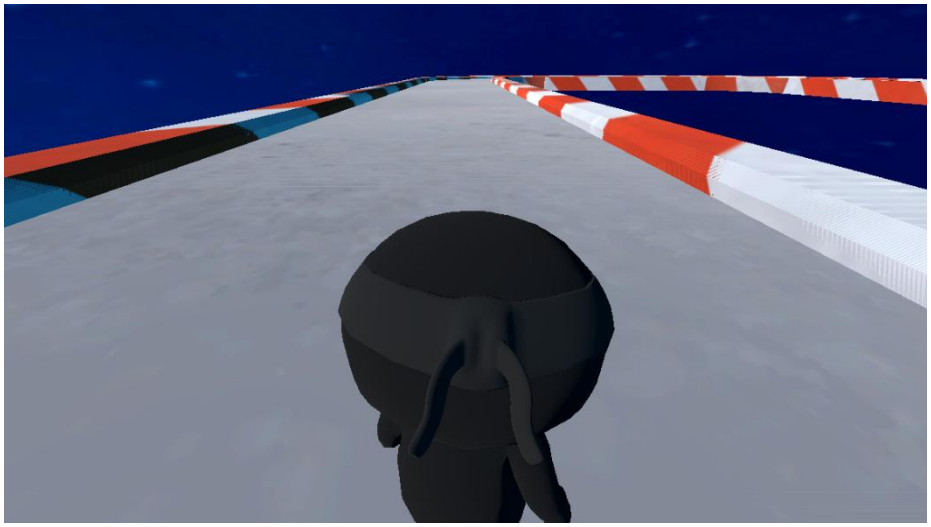


11 Add a sky to the game scene.

Under Sky, set **Sky** to **New Sky,** open **Sky** and set **Sky Material** to **New PanoramaSkyMaterial.**

Then, click **Panorama** and drag and drop a texture for the sky into **Panorama**. Find the **Sky Textures** in the **Assets** > **Textures** > **Sky Textures** folder.

12 Playtest the game and run along the racetrack to check if all the track pieces are aligned. Use **WASD** or the **direction buttons** to move Codey and hold down the **shift** key while moving to sprint or use the spacebar to jump.



It may be helpful to adjust the position in the Inspector under transform when fine-tuning the track alignment, but keep in mind that not all the pieces will align together perfectly.

13

Create a ground for the game scene.

Add a **Node3D** as a **child** to **Racetrack** and **rename** the node **Background**. This will help organize and store different components of the track scene.

Then, create a ground for the game scene. A **StaticBody3D**, a **CollisionShape3D** and a **MeshInstance3D** are needed and one of these nodes will parent the other two.

Some ground materials can be found in the Assets > Textures folder and applied in the **Inspector** for **MeshInstance3D** under **Material**.

Pro Tip:

Adjust the **size** of the mesh and the collision shape when making the ground, not the scale.

14

Find the **tree_large.tscn** and **tree_small.tscn** in **Scenes > Objects** folder and add trees in to decorate the background. Organize the trees as children to a **Trees** Node3D under Background.

Track can also be reparented to the **Background** node to keep organized.

Pause for **Sensei Stop #1**!

Check with a Code Sensei and confirm that requirements 1 – 5 are set up correctly.

Reminder: Save your work!

Requirement #6: Create The Finish Line

Codey is responsible for... • ~~User input & movement~~ • ~~Camera movement~~ • Resetting after colliding with a hazard • Spawning & using powerups • Updating and displaying the powerup UI
The finish line is responsible for... • Letting the game know when Codey has crossed the finish line
The checkpoint is responsible for... • Letting the game manager know when Codey has passed a checkpoint • Ensuring a checkpoint can't be "passed" more than once per lap • Resetting
The game manager is responsible for... • Counting checkpoints as they're passed by Codey • Checking if all the checkpoints have been passed when Codey crosses the finish line • Telling checkpoints when to reset • The time limit and race countdown • Checking if Codey has completed all the necessary laps • Updating and displaying the game UI • Displaying the game over UI
BoxSpawner is responsible for... • Spawning and placing the Mystery Boxes along the track
The Mystery Boxes are responsible for... • Their rotation • Disappearing when Codey collides with a box • Respawning after a set amount of time
The Powerups are responsible for... • Their movement • Destroying themselves after colliding with a Hazard
The Hazards are responsible for... • Destroying themselves

Requirement #6: Create The Finish Line

Create a finish line that prints a message to the console when crossed by Codey.

- ☐ Create a new scene **finish_line.tscn** in the **Scenes > Objects** folder with an **Area3D** as the root node.
  - Use the **finish_line.png** texture in the **Assets > Textures** folder.
  - Adjust the **size** of the finish line, not the scale. Refer to the ground setup in the previous section as needed.
  - Add the **finish_line.tscn** scene to Racetrack as a **child** to **Background**. **Note**: it may be helpful to adjust the size of the finish line in the finish_line.tscn scene, then look at the size of the finish line on the racetrack and make further adjustments in the finish_line.tscn scene as necessary.
- ☐ Attach a new script, **finish_line.gd** to the root node of the **finish_line.tscn**. Code the script to print a message to the console when Codey's area, which is in the node group Player, passes the finish line.

Pause for **Ninja Stop #2**!

- Does a message print to the console when Codey crosses the finish line?

Reminder: Save your work!

Requirement #6: Hints

- ☐ Does the **finish_line.tscn** scene contain an **Area3D**, **CollisionShape3D** & **MeshInstance3D**?
- ☐ Is a **QuadMesh** used as the **MeshInstance3D** with **Orientation** set to **Y Face**?
- ☐ Is the **CollisionShape3D** tall enough for Codey to collide with?
- ☐ Is the **area_entered** signal connected to the **finish_line.gd** script?
- ☐ Is the `is_in_group()` method used?

Requirement #6: Resources

- Refer to signals and groups in BB Activity 07: Polyrun
- Godot 4.4 Area3D Documentation
- Godot 4.4 Groups Tutorial

Requirement #7: Create the Checkpoints

Codey is responsible for...

- ~~User input & movement~~
- ~~Camera movement~~
- Resetting after colliding with a hazard
- Spawning & using powerups
- Updating and displaying the powerup UI

The finish line is responsible for...

- ~~Letting the game know when Codey has crossed the finish line~~

The checkpoint is responsible for...

- Letting the game manager know when Codey has passed a checkpoint
- Ensuring a checkpoint can't be "passed" more than once per lap
- Resetting

The game manager is responsible for...

- Counting checkpoints as they're passed by Codey
- Checking if all the checkpoints have been passed when Codey crosses the finish line
- Telling checkpoints when to reset
- The time limit and race countdown
- Checking if Codey has completed all the necessary laps
- Updating and displaying the game UI
- Displaying the game over UI

BoxSpawner is responsible for...

- Spawning and placing the Mystery Boxes along the track

The Mystery Boxes are responsible for...

- Their rotation
- Disappearing when Codey collides with a box
- Respawning after a set amount of time

The Powerups are responsible for..

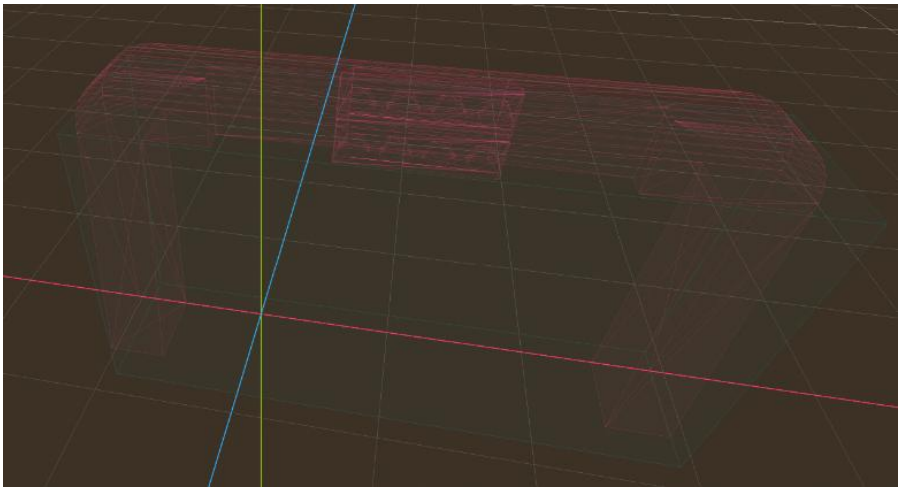
- Their movement
- Destroying themselves after colliding with a Hazard

Requirement #7: Create the Checkpoints

Add checkpoints to the game to prevent the player from crossing the finish line before completing the race.

Complete the checkpoint.tscn scene

- Open the **checkpoint.tscn** in the **Scene > Objects** folder. Finish building the scene so that:
  - Codey can't pass through the **overheadLights mesh**.
  - The Checkpoint can detect when Codey passes underneath it.



- Add the checkpoint.tscn scene to Racetrack to add in Checkpoints! Use a **Node3D** named **Checkpoints** to organize the checkpoints in the game, then reparent **FinishLine** so it's a **child** of **Checkpoints**.

Code the checkpoint.gd script

- ☐ Attach a new script **checkpoint.gd** to the root node of the **checkpoint.tscn** scene.
- ☐ Code the checkpoint.gd script to emit a signal when Codey passes the checkpoint. Include a condition to only emit the signal the first time the checkpoint has been passed and a print a message to the console when Codey passes the checkpoint. In the script, define the following:
    - A **signal** `passed_checkpoint`
    - A **variable** `passed` of type `bool` assigned to `false`
    - A **function** `player_entered()` that takes no parameters and returns void. The `player_entered()` function should emit the `passed_checkpoint` signal then print to the console if Codey passes the checkpoint for the *first* time.

Pause for **Ninja Stop #3**!

- Does a message print to the console when Codey passes a checkpoint?
- Can a checkpoint be passed more than once by Codey?

Reminder: Save your work!

Requirement #7: Hints

- ☐ Does the **checkpoint.tscn** scene contain an **Area3D**, **StaticBody3D** and **two CollisionShape3D** nodes?
- ☐ Was a **Trimesh Collision Shape** created from the **MeshInstance3D** node and **reparented** to the **StaticBody3D**?
- ☐ Is the **area_entered** signal connected to the checkpoint.gd script?
- ☐ Is the `is_in_group()` method used?
- ☐ Is the variable `passed` set to `true` after Codey has passed the checkpoint?

```
9
10  func player_entered() -> void:
11      # if not ???:
12          # ??? = true
13          # ###.emit()
14          # print("???")
15
16
17  func _on_area_entered(area: Area3D) -> void:
18      # if area.???("???")
19          # ???
```

Requirement #7: Resources

- Refer to Requirement #3: Set Up The Track Pieces
- Refer to Requirement #6: Create the Finish Line

Requirement #8: Code The Checkpoints

Codey is responsible for...

- ~~User input & movement~~
- ~~Camera movement~~
- Resetting after colliding with a hazard
- Spawning & using powerups
- Updating and displaying the powerup UI

The finish line is responsible for...

- ~~Letting the game know when Codey has crossed the finish line~~

The checkpoint is responsible for...

- ~~Letting the game manager know when Codey has passed a checkpoint~~
- ~~Ensuring a checkpoint can't be "passed" more than once per lap~~
- Resetting

The game manager is responsible for...

- Counting checkpoints as they're passed by Codey
- Checking if all the checkpoints have been passed when Codey crosses the finish line
- Telling checkpoints when to reset
- The time limit and race countdown
- Checking if Codey has completed all the necessary laps
- Updating and displaying the game UI
- Displaying the game over UI

BoxSpawner is responsible for...

- Spawning and placing the Mystery Boxes along the track

The Mystery Boxes are responsible for...

- Their rotation
- Disappearing when Codey collides with a box
- Respawning after a set amount of time

The Powerups are responsible for...

- Their movement
- Destroying themselves after colliding with a Hazard

The Hazards are responsible for...

- Destroying themselves

Requirement #8: Code The Checkpoints

Code the game to count the checkpoints as they're passed by Codey. All checkpoints will need to be passed to cross the finish line and win the game.

- ☐ In the **racetrack.tscn** scene, attach a new script **game_manager.gd** to the root node **Racetrack**.
- ☐ Code the script to find all the checkpoints and increase the number of checkpoints passed as Codey progresses around the track.
- ☐ Print a message to the console to show how many checkpoints have been passed as Codey moves around the track.
- ☐ In the script, the following will need to be defined:
  - A **variable** `checkpoints` of type `Array[Node]` assigned to an `empty array`
  - A **variable** `total_checkpoints` of type `int` assigned to `0`
  - A **variable** `checkpoints_passed` of type `int` assigned to `0`
  - The `_ready()` method
  - A **function** `_on_checkpoint_passed()` that takes no parameters and returns void

Pause for **Ninja Stop #4**!

- Does the script correctly track the checkpoints?
- Is the number of checkpoints "passed" updated as Codey moves around the track?

Reminder: Save your work!

Requirement #8: Hints

- ☐ Is a node in the **checkpoint.tscn** part of a **global node group** named **Checkpoint**?
- ☐ Is the `get_nodes_in_group()` method used to create an array of checkpoints in the game?
- ☐ Is the `size()` method used to get the total number of checkpoints in the game?
- ☐ Is the `passed_checkpoint` signal **connected** to the `_on_checkpoint_passed()` function?
- ☐ Is the number of `checkpoints_passed` increased when a checkpoint is passed?

```
6
7   func _ready() -> void:
8       # ??? = get_tree().???("???")
9       # total_checkpoints = ???.???
10
11      # for ??? in ???:
12          # ???.passed_checkpoint.???(???)
13
```

Requirement #8: Resources

- Refer to coding and connecting signals in SB Activity 06: Evil Fortress of Doctor Worm

Requirement #9: Code the Finish Line

Codey is responsible for...

- ~~User input & movement~~
- ~~Camera movement~~
- Resetting after colliding with a hazard
- Spawning & using powerups
- Updating and displaying the powerup UI

The finish line is responsible for...

- ~~Letting the game know when Codey has crossed the finish line~~

The checkpoint is responsible for...

- ~~Letting the game manager know when Codey has passed a checkpoint~~
- ~~Ensuring a checkpoint can't be "passed" more than once per lap~~
- Resetting

The game manager is responsible for...

- ~~Counting checkpoints as they're passed by Codey~~
- Checking if all the checkpoints have been passed when Codey crosses the finish line
- Telling checkpoints when to reset
- The time limit and race countdown
- Checking if Codey has completed all the necessary laps
- Updating and displaying the game UI
- Displaying the game over UI

BoxSpawner is responsible for...

- Spawning and placing the Mystery Boxes along the track

The Mystery Boxes are responsible for...

- Their rotation
- Disappearing when Codey collides with a box
- Respawning after a set amount of time

The Powerups are responsible for...

- Their movement
- Destroying themselves after colliding with a Hazard

The Hazards are responsible for...

- Destroying themselves

Requirement #9: Code the Finish Line

Code the finish line so it can only be "crossed" after all the checkpoints have been passed.

- ☐ In the **finish_line.gd** script, create a new signal `finish_line_crossed` and emit the signal when Codey crosses the finish line.
- ☐ Define a function `finish_line_crossed()` in the **game_manager.gd** script that prints a message to the console when the finish line is crossed if all the checkpoints have been passed by Codey. The function `finish_line_crossed()` takes no parameters and returns void.
- ☐ Define a variable `finish_line` that references the **FinishLine** node in the racetrack.tscn scene.
- ☐ Connect the `finish_line_crossed` signal to the `finish_line_crossed()` function.

Pause for **Ninja Stop #5**!

- Does the finish line message print to the console if Codey does not pass all the checkpoints?

Reminder: Save your work!

Requirement #9: Hints

- ☐ Is the `finish_line` variable used to connect the `finish_line_crossed` signal to the `finish_line_crossed()` function inside the `_ready()` method?
- ☐ Does the `finish_line_crossed()` function compare the values of `checkpoints_passed` and `total_checkpoints`?

Requirement #9: Resources

- Refer to Requirement #8: Code the Checkpoints.

Requirement #10: Code The Checkpoints UI

Codey is responsible for...

- ~~User input & movement~~
- ~~Camera movement~~
- Resetting after colliding with a hazard
- Spawning & using powerups
- Updating and displaying the powerup UI

The finish line is responsible for...

- ~~Letting the game know when Codey has crossed the finish line~~

The checkpoint is responsible for...

- ~~Letting the game manager know when Codey has passed a checkpoint~~
- ~~Ensuring a checkpoint can't be "passed" more than once per lap~~
- Resetting

The game manager is responsible for...

- ~~Counting checkpoints as they're passed by Codey~~
- ~~Checking if all the checkpoints have been passed when Codey crosses the finish line~~
- Telling checkpoints when to reset
- The time limit and race countdown
- Checking if Codey has completed all the necessary laps
- Updating and displaying the game UI
- Displaying the game over UI

BoxSpawner is responsible for...

- Spawning and placing the Mystery Boxes along the track

The Mystery Boxes are responsible for...

- Their rotation
- Disappearing when Codey collides with a box
- Respawning after a set amount of time

The Powerups are responsible for...

- Their movement
- Destroying themselves after colliding with a Hazard

The Hazards are responsible for...

- Destroying themselves

Requirement #10: Code The Checkpoints UI

Create a user interface and code the game_manager.gd script to update the user interface and display and update the number of checkpoints passed and the total number of checkpoints in the game.

- In the **racetrack.tscn** scene, use a **CanvasLayer** and a **Label** node named **CheckpointLabel** to create the user interface.
- In the **game_manager.gd** script, update the label text at the start of the game and when a checkpoint is passed. A variable `checkpoint_label` will need to be defined.

Pause for **Ninja Stop #6**!

- Does the user interface update the number of checkpoints passed as Codey moves around the track?

Reminder: Save your work!

Requirement #10: Hints

- The label UI can be customized under **Theme Overrides**.
- Is a variable `checkpoint_label` that references the **CheckpointLabel** node defined in the script?
- Is the `str()` method use to convert the `checkpoints_passed` and `total_checkpoints` variables when concatenating the label text?

Requirement #10: Resources

- Refer to UI creation and updating Labels in SB Activity 13: Food Frenzy Part 1.

Requirement #11: Add in Hazards

Codey is responsible for...

- ~~User input & movement~~
- ~~Camera movement~~
- Resetting after colliding with a hazard
- Spawning & using powerups
- Updating and displaying the powerup UI

The finish line is responsible for...

- ~~Letting the game know when Codey has crossed the finish line~~

The checkpoint is responsible for...

- ~~Letting the game manager know when Codey has passed a checkpoint~~
- ~~Ensuring a checkpoint can't be "passed" more than once per lap~~
- Resetting

The game manager is responsible for...

- ~~Counting checkpoints as they're passed by Codey~~
- ~~Checking if all the checkpoints have been passed when Codey crosses the finish line~~
- Telling checkpoints when to reset
- The time limit and race countdown
- Checking if Codey has completed all the necessary laps
- ~~Updating and displaying the game UI~~
- Displaying the game over UI

BoxSpawner is responsible for...

- Spawning and placing the Mystery Boxes along the track

The Mystery Boxes are responsible for...

- Their rotation
- Disappearing when Codey collides with a box
- Respawning after a set amount of time

The Powerups are responsible for...

- Their movement
- Destroying themselves after colliding with a Hazard

The Hazards are responsible for...

- Destroying themselves

Requirement #11: Add in Hazards

Add Hazards to the racetrack that restart the game when Codey collides with them.

- Add a **Node3D** named **Hazards** as a **child** to **Background** to organize the hazards. Then, add hazards to the track. The **hazard.tscn** scene can be found in the **Scenes > Objects** folder.
- In the **codey.tscn** scene, use the `reload_scene()` function to reload the scene when Codey collides with a Hazard.

Pause for **Ninja Stop #7**!

- Does the game reload when Codey collides with a Hazard?

Reminder: Save your work!

Requirement #11: Hints

- ☐ Is the **area_entered** signal connected to the codey.gd script?
- ☐ Is the `is_in_group()` method used?
- ☐ The **area_entered** signal is emitted from **Codey's Area3D** when another node's Area3D enters it. In the **hazard.tscn** scene, should the Area3D node or the Hazard node be part of the global **node group Hazard**?

Requirement #11: Resources

- Refer to Requirement #6: Create the Finish Line or Requirement #7: Create the Checkpoints

Pause for **Sensei Stop #2**!

Check in with a Code Sensei and confirm that requirements 6 – 11 are set up correctly.

Reminder: Save your work!

Requirement #12: Spawn Mystery Boxes

Codey is responsible for...

- ~~User input & movement~~
- ~~Camera movement~~
- ~~Resetting after colliding with a hazard~~
- Spawning & using powerups
- Updating and displaying the powerup UI

The finish line is responsible for...

- ~~Letting the game know when Codey has crossed the finish line~~

The checkpoint is responsible for...

- ~~Letting the game manager know when Codey has passed a checkpoint~~
- ~~Ensuring a checkpoint can't be "passed" more than once per lap~~
- Resetting

The game manager is responsible for...

- ~~Counting checkpoints as they're passed by Codey~~
- ~~Checking if all the checkpoints have been passed when Codey crosses the finish line~~
- Telling checkpoints when to reset
- The time limit and race countdown
- Checking if Codey has completed all the necessary laps
- Updating and displaying the game UI
- Displaying the game over UI

BoxSpawner is responsible for...

- Spawning and placing the Mystery Boxes along the track

The Mystery Boxes are responsible for...

- Their rotation
- Disappearing when Codey collides with a box
- Respawning after a set amount of time

The Powerups are responsible for...

- Their movement
- Destroying themselves after colliding with a Hazard

The Hazards are responsible for...

- Destroying themselves

Requirement #12: Spawn Mystery Boxes

Create a BoxSpawner to spawn groups of Mystery Boxes. Later, the boxes will be coded to provide Codey with Powerups. Multiple BoxSpawners will be used to place rows of Mystery Boxes at different positions along the track.

- ☐ Add a **Node3D** named **MysteryBoxes** as a **child** to **Background** to organize the **BoxSpawners**.
- ☐ Add a **Node3D** named **BoxSpawner1** as a **child** to **MysteryBoxes**, then find the **mystery_box.tscn** scene in the **Scenes > Objects** folder and add it as a child to **BoxSpawner1**. This node will be used to visualize the initial spawn point for the box. Place the spawner on the track by moving **BoxSpawner1**, not MysteryBox.
- ☐ Attach a new script **box_spawner.gd** to **BoxSpawner1**. The script will instantiate the mystery_box.tscn scene, add it to the scene tree, and position the Mystery Boxes in a row equal distance apart along the x-axis. Inside the box_spawner.gd script, define the following:
  - `@export` **variable** `box_scene` of type `PackedScene`
  - `@export` **variable** `number_of_boxes` of type `int`
  - `@export` **variable** `x_offset` of type `int` (space between the boxes so they don't spawn on top of each other)
  - The `_ready()` method
- ☐ Duplicate **BoxSpawner** to spawn more boxes along the track, then delete the **MysteryBox** child node once all the spawn points have been set.
  Note: The BoxSpawner node may need to be rotated to spawn boxes around different parts of the track.

Pause for **Ninja Stop #8**!

- Do the boxes spawn across the track?
- Do any of the BoxSpawner nodes need to be rotated?

Reminder: Save your work!

Requirement #12: Hints

- ☐ Is a `for`-loop used?
- ☐ Is the `range()` method used?
- ☐ Is a variable, like `box`, assigned to the instantiated box scene?
- ☐ Is the `add_child()` method used?
- ☐ Is `x_offset` multiplied by something when setting `position.x` to keep the spacing even between the boxes?
- ☐ Is the **mystery_box.tscn** scene assigned in the **Inspector**?


```
7
8   func _ready() -> void:
9       # for i in ???(number_of_boxes):
10          # var box = ???.???
11          # add_child(???)
12          # ???.position.x += ??? * i
13
```

Requirement #12: Resources

- Refer to instantiation in BB Activity 09: Dropping Bombs Part 3
- Refer to instantiation in SB Activity 08: World of Color

Requirement #13: Code The Mystery Boxes

Codey is responsible for...

- ~~User input & movement~~
- ~~Camera movement~~
- ~~Resetting after colliding with a hazard~~
- Spawning & using powerups
- Updating and displaying the powerup UI

The finish line is responsible for...

- ~~Letting the game know when Codey has crossed the finish line~~

The checkpoint is responsible for...

- ~~Letting the game manager know when Codey has passed a checkpoint~~
- ~~Ensuring a checkpoint can't be "passed" more than once per lap~~
- Resetting

The game manager is responsible for:

- ~~Counting checkpoints as they're passed by Codey~~
- ~~Checking if all the checkpoints have been passed when Codey crosses the finish line~~
- Telling checkpoints when to reset
- The time limit and race countdown
- Checking if Codey has completed all the necessary laps
- Updating and displaying the game UI
- Displaying the game over UI

BoxSpawner is responsible for:

- ~~Spawning and placing the Mystery Boxes along the track~~

The Mystery Boxes are responsible for:

- Their rotation
- Disappearing when Codey collides with a box
- Respawning after a set amount of time

The Powerups are responsible for:

- Their movement
- Destroying themselves after colliding with a Hazard

The Hazards are responsible for:

- Destroying themselves

Requirement #13: Code The Mystery Boxes

Code the Mystery Boxes to rotate, disappear when Codey collects a box and then respawn.

- In the **mystery_box.tscn** scene, attach a new script **mystery_box.gd** to the root node. Code the script to rotate the box around the y-axis, disappear (not delete) when Codey collides with a box, and reappear after a set amount of time. In the mystery_box.gd script, define the following:
  - A **variable** `rot_speed` of type `float` and assign it to a value.
  - A **variable** `respawn_time` of type `int` and assign it to a value.
  - The `_physics_ process()` method.
  - A function `player_entered()`, which takes 1 **parameter** `active` of type `bool` and returns void.
- Code the `player_entered()` function to make the box disappear or reappear using the `active` parameter and print to the console. Call the function when Codey collides with a box.

Pause for **Ninja Stop #9**!

- Do the boxes disappear and reappear?
- Can Codey still "collide" with boxes even when they can't be seen?

Reminder: Save your work!

Requirement #13: Hints

- ☐ Is the `rotate.y` parameter used and incremented?
- ☐ Is `delta` used as a multiplier for `rot_speed?`
- ☐ Is `await` used with the `create_timer()` method to keep the box invisible until the timeout signal is emitted?
- ☐ Is the **area_entered** signal connected to the **mystery_box.gd** script.

```
4
5   func _process(delta: float) -> void:
6       # rotation.??? += ??? * ???
7
```

Requirement #13: Resources

- Refer to timers and visibility in SB Activity 16: Food Frenzy Part 2
- Godot 4.4 rotate() Documentation
- Godot 4.4 create_timer() Documentation

Requirement #14 (Instructions): Deactivate the Mystery Box

15 Codey can still collide with a MysteryBox even when it can't be seen. Code the box to deactivate when it disappears.

In the **mystery_box.gd** script, define a variable `area_3d` that references the box's Area3D node.

16 In the `player_entered()` function, use `set_deferred()` to set the area_3d's `"monitoring"` and `"monitorable"` properties to the parameter `active`.

> `set_deferred():` Assigns value to the given property at the end of the current frame.
>
> **Parameters:**
>
> 1. `property (StringName)`: the object's property
> 2. `value (Variant):` the value being assigned
>
> `Returns (void)`

`set_deferred()` is used to update the properties at the end of the current frame to avoid interrupting other game operations. Using the parameter `active` to set the properties (and the box's visibility) allows the function to be used to activate and deactivate the box.

17 Values for `rot_speed` and `respawn_time` can't be set in the racetrack.tscn scene if the boxes are instantiated by the BoxSpawner. Export variables can be defined in the box_spawner.gd script so these values can be set in the racetrack.tscn scene. Setter functions can be created in the mystery_box.gd script and called in the box_spawner.gd script.

In the **mystery_box.gd** script, define the following functions:

- A function `set_rot_speed()` which takes 1 parameter `new_rot_speed` of type `float` and returns `void`. Inside the function, assign `rot_speed` to `new_rot_speed`.
- A function `set_respawn_time()` which takes 1 parameter `new_respawn_time` of type `int` and returns `void`. Inside the function, assign `respawn_time` to `new_respawn_time`.

18 Remove the value assignments at the top of the script for the `rot_speed` and `respawn_time` **variables**. Then, return to the **box_spawner.gd** script and define the following variables:

- `@export` variable `rot_speed` of type `float`
- `@export` var `respawn_time` of type `int`

19 Inside the `_ready()` method, call the `set_rot_speed()` and `set_respawn_time()` functions for each box that is instantiated, with the `rot_speed` and `respawn_time` variables as arguments.

20 Values for the `rot_speed` and `respawn_time` variables can now be assigned in the inspector for each **BoxSpawner** node.

Pause for **Sensei Stop #3**!

Check in with a Code Sensei and confirm that requirements 12 – 14 are working correctly.

Reminder: Save your work!

Requirement #15 (Instructions): Powerups

Codey is responsible for...

- ~~User input & movement~~
- ~~Camera movement~~
- ~~Resetting after colliding with a hazard~~
- Spawning & using powerups
- Updating and displaying the powerup UI

The finish line is responsible for:

- ~~Letting the game know when Codey has crossed the finish line~~

The checkpoint is responsible for...

- ~~Letting the game manager know when Codey has passed a checkpoint~~
- ~~Ensuring a checkpoint can't be "passed" more than once per lap~~
- Resetting

The game manager is responsible for:

- ~~Counting checkpoints as they're passed by Codey~~
- ~~Checking if all the checkpoints have been passed when Codey crosses the finish line~~
- Telling checkpoints when to reset
- The time limit and race countdown
- Checking if Codey has completed all the necessary laps
- Updating and displaying the game UI
- Displaying the game over UI

BoxSpawner is responsible for:

- ~~Spawning and placing the Mystery Boxes along the track~~

The Mystery Boxes are responsible for:

- ~~Their rotation~~
- ~~Disappearing when Codey collides with a box~~
- ~~Respawning after a set amount of time~~

The Powerups are responsible for:

- Their movement
- Destroying themselves after colliding with a Hazard

The Hazards are responsible for:

- Destroying themselves

Requirement #15 (Instructions): Powerups

21 The mystery box needs to be coded to give Codey a powerup when collecting a box.

In the **codey.gd** script, add an if-statement to the **area_entered** signal's receiver function under **TODO 1** to print a message to the console when Codey collides with a box. Using a global node group named **MysteryBox** will be helpful here.

22 In the **codey.tscn** scene, add a **Node3D** renamed to **PowerupManager** as a child to Codey, then attach a new script **powerup_manager.gd** to the node.

23 Add a **CanvasLayer** as a **child** to **Codey**, then a **TextureRect** renamed to **PowerupImage** as a **child** to **CanvasLayer**.

24 Set **Texture** to **powerup_0.png**, **Expand Mode** to **Ignore Size**, **Size to x: 175, y: 100** and anchor the image in the top right corner of the CanvasLayer. This will show what powerup Codey has available to use, or a mystery box if no powerup is available.

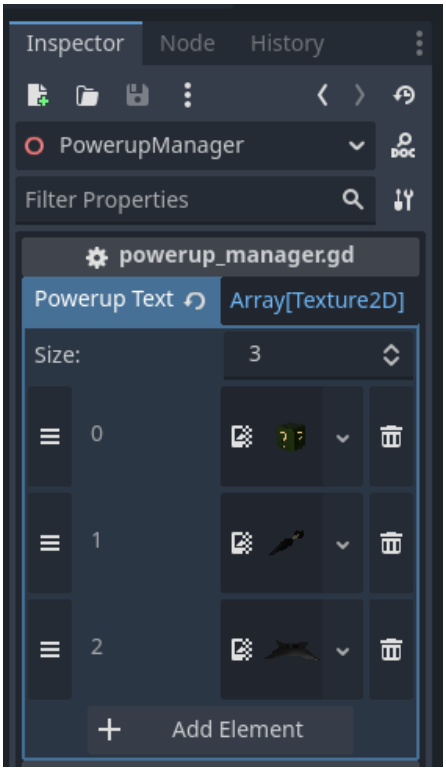
25 Inside the **powerup_manager.gd** script, define the following:

- **enum** `Powerups` with the values `NONE`, `BASIC`, `NAV_MESH`
- **variable** `current_powerup` of type `Powerups` assigned to `Powerups.NONE`
- **variable** `total_powerups` and assign it to the **size** of the `Powerups` enum
- `@onready` **variable** `powerup_image` set to the **PowerupImage** Node Path
- `@export` **variable** `powerup_textures` of type `Array[Texture2D]` assigned to an `empty array`
- `_ready()` method
- A function `set_image()` that takes 1 **parameter** `index` of type `int` and returns void.

26 Inside the `set_image()` function, update the `powerup_image` **texture** using the `index` parameter and the `powerup_textures` **array**.

27 Inside the `_ready()` method, call the `set_image()` function with the argument `current_powerup`.

28 Drag and drop the three powerup textures from the **Assets > Textures > Powerups** folder to add elements to the **Powerup Textures** array, then playtest the project. Try changing the value assigned to the `current_powerup` variable and see if the texture updates.



29 Define a new function `mystery_box_collected()` which returns void to select a random powerup when Codey collects a box. Inside the function:

- Define a **variable** `random_powerup` and assign it to a **random integer** in the **range** of `total_powerups`. The value associated with the key NONE should not be included in the range.
- Assign `current_powerup` to `random_powerup as Powerups`.
- Call the `set_image()` function with `current_powerup` as an argument to update the UI with the randomly selected powerup.

```
18
19  func mystery_box_collected() -> void:
20      # var random_powerup = randi_range(???, ??? - ???)
21      # ??? = ??? as ???
22      # ???
23
24
```

30 In the **codey.gd** script, find **TODO 2** at the top of the script and define a **signal** `hit_mystery_box`. Update the code for the signal's receiver method under **TODO 1** to emit the signal when Codey collides with a mystery box.

```
11
12  # ---------------
13  # TODO 2: hit_mystery_box signal
14  # ---------------
15
16
```

31 Return to the **powerup_manager.gd** script and create an `@onready` **variable** `player` for Codey. Then, connect the player's `hit_mystery_box` signal to the `mystery_box_collected()` function inside the `_ready()` method.

32 Define the following functions:

- A function `use_powerup()` that takes no parameters and returns `void`.
- A function `use_basic_powerup()` that takes no parameters and returns `void`.
- A function `use_nav_powerup()` that takes no parameters and returns `void`.

33 Inside the `use_powerup()` function, write a `match` statement for `current_powerup` that:

- `Returns` when `current_powerup` is `NONE`.
- Calls the `use_basic_powerup()` function when `current_powerup` is `BASIC`.
- Calls the `use_nav_powerup()` function when `current_powerup` is `NAV_MESH`.
- Outside of the match statement, set `current_powerup` back to `NONE`, then call the `set_image()` function with the parameter `current_powerup`.

```
27
28  func use_powerup() -> void:
29      # match ???:
30          # Powerups.???:
31              # ???
32          # ???:
33              # ???
34          # ???:
35              # ???
36      # ??? = ???.???
37      # ???
38
39  func use_basic_powerup() -> void:
40      pass
41
42  func use_nav_powerup() -> void:
43      pass
44
```

34 Use the code completion to define the `_input()` method which takes 1 **parameter** `event` of type `InputEvent` and returns `void`. Inside the method, check if the Input action **`"powerup"`** is pressed, then call the `use_powerup()` function. The powerup action in the Input Map is included with the starter code.

> `_input()`: Called when there is an input event.
>
> **Parameters:**
>
> 1. `event (InputEvent)`: the input event
>
> `Returns (void)`

Pressing the **E** key on the keyboard triggers this action. After using the powerup, the texture in the UI will return to the image of the mystery box.

Playtest the scene to see the powerup UI update when Codey collides with boxes and uses the powerups.

Requirement #16 (Instructions): Basic Powerup

Codey is responsible for...

- ~~User input & movement~~
- ~~Camera movement~~
- ~~Resetting after colliding with a hazard~~
- ~~Spawning & using powerups~~
- ~~Updating and displaying the powerup UI~~

The finish line is responsible for...

- ~~Letting the game know when Codey has crossed the finish line~~

The checkpoint is responsible for...

- ~~Letting the game manager know when Codey has passed a checkpoint~~
- ~~Ensuring a checkpoint can't be "passed" more than once per lap~~
- Resetting

The game manager is responsible for...

- ~~Counting checkpoints as they're passed by Codey~~
- ~~Checking if all the checkpoints have been passed when Codey crosses the finish line~~
- Telling checkpoints when to reset
- The time limit and race countdown
- Checking if Codey has completed all the necessary laps
- Updating and displaying the game UI
- Displaying the game over UI

BoxSpawner is responsible for...

- ~~Spawning and placing the Mystery Boxes along the track~~

The Mystery Boxes are responsible for...

- ~~Their rotation~~
- ~~Disappearing when Codey collides with a box~~
- ~~Respawning after a set amount of time~~

The Powerups are responsible for...

- Their movement
- Destroying themselves after colliding with a Hazard

The Hazards are responsible for...

- Destroying themselves

Requirement #16 (Instructions): Basic Powerup

35 The type of powerup Codey collects is now randomly generated, but they do not spawn into the scene yet. Code the Basic Powerup to spawn when used, move forward as if thrown and destroy itself when it collides with a Hazard or after a set amount of time has passed.

In the **powerup_manager.gd** script, define the following variables and assign their values in the Inspector for **PowerupManager**:

- `@export` **variable** `basic_scene` of type `PackedScene`
- `@export` **variable** `spawn_offset` of type `int`

36 Inside the `use_basic_powerup()` function:

- Define a **variable** `projectile`, assign it the **instantiated** `basic_scene` and add it as a child the main scene by using the `get_tree()` method and the `current_scene` property.
- Define a variable `spawn_position` and assign it to the player's `origin + basis.z * spawn_offset`. Be sure to use `global_transform` for the transforms position (**origin**) and rotation & scale (**basis**) in the **World3D** space.
- Set the `origin` for projectile to `spawn_position`
- Set the `basis` for projectile to the **player's** `basis`

```
41
42  func use_basic_powerup() -> void:
43      # var ??? = ???.???
44      # ???.current_scene.???(???)
45      # var ??? = ???.global_transform.??? + ???.???.???.z * ???
46      # projectile.???.??? = ???
47      # ???.???.??? = player.???.???
48
```

37 Playtest the game and use a basic powerup to see the powerup spawn in the scene. Codey will need to move after the powerup is spawned so it can be seen.

38 Open the **basic_powerup.tscn** scene and attach a new script **basic_powerup.gd** to the root node **BasicPowerup**. Inside the script, define the following:

- `@export` **variable** `speed` of type `float`
- `@export` **variable** `lifetime` of type `float`
- The `_physics_process()` method
- The `_ready()` method

39 Inside the `_ready()` method, create and connect a timer using the `lifetime` variable which destroys the powerup after the timer has timed out.

```
3   @export var speed: float = 20
4   @export var lifetime: float = 7
5
6   func _ready() -> void:
7       # ???.???(lifetime).???.connect(???)
8
```

40 Inside the `_physics_process()` method, use the `global_translate()` method and the `speed` variable and `delta` parameter to move the powerup forward.

```
6  func _ready() -> void:
7      get_tree().create_timer(lifetime).timeout.connect(queue_free)
8
9  func _physics_process(delta: float) -> void:
10     # global_translate(global_transform.???.??? * ??? * speed)
11
12
```

> `global_translate()`: Moves the global (world) transformation by Vector3 offset
>
> **Parameters:**
>
> 1. `offset (Vector3)`: The Vector3 the object is moved by
>
> `Returns (void)`

41 Connect a signal through code or the interface and code the powerup to destroy itself if it hits a node in the Hazard group.

42 In the **hazard.tscn** scene, connect a signal and code a **hazard.gd** script to destroy the **Hazard** when it collides with a node in the global group **Powerup**.

Pause for **Sensei Stop #4**!

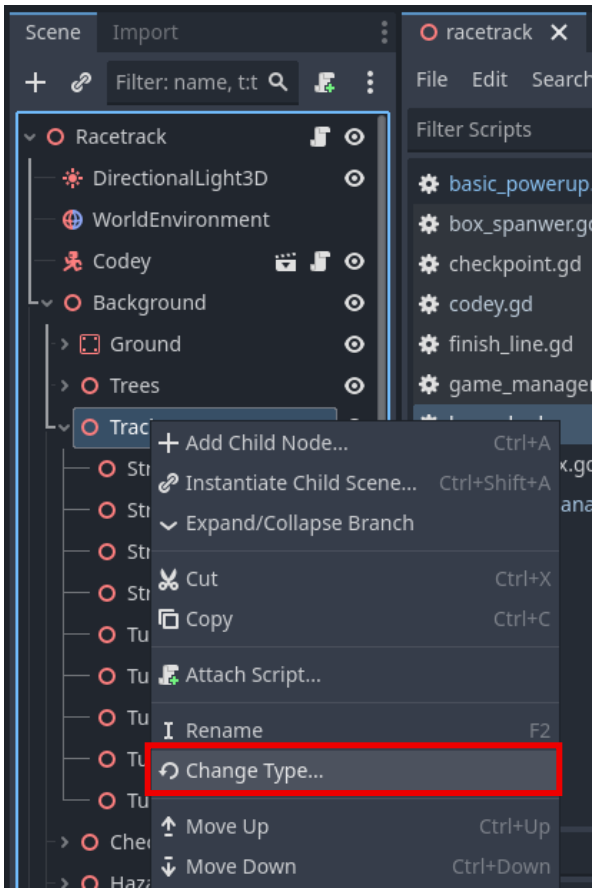
Check in with a Code Sensei and confirm that requirements 15 – 16 are set up correctly.

Reminder: Save your work!

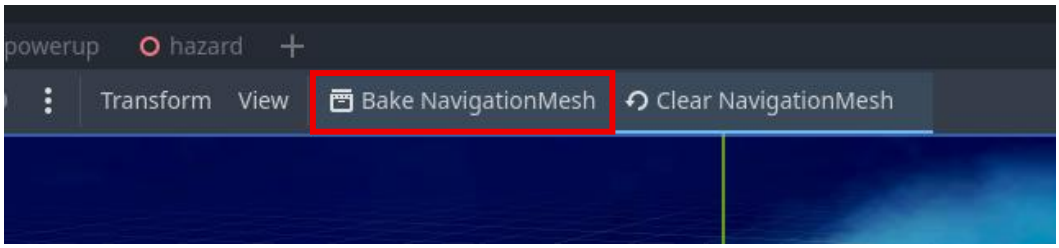
Requirement #17 (Instructions): Navigation Powerup

43 Code the Navigation Powerup to find and destroy the hazard closest to Codey.

In the **racetrack.tscn** scene, right-click on **Track** and select **Change Type**. Change the **Node3D** to a **NavigationRegion3D** and give it a new **NavigationMesh**.



44 Select the **Track NavigationRegion3D** and **Bake** the **NavigationMesh**.

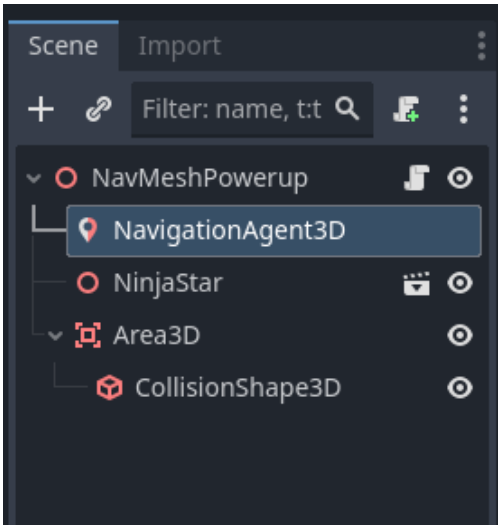


45 In the powerup_manager.gd script, define a **variable** `nav_scene` for the **nav_mesh_powerup.tscn** scene. Then, code the `use_nav_powerup()` function to spawn the powerup as a child of the `nav_region`, *not* the `current_scene`.

Refer to the `use_basic_powerup()` function as needed.

46 In the **nav_mesh_powerup.tscn** scene, add a **NavigationAgent3D** as a child to **NavMeshPowerup**.

Then, attach a new script **nav_mesh_powerup.gd** to the scene's root node.



47 Inside the nav_mesh_powerup.gd script, define the following:

- **variable** `agent` assigned to the **NavigationAgent3D** in the scene
- `@export` **variable** `speed` of type `float` assigned to a value
- **variable** `target` of type `Area3D` assigned to `null`.
- A function `_find_target()`, which takes no parameters and returns `void`

48

Inside the `_find_target()` function, define a variable `obstacles` and assign it to nodes in the **Hazard** group.
Then, define a variable `closest_dist` of type `float` and use `:=` to assign it to `INF`.

`INF` is a constant that represents positive infinity. Setting closest distance to `INF` ensures that any target will be closer than the starting value for `closest_dist`.

```
1   extends Node3D
2
3   @onready var agent: NavigationAgent3D = $NavigationAgent3D
4   @export var speed: float = 10.0
5   var target: Area3D = null
6
7   func _find_target() -> void:
8       # var obstacles = ???
9       # var closest_dist := INF
10
```

49

Use a `for`-loop to iterate through the array obstacles. For each obstacle, define a variable `dist` and assign it to the distance between the powerup and the node.
Use the `distance_to()` method and the `global_position` property.

Then, if `dist` is less than `closest_dist`, update `closest_dist` and `target`.

```
6
7   func _find_target() -> void:
8       # var obstacles = ???
9       # var closest_dist := INF
10      # for ??? in ???:
11          # var dist = ???.distance_to(???.???)
12          # if ??? < ???:
13          # ??? = ???
14          # ??? = ???
15
```

50 Outside the `for` loop, use the `set_target_position()` and the `look_at()` methods to give the agent a `target` if a target exists. Destroy the powerup if a target does not exist.

```
6
7   func _find_target() -> void:
8       # var obstacles = ???
9       # var closest_dist := INF
10      # for ??? in ???:
11          # var dist = ???.distance_to(???.???)
12          # if ??? < ???:
13          # ??? = ???
14          # ??? = ???
15
16      # if ???:
17          # agent.???(???.???)
18          # look_at(???.global_positIon)
19      # else:
20          # ???
21
```

Then, define a `_ready()` method and use `call_deferred()` to call the `_find_target()` function.

51 Define the **_physics_process()** method. Inside the method, if there is no target, **return**.

Then, use the **`is_navigation_finished()`** method, which returns true if the agent's navigation is finished, to check if agent has reached its target. If the target has been reached, destroy the powerup and return.

```
24
25  func _physics_process(delta: float) -> void:
26      # if ??? == ???:
27          # ???
28
29      # if ???.is_navigation_finished():
30          # ???
31          # ???
32
```

`is_navigation_finished()`: Returns true if the agent's navigation has finished. If the target is reachable, navigation ends when the target is reached. If the target is unreachable, navigation ends when the last point of the path is reached.

`Parameters: None`

`Returns (bool)`: returns true if the agent's navigation has finished.

52 Inside `_physics_process(),` use the `get_next_path_position()` method to get a `destination` for agent and the `normalized()` method to get the `direction`. Defining variables for destination and direction may be helpful here.

Then, use the `look_at()` method to point agent at the target and `global_translate()` to move the agent towards the target.

```
24
25  func _physics_process(delta: float) -> void:
26      # if ??? == ???:
27          # ???
28
29      # if ???.is_navigation_finished():
30          # ???
31          # ???
32
33      # var destination = ???.???
34      # var direction = (??? - global_position).???
35      # look_at(??? + ???, ???)
36      # global_translate(??? * ??? * ???)
37
```

53 In the Inspector for Codey, assign **Nav Region** to **Track**.

Playtest the game. Press **E** to use the powerup and watch it navigate towards a hazard!

54 The hazard and powerup don't destroy themselves when they collide.

In the **nav_mesh_powerup.gd** script, code the powerup to destroy itself when it collides with a **Hazard**.

Then, make the necessary adjustments so the hazard is destroyed when hit with the powerup.

55

The Ninja star doesn't rotate when thrown.

Attach a new script named **spin.gd** to **NinjaStar** in the **nav_mesh_powerup.tscn** scene.

Define an `@export` variable `rot_speed` of type `float` and a `_process()` method. Code the Ninja star to rotate around the `y` axis using the `rot_speed()` method.

```
3   @export var rot_speed: float = 20
4
5   func _process(delta: float) -> void:
6       # rotation.??? += ??? * ???
7
```

Pause for **Sensei Stop #5**!

Check in with a Code Sensei and confirm that requirement 17 is working correctly.

Reminder: Save your work!

Requirement #18: Add a Race Countdown

Codey is responsible for...

- ~~User input & movement~~
- ~~Camera movement~~
- ~~Resetting after colliding with a hazard~~
- ~~Spawning & using powerups~~
- ~~Updating and displaying the powerup UI~~

The finish line is responsible for...

- ~~Letting the game know when Codey has crossed the finish line~~

The checkpoint is responsible for...

- ~~Letting the game manager know when Codey has passed a checkpoint~~
- ~~Ensuring a checkpoint can't be "passed" more than once per lap~~
- Resetting

The game manager is responsible for...

- ~~Counting checkpoints as they're passed by Codey~~
- ~~Checking if all the checkpoints have been passed when Codey crosses the finish line~~
- Telling checkpoints when to reset
- The time limit and race countdown
- Checking if Codey has completed all the necessary laps
- Updating and displaying the game UI
- Displaying the game over UI

BoxSpawner is responsible for...

- ~~Spawning and placing the Mystery Boxes along the track~~

The Mystery Boxes are responsible for...

- ~~Their rotation~~
- ~~Disappearing when Codey collides with a box~~
- ~~Respawning after a set amount of time~~

The Powerups are responsible for...

- ~~Their movement~~
- ~~Destroying themselves after colliding with a Hazard~~

The Hazards are responsible for...

- ~~Destroying themselves~~

Requirement #18: Add a Race Countdown

Add a race countdown so Codey can't move until the countdown has finished.

- ☐ The codey.gd script contains a function `start_game()` which takes parameter `start` of type `bool`. This function updated the `game_start` variable of type `bool` on the codey.gd script. When `game_start = true`, Codey can move. When `game_start = false`, Codey can't move.
  In the codey.gd script, find **TODO 3** inside the `_ready()` method and remove or comment out the call to the `start_game()` function.
- ☐ In the racetrack.tscn scene, add a **Label** named **CountdownLabel** as a child to **CanvasLayer** and design the UI for the countdown timer.
- ☐ Code the game_manager.gd script to countdown to the game start, update the UI as needed, and start the game by calling the `start_game()` function with `game_start` as the argument when the countdown has ended so Codey can move. In the game_manager.gd script, define the following:
  - A variable `player` that references **Codey** in the racetrack.tscn scene
  - A variable `countdown_label` that references the **CountdownLabel** node
  - A variable `countdown_timer` of type `float` assigned to a value
  - A variable `game_start` of type `bool` assigned to `false`
  - The `_process()` method
  - A function `_on_timeout()` that takes no parameters and returns `void`. **Note**: a timer will need to be created (either through code or with a Timer node) and connected to the `_on_timeout()` function.

Pause for **Ninja Stop #10**!

- Can Codey move before the countdown ends?
- Does the countdown disappear when the timer finishes?

Reminder: Save your work!

Requirement #18: Hints

- ☐ Is the **CountdownLabel** text updated when the game starts?
- ☐ Is the `str()` method used?
- ☐ Is the value of `countdown_time` rounded using `roundi()` before updating the UI?
- ☐ Is the value of `countdown_time` decremented by `delta`?

```
func _process(delta: float) -> void:
    # ??? -= delta
    # ???.??? = str(roundi(???))
```

- ☐ Is the `timeout` **signal** connected?
- ☐ Is `game_start` set to `true` before the `start_game()` function is called?

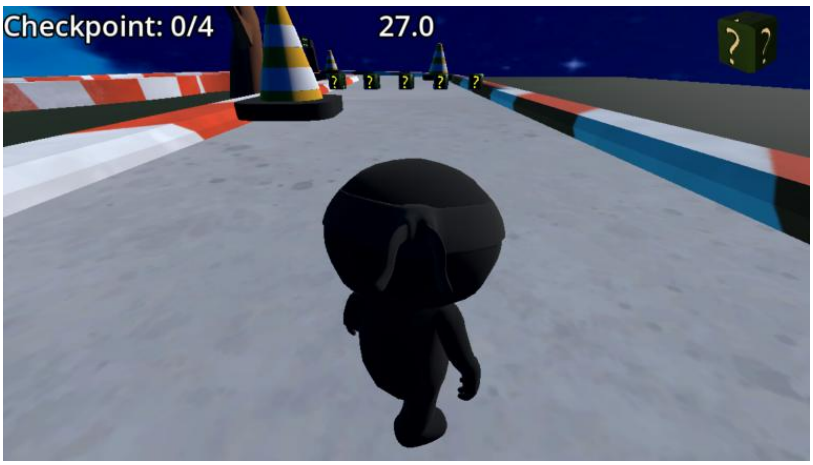
Requirement #18: Resources

- Godot 4.4 roundi() Documentation
- Refer to timers and visibility in SB Activity 16: Food Frenzy Part 2

Requirement #19: Add a Time Limit to the Race

Add a time limit to the game where time is added as the checkpoints are passed.

- ☐ Add a **TimerLabel** Label to the UI.
- ☐ Code the game_manager.gd script to decrease how much time is left in the game, update the **TimerLabel** and add time to the game when a checkpoint is passed. In the game_manager.gd script, define the following:
  - `@export` variable `time_increment` of type `float`
  - `@onready` variable `timer_label` for the **TimerLabel** node
  - A variable `time_left` of type `float` assigned to `0.0`
  - The `_process()` method
    Note: use the value assigned to `time_increment` as `time_left` when the game starts
- ☐ Inside the `_process()` method, only decrease the value of `time_left` if `game_start = true`.



Pause for **Ninja Stop #11**!

- Do does the time limit appear when the countdown ends?
- Does Codey get more time after passing a checkpoint?

Reminder: Save your work!

Requirement #19: Hints

- ☐ Is the `timer_label` hidden at the start of the game and visible once the countdown timer has timed out?
- ☐ Is `time_left` decremented once the game has started?
- ☐ Are the `str()` and `roundi()` or `roundf()` methods used when updating the timer label?

```
func _process(delta: float) -> void:
    countdown_timer -= delta
    countdown_label.text = str(roundi(countdown_timer))
    # if ???:
        # ??? -= delta
        # ???
```

- ☐ Is `time_left` incremented when a checkpoint is passed?

Requirement #19: Resources

- Refer to Requirement #18: Add A Race Countdown
- Godot 4.4 rounf() Documentation

Requirement #20: Lap Counter

Codey is responsible for...

- ~~User input & movement~~
- ~~Camera movement~~
- ~~Resetting after colliding with a hazard~~
- ~~Spawning & using powerups~~
- ~~Updating and displaying the powerup UI~~

The finish line is responsible for...

- ~~Letting the game know when Codey has crossed the finish line~~

The checkpoint is responsible for...

- ~~Letting the game manager know when Codey has passed a checkpoint~~
- ~~Ensuring a checkpoint can't be "passed" more than once per lap~~
- Resetting

The game manager is responsible for...

- ~~Counting checkpoints as they're passed by Codey~~
- ~~Checking if all the checkpoints have been passed when Codey crosses the finish line~~
- Telling checkpoints when to reset
- ~~The time limit and race countdown~~
- Checking if Codey has completed all the necessary laps
- Updating and displaying the game UI
- Displaying the game over UI

BoxSpawner is responsible for...

- ~~Spawning and placing the Mystery Boxes along the track~~

The Mystery Boxes are responsible for...

- ~~Their rotation~~
- ~~Disappearing when Codey collides with a box~~
- ~~Respawning after a set amount of time~~

The Powerups are responsible for...

- ~~Their movement~~
- ~~Destroying themselves after colliding with a Hazard~~

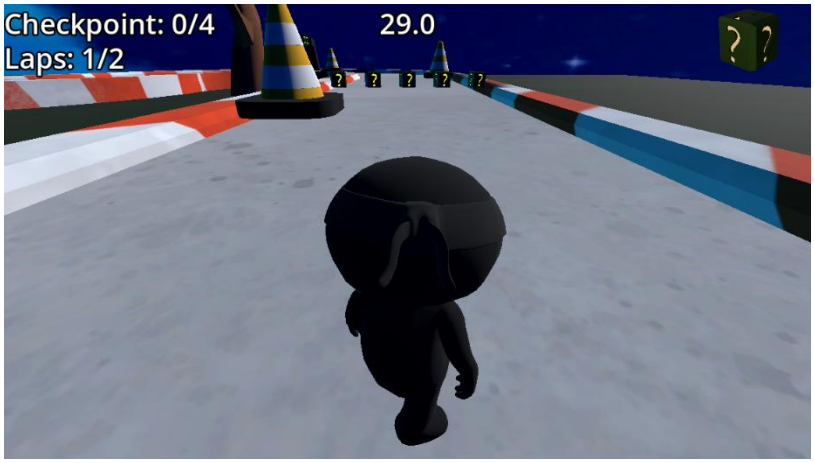
The Hazards are responsible for...

- ~~Destroying themselves~~

Requirement #20: Lap Counter

Make Codey complete multiple laps around the track before winning the game. Add UI components to show how many laps have been completed and how many are remaining.

- Add a **LapLabel** Label to the UI.
- Code the **game_manager.gd** script to check if Codey has completed all the laps when the finish line is crossed. If not all the laps are completed, reset the checkpoints, increase the number of laps complete and the value of `time_left`. In the **game_manager.gd** script, define the following:
  - A variable `lap_label` that references the **CountdownLabel** node
  - `@export` variable `total_laps` of type `int`
  - A variable `current_lap` of type `int` assigned to `0`
- In the **checkpoint.gd** script, define a function `reset()` that resets the value of `passed` to `false`. Call the function in the **game_manager.gd** script if the finish line is crossed but not all the laps are complete to reset the checkpoints so Codey can pass them again.



Pause for **Ninja Stop #12**!

- Do the checkpoints reset at the start of a new lap?
- Can Codey "cross" the finish line before all the laps are complete?

Reminder: Save your work!

Requirement #20: Hints

- [ ] Are the label texts updating as expected during the game?
- [ ] Is an if-statement used in the `finish_line_crossed()` function to check if `current_lap` is greater than `total_laps`?
- [ ] Are the values of `current_lap`, `checkpoints_passed` and `time_left` updated if Codey needs to complete more laps?
- [ ] Is a `for`-loop used to reset all the checkpoints in the checkpoints array?

```
func finish_line_crossed() -> void:
    if checkpoints_passed >= total_checkpoints:
        current_lap += 1
        lap_label.text = "Laps: " + str(current_lap) + "/" + str(total_laps)
        # if ??? >= ???:
            # ???
        # else:
            # ??? = 0
            # ??? += ???
            # checkpoint_label.text = ???
            # for ???:
                # ???
```

Requirement #20: Resources

- Refer to Requirements 8 - 10

Requirement #21: Game Over User Interface

Codey is responsible for...

- ~~User input & movement~~
- ~~Camera movement~~
- ~~Resetting after colliding with a hazard~~
- ~~Spawning & using powerups~~
- ~~Updating and displaying the powerup UI~~

The finish line is responsible for...

- ~~Letting the game know when Codey has crossed the finish line~~

The checkpoint is responsible for...

- ~~Letting the game manager know when Codey has passed a checkpoint~~
- ~~Ensuring a checkpoint can't be "passed" more than once per lap~~
- ~~Resetting~~

The game manager is responsible for...

- ~~Counting checkpoints as they're passed by Codey~~
- ~~Checking if all the checkpoints have been passed when Codey crosses the finish line~~
- ~~Telling checkpoints when to reset~~
- ~~The time limit and race countdown~~
- ~~Checking if Codey has completed all the necessary laps~~
- ~~Updating and displaying the game UI~~
- Displaying the game over UI

BoxSpawner is responsible for...

- ~~Spawning and placing the Mystery Boxes along the track~~

The Mystery Boxes are responsible for...

- ~~Their rotation~~
- ~~Disappearing when Codey collides with a box~~
- ~~Respawning after a set amount of time~~

The Powerups are responsible for...

- ~~Their movement~~
- ~~Destroying themselves after colliding with a Hazard~~

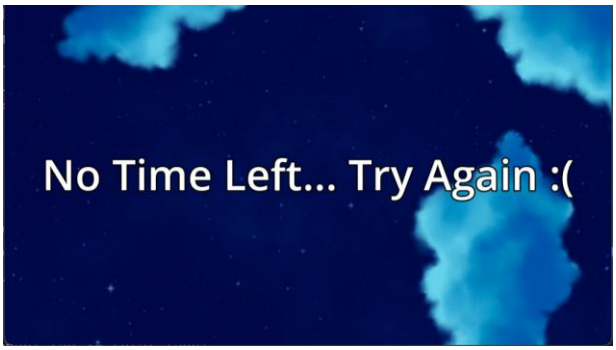
The Hazards are responsible for...

- ~~Destroying themselves~~

Requirement #21: Game Over User Interface

Design game over UI that appears when Codey wins the race and a separate UI that appears when time runs out and Codey loses the race.

- In the **game_manager.gd** script, define the following:
  - A function `on_game_win()` and a function `on_game_lose()`
    OR
  - A function `on_game_end()`
    Note: In the game_manager.gd script, setting `game_start` to `false` and stopping Codey's movement when the `time_left` falls below `0` may be helpful.
- For the game over UI, each screen must contain at least:
  - Game over text
  - A background image (the sky textures in the Assets > Textures > Sky folder can be used with a TextureRect node if needed)



Pause for **Sensei Stop #6**!

Check in with a Code Sensei before moving on and reflect on the following:

- How were collisions with other game objects checked and coded?
- Why might Codey be responsible for spawning the powerups instead of the game manager?
- How might game design affect the overall experience when creating a racing game?

Reminder: Save your work!

Requirement #21: Hints

- Where should the `on_game_win()` and `on_game_lose()` or `on_game_end()` function(s) be called?
- Does the `start_game()` function in the codey.gd script need to be called?
- Are any scenes being switched to display the game over UI?
- How might the powerup UI be hidden in the codey.tscn scene when the game ends?

Requirement #21: Resources

- Refer to UI design and changing scenes in SB Activity 15: Amazing Ninja Worlds Part 3
- Refer to UI design in SB Activity 13: Food Frenzy Part 1
- Refer to UI design in SB Activity 16: Food Frenzy Part 2