## Header Information

* Module Name : Level1Gameplay1
* Project Title : Menkorakure
* Author: Abdelah Mohammed Aliy
* Date: 04/07/2025

## Module Description

**Purpose:**

The module manages the navigation flow of the game and the primary

gameplay interaction in a 2D platformer game made with the Godot

Engine. It features a main menu, a stage intro screen, and a gameplay

scene where the player utilizes an Engineer character to interact with a Monk NPC.

**What Problem Does It Solve?**

The module provides a structured game flow by:

* Displaying a primary menu to enable the player to start or exit the game.
* Displaying an introduction to a level to establish context for the gameplay.
* Enabling fundamental gameplay mechanics such as player movement, jumping, and NPC interaction to enable a fun experience.

**Where Does It Fit in the Overall System?**

This module is the game's backbone, handling the initial user interface

(main menu), opening sequence (introduction to the stage), and main

game loop (Engineer-Monk interaction). It serves as the entry point and

central gameplay component, with space to be expanded upon in the future

(new levels or gameplay mechanics).

## Dependencies

This module depends on the following:

* **Godot Engine (version 4.x):** The game development engine utilized to build and run the game.
* **Input Actions:** left, right, jump, interact: Defined in Godot's Input Map for player input (mapped to "left arrow","right arrow", "Space", and "E" keys, respectively).
* **Assets:** Sprite images of Monk and Engineer, located in res://LEVEL1/assets/.

## Folder & File Structure

The code is organized within the LEVEL1 folder, with scenes, scripts, and assets in

their respective subfolders.

res://

└── LEVEL1/

├── scenes/

│ ├── game\_play\_1.tscn

│ ├── engineer.tscn

│ ├── monk.tscn

│ ├── MainMenu.tscn

│ └── StageIntro.tscn

├── scripts/

│ ├── engineer.gd

│ ├── monk.gd

│ ├── main\_menu.gd

│ └── stage\_intro.gd

└── assets/

└── PNG/

**File Descriptions:**

Scenes Folder (LEVEL1/scenes/):

* MainMenu.tscn: Main menu scene, with the title "Level 1", "Play", and "Exit" buttons.
* StageIntro.tscn: Stage introduction scene, with "Stage-1" and "Talking with the Monk" for 3 seconds.
* game\_play\_1.tscn: Main game scene, with the Engineer, Monk, and UI.
* engineer.tscn: Scene for the Engineer character (player), instanced in game\_play\_1.tscn.
* monk.tscn: Scene for the Monk NPC, instanced in game\_play\_1.tscn.

Scripts Folder (LEVEL1/scripts/):

* main\_menu.gd: Script that processes the main menu button interaction.
* stage\_intro.gd: Script that processes the timed progress from the game scene introduction to the game scene.
* engineer.gd: Script that processes the Engineer's movement, jumping, and interaction logic.
* monk.gd: Script that processes the Monk's interaction logic, for example, prompt and dialogue display.

Assets Folder (LEVEL1/assets/):

* contains different images that used in the game

## Key Functions or Components

Here are the primary functions and features within this module.

1. **Function: \_on\_play\_button\_pressed** (in LEVEL1/scripts/main\_menu.gd)

* Purpose: Handles the "Play" button press in the main menu, transitioning to the stage introduction scene.
* Inputs: None (triggered by the pressed signal of the PlayButton).
* Outputs: Transitions the game to res://LEVEL1/Scenes/StageIntro.tscn.
* Logic Summary:
  + Uses get\_tree().change\_scene\_to\_file() to load the StageIntro.tscn scene when the "Play" button is pressed.

2. **Function: \_on\_exit\_button\_pressed** (in LEVEL1/scripts/main\_menu.gd)

* Purpose: Handles the "Exit" button press in the main menu, quitting the game.
* Inputs: None (triggered by the pressed signal of the ExitButton).
* Outputs: Exits the game.
* Logic Summary:
  + Calls get\_tree().quit() to close the game.

3**. Function: \_ready** (in LEVEL1/Scripts/stage\_intro.gd)

* Purpose: Manages the stage introduction sequence, displaying the stage title for 3 seconds before transitioning to the gameplay scene.
* Inputs: None (runs automatically when the scene loads).
* Outputs: Transitions the game to res://LEVEL1/Scenes/GamePlay1.tscn.
* Logic Summary:
  + Uses await get\_tree().create\_timer(3.0).timeout to wait for 3 seconds.
  + Calls get\_tree().change\_scene\_to\_file() to load the GamePlay1.tscn scene.

4. **Function: \_physics\_process** (in LEVEL1/scripts/engineer.gd)

* Purpose: Handles the Engineer’s movement, jumping, animation, and interaction with the Monk.
* Inputs:
  + delta: The time elapsed since the last frame (float).
  + Player input via Input actions (left, right, jump, interact).
* Outputs: Updates the Engineer’s position, velocity, and animation state.
* Logic Summary:
  + Updates the Engineer’s velocity based on input direction (left/right).
  + Applies gravity when not on the floor.
  + Handles jumping when the jump action is pressed.
  + Plays animations (default, running, jumping) based on the Engineer’s state.
  + Flips the sprite horizontally based on movement direction.
  + Calls the Monk’s interact() function if the Engineer is near the Monk and the interact action ("E" key) is pressed.

4. **Function: \_physics\_process** (in LEVEL1/scripts/engineer.gd)

* Purpose: Handles the Engineer’s movement, jumping, animation, and interaction with the Monk.
* Inputs:
  + delta: The time elapsed since the last frame (float).
  + Player input via Input actions (left, right, jump, interact).
* Outputs: Updates the Engineer’s position, velocity, and animation state.
* Logic Summary:
  + Updates the Engineer’s velocity based on input direction (left/right).
  + Applies gravity when not on the floor.
  + Handles jumping when the jump action is pressed.
  + Plays animations (default, running, jumping) based on the Engineer’s state.
  + Flips the sprite horizontally based on movement direction.
  + Calls the Monk’s interact() function if the Engineer is near the Monk and the interact action ("E" key) is pressed.

5**. Function: interact** (in LEVEL1/scripts/monk.gd)

* Purpose: Displays the Monk’s dialogue when the Engineer interacts with the Monk.
* Inputs: None (called by the Engineer when the "E" key is pressed).
* Outputs: Shows the dialogue for 3 seconds and updates UI visibility.
* Logic Summary:
  + Sets the DialogueLabel text to "You can find the manuscripts and the hidden chamber in the first room."
  + Shows the DialogueLabel and hides the PromptLabel.
  + Waits for 3 seconds using await get\_tree().create\_timer(3.0).timeout.
  + Hides the DialogueLabel and shows the PromptLabel if interaction is still possible.

6. **Function: \_on\_interaction\_zone\_body\_entered** (in LEVEL1/scripts/monk.gd)

1. Purpose: Detects when the Engineer enters the Monk’s interaction zone and shows the interaction prompt.
2. Inputs: body (Node2D) – The node that entered the interaction zone.
3. Outputs: Updates the can\_interact flag and shows the prompt.
4. Logic Summary:
   * Checks if the entering body is the Engineer.
   * Sets can\_interact to true and assigns the Monk as the Engineer’s nearby\_monk.
   * Displays the prompt ("To interact with the monk, press E") on the PromptLabel.

## Error Handling

**Invalid inputs:**

Player Input: The engineer.gd script uses Godot's Input system, which gracefully recovers from invalid inputs (e.g., if a key isn't pressed, the action just doesn't occur).

Node References: The monk.gd script uses @onready to ensure PromptLabel and DialogueLabel are available before they're accessed. If these nodes don't exist, Godot will throw a runtime error (caught in testing).

**Unexpected Behavior:**  
Signal Connection Issues: If signals (e.g., body\_entered, pressed) are not connected, input won't get the game to respond. This was mitigated by connecting signals using the Godot editor.

Scene Transition Calamities: If a scene file path is incorrect in MainMenu.gd or StageIntro.gd, Godot will crash with an error. This was remedied by checking paths throughout testing

## Testing and Validation

How the Module Was Tested:

**Manual Testing**:

* Tested the main menu by clicking the "Play" and "Exit" buttons to ensure they transition to the stage introduction and quit the game, respectively.
* Tested the stage introduction by verifying the 3-second timer and transition to the gameplay scene.
* Tested the gameplay scene by moving the Engineer, jumping, and interacting with the Monk to confirm the prompt and dialogue display correctly.

**Edge Cases Covered:**

* Player Movement: Tested edge cases like jumping while moving and interacting while jumping to ensure animations and physics behave correctly.
* Interaction Zone: Moved the Engineer in and out of the Monk’s interaction zone multiple times to confirm the prompt visibility toggles correctly.
* Scene Transitions: Tested incorrect scene paths to catch transition errors (fixed by correcting paths).

## Next Steps/To Do

* **Enhanced Dialogue System**: Add multiple dialogue lines for the Monk with a typing effect.
* **Sound Effects**: Add audio for jumping, interacting, and scene transitions.
* **Error Handling:** Add checks for missing animations or nodes to prevent crashes.

## Header Information

* Module Name : Level1Puzzle
* Project Title : Menkorakure
* Author: Walelign Tagese
* Date: 04/07/2025

### **PUzzle play level one**

### **1. Module Description**

## 1.1 purpose

This module is important since it is the core gameplay system for a 4X4 sliding puzzle game. It has different features which include:

* Tile management: this feature manages shuffling, movement validation and win detection.
* Time constraint: set’s 3-minutes countdown with automatic game termination and transitioning to the game UI.
* Dynamic UI: Menu overlays for win/loss states and player actions.

## 1.2 Role in project

Inputs:

* Mouse clicks (tile selection and movement)
* Timer events (from the control.gd)

Outputs:

* Visible UI at the win or loss conditions after the puzzle.
* Visible game puzzle chunks.
* Auto refilling and display of puzzle picture once solved.

Connections:

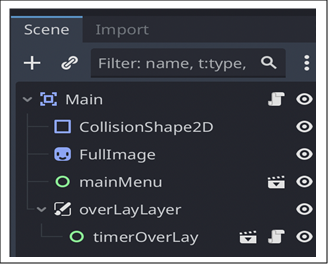
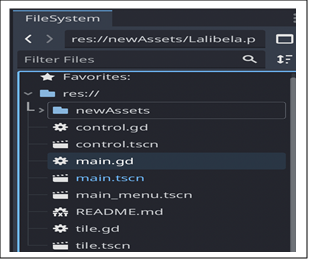
* Timer: name -> timerOverlay (control Godot node) emits timer\_completed to main.gd(main script)
* UI: name -> mainMenu it has buttons which trigger start\_game(),exist and additional items.

### **2. Dependencies**

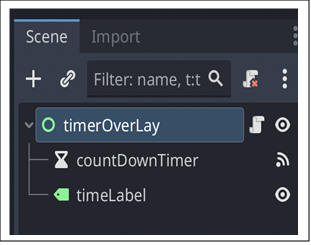
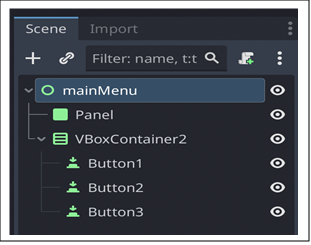
|  |  |  |
| --- | --- | --- |
| Dependency | Role | Usage Example |
| Godot 4 | Game engine | 2D rendering and input handling |
| Area2D | Tile collision detection. | Mouse click events on tiles. |
| CanvasLayer | UI isolation | Ensures timer/menu renders above game. |

### **3. Folder and File Structure**

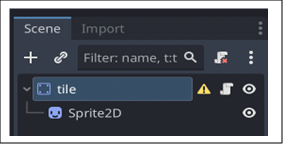
file structure of the main node(Area2D) Scene structure of the whole puzzle(.tscn)



file structure of timer(Control.gd) file structure of UI



file structure of tile(Area2D)



### **4. Key Functions & Components**

## 4.1 Core Functions(main.gd)

##### *start\_game()*

Purpose: it initializes the game which is the puzzle grid and shuffles tiles.

Key Steps:

* First it loads and splits the image into 16 tiles.
* Then it assigns textures and positions.
* Finally, it calls the shuffle\_tiles() method.

##### *shuffle\_tiles()*

The main purpose of this function is randomizing tile positions .

Logic:

* Performs 30 valid swaps by using the empty tile.
* Uses check\_neighbours(rows, cols) to validate moves.

##### *Check\_neighbours(rows, cols)*

Purpose : validating whether the clicked tile can move or not.

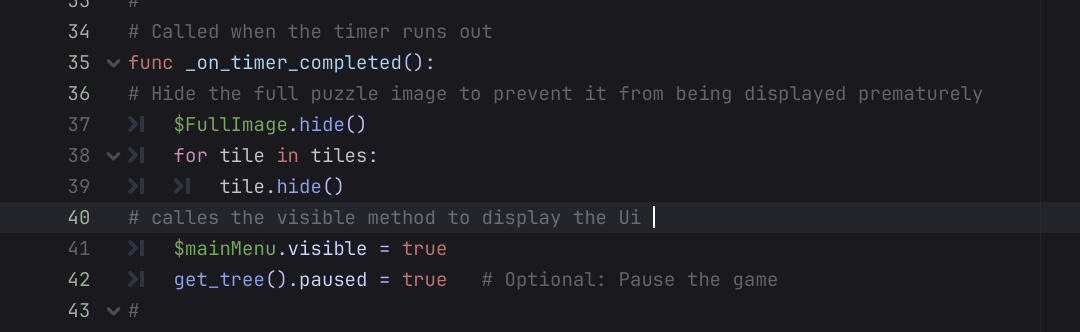
* As an input it takes the grid coordinates or (rows, cols).
* As an output it returns a Boolean value of true if the pressed tile is adjacent to empty space.

##### *\_on\_timer\_completed()*

Defines and handles timer expiry (time runs out).

Actions it preforms:

* Showing mainMenu(Ui which is going to be displayed ).
* Pauses gameplay.



### **4.2 Timer System (control.gd)**

##### *\_on\_timer\_timeout()*

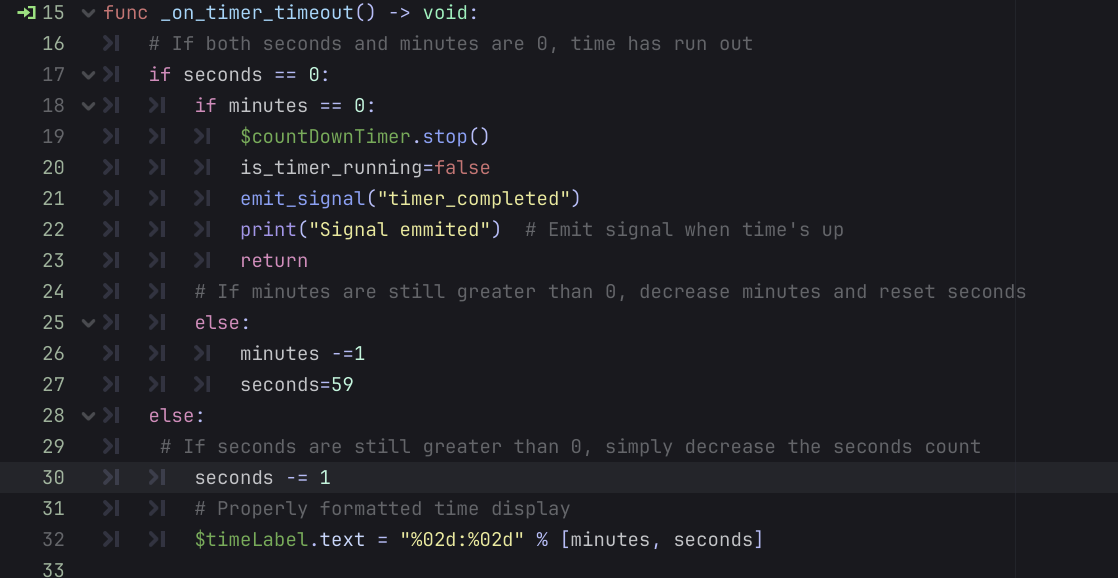
Purpose: counting down the time and emitting completion signal when the timer runs out.

Logic:

Decrements seconds/minutes.

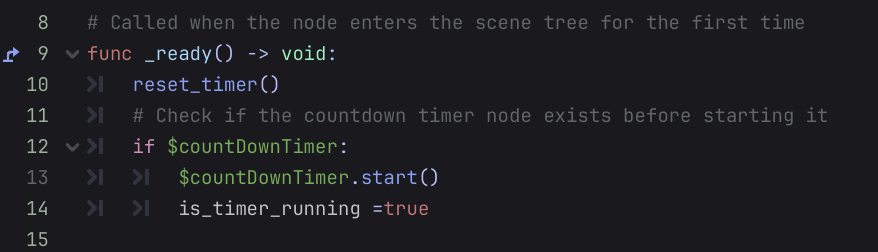
Formats time as MM:SS.

Emits timer\_completed at 00:00.

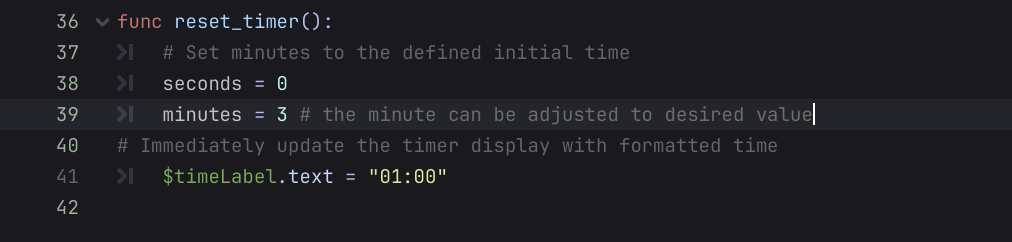


##### *\_ready()*

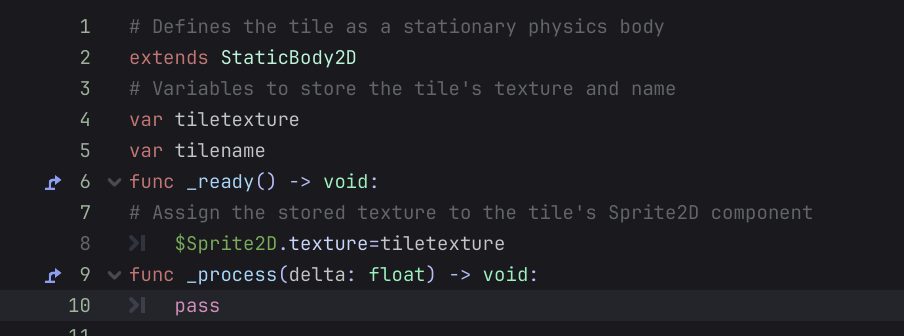
calls the reset\_timer function to ensure fresh start of the game.



##### *reset\_timer()*



### **4.3 tile(StaticBody2D)**



### **4.4 UI System(main.gd)**

Doesn’t have any script.

Inherited by the main.gd script as a node.

Has a button to emit signals to do different actions.

Has a Lalibela.png as an asset for the blared background.



### **4.5 Edge Cases**

## 1.Timer Expiry Mid -Swap

- Scenario: Player clicks a tile, and timer hits 00:00 before the game is completed.

- Expected: Game pauses immediately, UI displays and the tile movement halts.

- Test: During the game play the timer is forced to expiry to verify the case.

## 2. Rapid Consecutive Clicks

- Scenario: player clicks on tiles multiple times before the animation finishes.

- Expected: Input locks during and does take click at a time preventing invalid moves.

- Test: Rapid clicks to ensure only one move processes at a time.

### **4.6 Integration notes**

Pre requisites:

mainMenu must be a child node of Main.

a canvas layer should be added to the Main as a child node .

timer must be imported as a child node of the canvas layer.

## 1. Header Information

* Module Name**:** Player Module, World Building Module, Platform Scene, Sound Scene
* Project Title**:** 2D Platformer Adventure Game
* Author**:** Yohannes Mengistie
* Date**:** 4/7/2025

## 2. Module Descriptions

#### **1.Player Module**

This module controls the main character's behavior within the game.

Purpose: Manages player behavior such as movement, jumping, and animation.

Problem it Solves: Enables responsiveness and character control to interact with the game environment.

Role within System: Central component of game play experience; accepts input and interprets it as on-screen behaviors while handling collisions and interactions.

#### **2. World Building / Level Design Module**

Purpose: Develops and organizes the graphical and interactive layout of the game world by utilizing tile maps, platforms, and objects like chests.

Problem it Solves: Provides structured, walkable, and interactive areas to play.

Role in System: Acts as the main scene composition layer, communicating with both the art assets and gameplay logic (e.g., chest triggers or terrain edges).

#### **3.Platform Scene**

**Purpose:** Contains interactive elements such as static and moving platforms for player navigation.  
 **Problem it Solves:** Adds variety and challenge to player movement and physics.  
 **Role in System:** Supports player physics and enhances gameplay with timing/movement puzzles.

#### **4 . Sound Scene**

Purpose: Manages all audio features like background music and audio effects in the game like coin collection.

Problem it Solves: Enhances the player's immersion with sound effects and atmosphere.

Function in System: Acts upon game triggers and plays audio loops or effects.

## 3. Dependencies

* **Godot Engine (4.0+)** — Game development framework
* **TileMap** — For level layout and collision layers
* **AudioStreamPlayer** — For playing background music and effects
* **AnimationPlayer / AnimatedSprite2D** — For character and object animations
* **Input Events** — Keyboard control mapping
* **Timer** — For platform movement and timed events
* **Signals** — For inter-module communication (e.g., chest opened, coin collected)

## 4. Folder & File Structure

#### **Player/**

* AnimatedSprite2D: Manages player animation states (run, jump, idle)
* CollisionShape2D: Handles collision detection
* Camera2D: Follows player in the scene
* TimerScene: Triggers timed actions (e.g., respawn, death delay)
* Coin: Interactable collectible object
* Label: Displays information (e.g., score)

#### **GamePlay/**

* GameManager: Oversees game state, triggers, and win/loss conditions
* TileMap: Lays out level design including ground, obstacles, and triggers
* Player: Instance of the player controller
* Chest: Interactable object connected to quest or inventory system
* KillZone: Destroys player or resets position on collision
* Coins: Contains coin logic
* Platforms: Contains static and moving platform logic  
  + AnimationPlayer: Animates movement of platforms
* Clif, LiftRock, Cloud: Scene props and obstacles

## 5. Key Functions or Components

#### **Function: add\_point()**

* **Purpose:** Processes player collected coin
* **Output:** Updates the amount of coin the player collected
* **Logic Summary:** Checks body interaction, clear the coin from the visual, and applies to increasing the value by one.

#### **Function: on\_platform\_move()**

* **Purpose:** Moves platforms horizontally or vertically using a timer
* **Input:** Timer trigger
* **Output:** Updated platform position
* **Logic Summary:** Toggles direction and applies velocity based on timer interval

#### **Function: play\_sound(eventName)**

* **Purpose:** Plays sound based on gameplay event
* **Input:** String (e.g., "coin", "chest")
* **Output:** Plays respective AudioStream
* **Logic Summary:** Matches event with sound and plays using AudioStreamPlayer

#### **Signal: on\_body\_enterd**

* **Purpose:** Triggered when player interacts with a chest
* **Input:** Collision and interaction check
* **Output:** Sends signal to inventory or quest system

move\_and\_slide() is used to **move a physics body** (like CharacterBody2D or KinematicBody2D) while automatically handling:

* **Slopes**
* **Sliding along walls**
* **Stopping when hitting obstacles**
* **Floor detection** (so you can check is\_on\_floor())

It is the most common movement function in 2D platformers and top-down games.

## 6. Error Handling

* **Input Validation:** Uses Godot's built-in Input.is\_action\_pressed() to ensure only mapped actions are used
* **Physics & Collisions:** All moving platforms and the player use CollisionShape2D and KinematicBody2D to handle collisions robustly
* **Audio Errors:** Wrapped sound calls to avoid null audio streams if sound is not loaded
* **Fallbacks:** Player respawns at checkpoint if falling into KillZone or when health reaches zero

## 7. Testing & Validation

* **Manual Testing:** Each scene and module tested in isolation, as well as within full level context
* **Edge Cases:**  
  + Standing on moving platforms during direction change
  + Double-jumping prevention
  + Triggering multiple signals at once (e.g., coin + sound + UI update)
* **Known Issues:**  
  + Slight delay on first sound trigger if not preloaded
  + Player animation may glitch if switching directions mid-jump quickly

## 8. Next Steps / To-Do

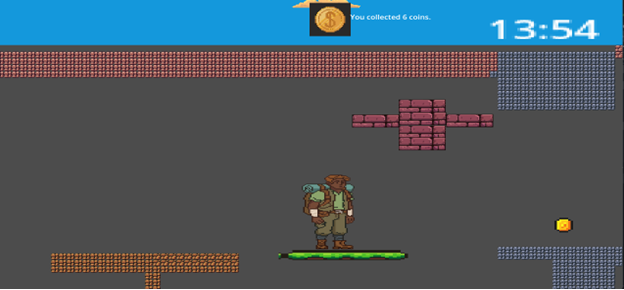
* Optimize player collision for smoother edge detection
* Add sound effect variations (e.g., randomize pitch for coins)
* Add checkpoint system with persistent progress

## 9. Integration Notes

* **Player Module** should be instanced into GamePlay scenes and positioned based on level design
* **Sound System** must be initialized at game start; triggered by events via signal
* **Chest** objects send signals to the InventoryManager or QuestSystem (not included here)
* **Moving platforms** require Timer nodes and are connected to the player's collision logic via signal/parenting for smooth movement

## The completed works:









**Level 2 Puzzle Code Documentation**

## 1. Header Information

* **Module Name:** Level\_2\_puzzle
* **Project Title:** Menkorakore
* **Author:** Anwar Andargie
* **Date:** April 7, 2025

## 2. Module Description

This module implements the core logic and interaction for the rotating ring puzzle in Level 2 of the Menkorakore game. Players interact with three concentric rings containing Ge'ez numerals, math symbols, and scientific formulas. The goal is to align the correct symbols in diagonal patterns to solve the puzzle and advance to the next level.

The puzzle:

* Presents 3 interactive rings (outer, middle, inner) with 8 segments each
* Each ring contains rotating labels with symbols (Ge'ez numerals, math symbols, and scientific formulas)
* Players can rotate rings using mouse drag or keyboard controls (Q/W, A/S, Z/X)
* Four valid combinations must be found in diagonal alignments
* A 60-second timer and success/failure feedback are provided

## 3. Dependencies

* Godot Engine 4.x
* Node2D, Label, Timer, Button, ColorRect nodes
* InputEvent handling for both mouse and keyboard controls
* Uses process\_mode and \_process() for animation and timer countdown

## 4. Folder & File Structure

/Level 2/

├── scripts/

│ └── ring\_puzzle.gd # Main logic script

├── ring\_puzzle.tscn # Main scene with all nodes

└── /assets/

└── champer.jpg # Background texture

Node Structure (in ring\_puzzle.tscn):

* + OuterRing, MiddleRing, InnerRing: Node2D nodes with segments
  + RingBorders: Line2D nodes for visual ring boundaries
  + ConfirmButton: Button to check combinations
  + Timer and Label: Countdown display
  + InstructionLabel: Shows dynamic feedback and progress
  + Background: Sprite2D for visual styling

## 5. Key Functions or Components

* setup\_rings()
  + Purpose: Initializes ring positions, segments, and labels
  + Logic: Creates segments with symbols, positions them evenly in a circle
  + Visual: Adds background colors and styling for each ring
* \_unhandled\_input(event)
  + Purpose: Handles both mouse and keyboard input for ring rotation
  + Inputs:
    - Mouse: Drag for rotation
    - Keyboard: Q/W (outer), A/S (middle), Z/X (inner)
  + Logic:
    - Tracks angle delta during drag
    - Snaps rotation to 45° segments
    - Keeps labels upright during rotation
* check\_solution()
  + Purpose: Evaluates diagonal combinations across all rings
  + Logic:
    - Checks 16 possible diagonal alignments (8 in each direction)
    - Validates against predefined combinations
    - Tracks solved equations
    - Shows success when all 4 combinations are found
* get\_symbol\_at\_index()
  + Purpose: Safely retrieves symbols from ring segments
  + Error Handling:
    - Null checks for all node access
    - Validates indices
    - Returns empty string for invalid cases

## 6. Error Handling

* + Input Validation:
    - Safe node access in get\_symbol\_at\_index
    - Null checks for all node operations
    - Index bounds checking
  + State Management:
    - Puzzle becomes unresponsive after success or timeout
    - Timer stops on puzzle completion
    - Disabled interaction after timeout

## 7. Testing & Validation

* + Testing Method: Manual gameplay and debug prints
  + Tested Scenarios:
    - Ring rotation and snapping at 45° steps
    - Label upright maintenance during rotation
    - Diagonal combination detection
    - Timer countdown and timeout
    - Success/failure feedback
  + Edge Cases:
    - Multiple diagonal combinations
    - Rapid rotation and confirmation
    - Timer expiration during interaction

## 8. Next Steps / To-Do

* + ✅ Implement diagonal combination checking
  + ✅ Add keyboard controls
  + ✅ Improve label rotation handling
  + 🔲 Add audio feedback for success/failure
  + 🔲 Animate smooth snapping (tween rotation)
  + 🔲 Add hint system after multiple failed attempts
  + 🔲 Implement save/load for puzzle progress

## 9. Integration Notes

* + Output:
    - puzzle\_solved flag for level progression
    - solved\_equations array for tracking progress
    - Visual feedback through labels and popups
  + Integration:
    - Scene switch to next level on puzzle completion
    - Timer expiration triggers level failure
    - Progress tracking for multiple combinations