

This Is The Only Level Game Report

https://youtu.be/e17XC_knApg



Introduction

"This Is The Only Level" is a unique puzzle platformer game featuring an elephant character that must navigate through what appears to be a single level repeatedly. The distinctive aspect of this game is that while the physical layout remains constant, each stage introduces a different gameplay mechanic or "twist" that requires the player to adapt their strategy. The game consists of one level with five distinct stages, each with its unique challenge. The player controls an elephant character, navigating through obstacles, avoiding spikes, pressing buttons to open doors, and ultimately reaching the exit pipe to progress to the next stage. The game features a clean interface that displays important information such as the death counter, timer, stage number, and helpful clues. The implementation uses object-oriented programming principles with Java and the StdDraw graphics library to create an engaging gameplay experience that requires both dexterity and problem-solving skills.

Implementation Details

Overview of the System Architecture

The game is implemented using object-oriented programming principles in Java, with the StdDraw library handling graphics rendering. The architecture follows a modular approach with four main classes:

1. **Stage**: Manages stage-specific properties such as gravity, velocity, control keys, and informational text.
2. **Player**: Handles player attributes, movement, collision detection, and rendering.
3. **Map**: Represents the game environment including obstacles, interactive elements, and spatial relationships.
4. **Game**: Controls game flow, user input processing, UI rendering, and stage progression.

The main class (`AhmetErenAslan`) initializes the game by setting up stages and starting gameplay.

Class Detailed Descriptions

Stage Class

The `Stage` class encapsulates all properties and behaviors specific to each game stage. Each stage maintains its unique characteristics while sharing the same physical level layout.

Key components of the Stage class:

- **Stage Information**: Both a brief clue and more detailed help text are stored to guide players.
- **Physical Properties**: The gravity and velocity values control how the player moves within the stage. Different values create distinct movement feels across stages.
- **Control Mapping**: Each stage can reconfigure the keyboard controls by assigning different key codes to actions.

The `getKeyCodes()` method returns an array of all control keys for the current stage, making it easy for the Game class to determine which keys should trigger which actions.

Player Class

The `Player` class represents the elephant character controlled by the user, handling its position, movement, and visual representation.

Key aspects of the Player class

- **Position Tracking**: The x and y coordinates determine where the player is rendered on the screen.

- **Visual Representation**: The draw() method selects the appropriate sprite based on the player's direction.
- **Movement Physics**: The velocityY field supports the implementation of jumping and falling mechanics.
- **Directional Awareness**: The directionRight boolean tracks which way the player is facing, ensuring the correct sprite is displayed.

The `respawn()` method resets the player's position when they die or when a stage is restarted, placing them back at the defined spawn point.

Map Class

The `Map` class represents the game environment, handling the layout, obstacles, interactive elements, and collision detection.

Key responsibilities of the Map class:

- **Environment Definition**: Arrays store the coordinates of all level elements (obstacles, spikes, pipes, etc.).
- **Collision Detection**: The `checkCollision()` method determines if the player can move to a desired position.
- **Player Movement Logic**: The `movePlayer()` method applies physics and handles collisions.
- **Interactive Elements**: Methods like `pressButton()` and logic for opening doors implement stage-specific mechanics.
- **Visual Rendering**: The `draw()` method renders all map elements with appropriate colors and positions.

The `changeStage()` method checks if the player has reached the exit pipe, indicating completion of the current stage.

Game Class

The `Game` class is the central controller that manages the overall game state, including:

Key responsibilities of the Game class

- **Game Loop Management**: The `play()` method contains the main game loop, updating the game state and rendering.

- **Input Handling**: The `handleInput()` method processes keyboard and mouse input.
- **UI Rendering**: Draws UI elements like the timer, death counter, and help text.
- **Stage Progression**: Manages transitions between stages and end-game conditions.
- **Game State Management**: Handles game reset, restart, and completion scenarios.

The Game class ties all other components together, creating a cohesive gameplay experience.

AhmetErenAslan Class (Main)

The main class initializes the game by creating the stages, configuring their unique properties, and starting the game loop:

This class configures all five stages with their unique properties:

1. **Stage 1**: Normal controls - standard platformer controls
2. **Stage 2**: Reversed left/right controls - challenges player's muscle memory
3. **Stage 3**: Auto-jumping mechanics - requires precise timing to navigate
4. **Stage 4**: Multiple button presses required - tests persistence
5. **Stage 5**: Restricted movement (cannot move left) - forces forward planning

How Classes Interact

The interaction between classes follows a clear hierarchy that allows for encapsulation while enabling necessary communication:

1. Game and Map Interaction:**

- The Game class creates a Map instance for each stage
- Game processes user input and delegates movement commands to the Map
- Map reports back when a stage is complete

2. Map and Player Interaction:**

- Map contains a Player instance and controls its movement
- Map checks for collisions before updating player position
- Map detects interactions between the player and elements like buttons or spikes

3. Map and Stage Interaction:**

- Map references its Stage instance to determine physics properties
- Stage provides key code mappings for player controls
- Stage supplies information text displayed in the UI

4. **Game and Stage Interaction:**

- Game maintains the current stage index and advances it upon completion
- Game displays stage-specific help and clue texts

The following sequence diagram illustrates a typical game loop iteration:

1. Game loop begins in Game.play()
2. Game processes input via handleInput()
3. Input is translated to movement commands for Map
4. Map.movePlayer() updates Player position considering physics and collisions
5. Map checks if Player has reached the exit pipe
6. Game updates the display with current state
7. Loop repeats until game completion or exit

Implementation of Game Mechanics

Player Movement and Physics

The player movement system implements basic platformer physics:

- **Horizontal Movement**: Controlled by arrow keys with velocity determined by the current stage
- **Jumping**: Applied as an instantaneous upward velocity that gradually decreases due to gravity
- **Gravity**: Constantly pulls the player downward at a rate specified by the stage

The code in `Map.movePlayer()` handles these physics calculations

Collision Detection

Collision detection is crucial for gameplay, preventing the player from passing through obstacles.

Button and Door Interaction

The game implements an interactive button and door mechanism:

1. Player steps on the button

2. Button press count increments
3. When count reaches the required number (1 or 5 depending on stage), the door opens

Stage Transition and Completion

When the player reaches the exit pipe, the game shows a completion message and advances to the next stage:

User Interface

The game provides a clean interface showing essential information:

- **Timer**: Shows elapsed gameplay time in minutes:seconds:milliseconds format
- **Death Counter**: Tracks the number of player deaths
- **Stage and Level Indicators**: Shows current progress
- **Help System**: Provides clues and detailed help when requested
- **Control Buttons**: Allow for restarting the current stage or resetting the entire game

Conclusion

The "This Is The Only Level" game demonstrates the principles of object-oriented design in action, with clear separation of concerns between classes that handle different aspects of the game. The Stage class enables the creation of varied gameplay experiences while reusing the same physical level layout. The Map class manages the complex spatial relationships and interactions, while the Player class focuses on character-specific behaviors.

The implementation successfully achieves the assignment goals of creating a nostalgic game that challenges players with varied mechanics across multiple stages. The modular design allows for easy extension with additional stages, each with unique twists on the core gameplay.

By combining physics-based movement, collision detection, interactive elements, and a clean user interface, the game provides an engaging experience that requires both dexterity and problem-solving skills. The visual elements, including the elephant character sprites and environment graphics, create a cohesive aesthetic that enhances the gameplay experience.

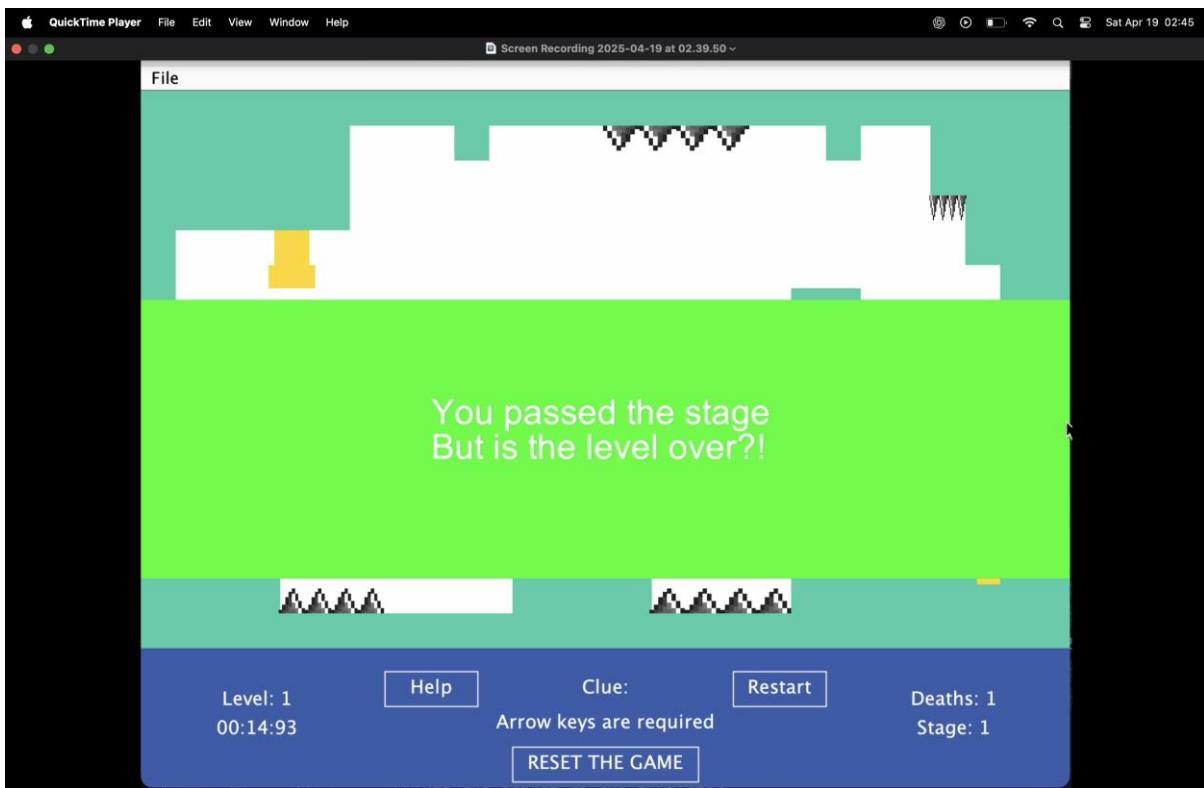
There are several screenshots of the game on the next pages:



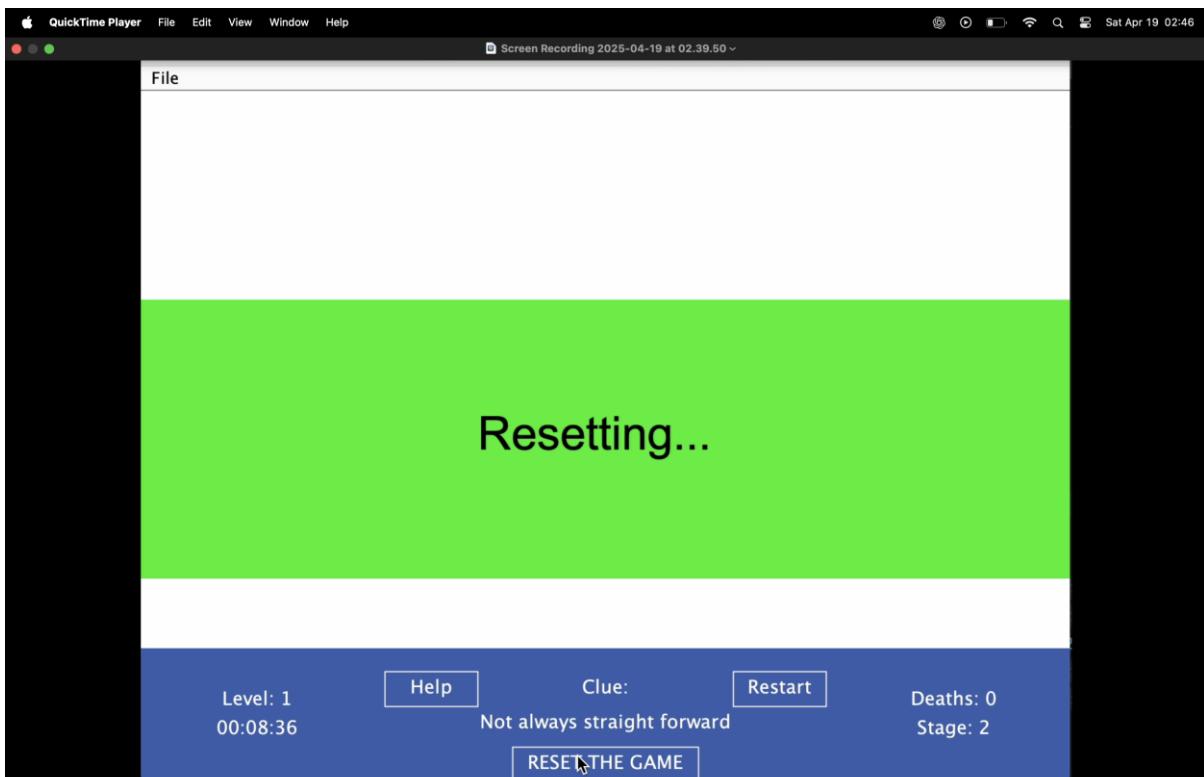
(GAME ENVIRONMENT)



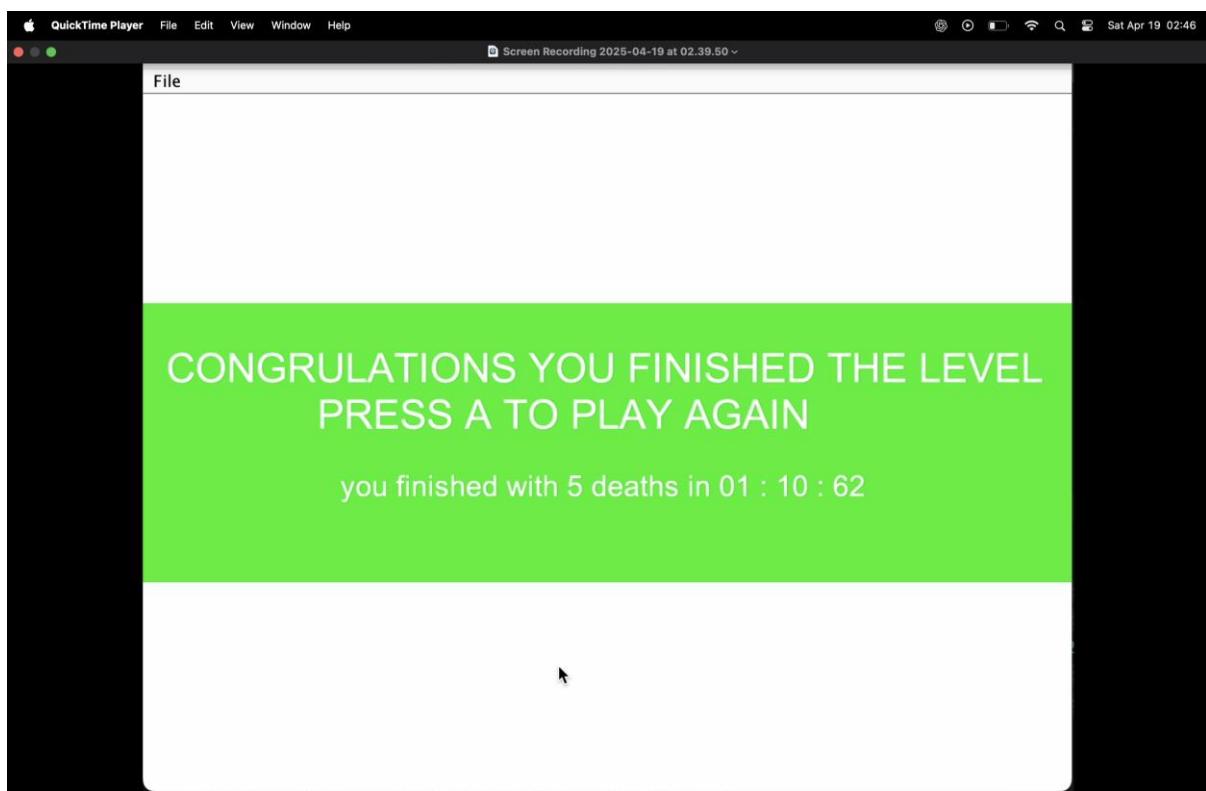
(BUTTON PRESSED ANIMATION)



(PASSING STAGE SCENE)



(RESARTING SCENE)



(FINISH SCENE)