Chill Duckie Project

Beyrem Hadj Fredj

Mia Sara Christina Dreier

Süheyla Caglayan

January 29, 2025

1. Game Description

The Chill Duckie team has developed a 2D MazeRunner-concept game, emphasizing a special storyline, great music and sound effects, and a challenging gaming experience. The game has been developed from the top down only using the LibGDX library in Java.

1.1 Game Storyline

"The main character Orpheus, and his beloved wife Eurydice, both lost their lives from a venomous snake bite. Until one stormy night, he was mysteriously resurrected and rose from his grave. Disoriented, he searched desperately for Eurydice's final resting place, but it was nowhere to be found. A whispered rumor from a wandering spirit spoke of her tomb on the far side of the vast cemetery. Now, Orpheus must traverse the treacherous cemetery, avoiding its supernatural monsters and deadly traps, to reunite with his wife and escape the realm of the dead together."

1.3 Goal

Survive from the enemies and escape the maps, collect the statue (only one statue per level) for the exits to become functional.

1.4 Gameplay

- You only have 3 lives.
- Avoid the traps, stepping on them will result in losing a heart.
- Stay away from the mobs, if you get too close they will start chasing you, if you get attacked you lose a heart.
- There are 4 different types of mobs: Ghosts, Slime, Bat, and Spider; they all behave the same.
- Shadows (Dark ghosts) are lost souls; looking at them will result in decreased movement speed.
- You can find floating hearts around the map to restore 1 HP, only if you have less than 3 hearts.
- Buffs are located around the maps; collecting them will give you 3 seconds of immunity.
- When a heart is lost, the character will turn red for a moment.
- When a heart is restored, the character will turn green for a moment.
- When a buff has been collected, the character will turn golden for a moment.
- When a slow effect is active, the character will turn blue for a moment.
- The character dies if all lives are lost. You will have to restart the level.

1.5 Keybinds

- Use the arrows or `"W"`,`"S"`,`"D"`,`"A"` for moving your character respectively up, down, right, or left.
- Press and hold `"Left Shift"` or `"Right Shift"` to sprint.
- Scroll up or down with your mouse to respectively zoom in or out the camera.
- (You can cheat by pressing `"I"` to heal and `"O"` to take damage SHHHHH!)

1.6 How to Run the Game

After opening the project, follow these steps to configure the Run/Debug configuration:

1. The game uses Java SDK 17. Select `<directory>.desktop.main` as the module.
2. Choose `<package>.DesktopLauncher` as the class containing the main method.
3. **Remark:** For Windows users only, make sure to clear the VM option field by deleting `-XstartOnFirstThread`.
4. Finally, press `Run Game`. Have fun!

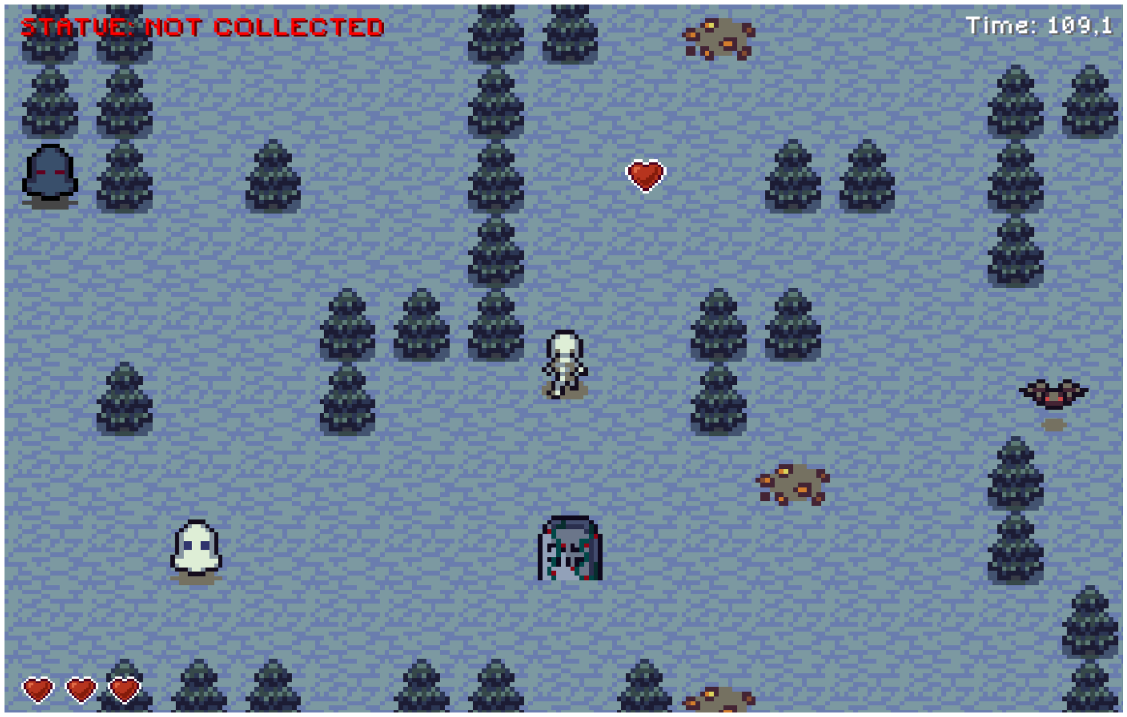

Figure 1: Gameplay screenshot

2. Game Features

- Splash screen animation when the game starts, displaying the team logo.
- Self-made (recorded) audio tracks and sound effects.
- Menu screen through which you can navigate options to start the game, provides access to the levels menu, settings menu, and quit the game.
- Settings screen provides controls for adjusting music and sound effects volume and the option to toggle full-screen mode, to the player's preference.
- A levels menu allows the player to choose between different game levels and navigate back to the main menu.
- When the game is started through the main menu, before starting level 1, it shows a quick cut-scene with the game storyline.
- In the cut-scene screen, the storyline is shown in the form of paragraphs, slowly fading in, plus the ability to skip the animation or show the next paragraph by pressing `"Space"`.
- Collectables can be found around the map like a 3-second invincibility buff, hearts that restore one life.

- Interactive traps, resulting in the loss of 1 life when stepped on.
- Mobs (dynamic obstacles), will chase the player when he gets too close to their range, coming into contact with the enemies will decrease a life.
- Mob types are randomly chosen, they are different every time you play the level.
- Special mobs, Shadows in this case, with unique look and animations, when the character is close enough, their eyes will open and give a slow effect only if he's going in their direction.
- When `"ESC"` is pressed, the player will access the pause menu, it provides options to resume, restart, access the settings menu, access the levels menu, go back to the main menu, or quit the game.
- In the gameplay, a HUD shows the necessary information, like the lives left, statue collection status, and a timer.
- After the statue (key) has been collected, the exits become functional, and an animation around the character points toward the nearest exit.
- By pressing and holding `"Shift"` the character can sprint.
- Sound effects are played whenever the character takes damage, fails the level (losing all lives), collects the statue, and reaches the exit.
- Video effects occur whenever the character takes damage, restores one heart, collects a buff, and is being slowed.

3. Project Structure

Along with the documentation, for more details, check the UML Diagram provided.

3.1 Screens

To take care of the many menu screens in the game, making a parent class called `BaseScreen` which most other screens inherit from, was the most relevant approach. This class implements common functionality shared across all of the screens (except the SplashScreen and the CutSceneScreen), like the background image, stage, render method, resize method, show method, and dispose method.

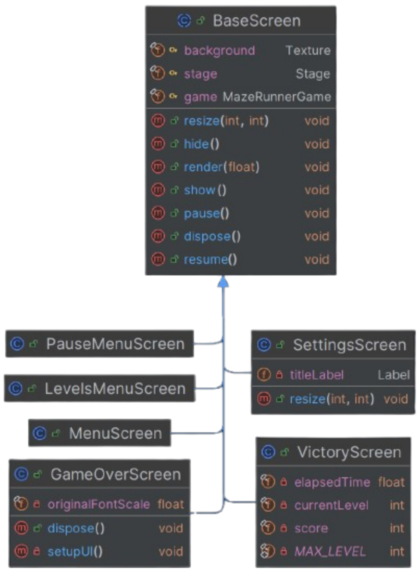

Figure 2: BaseScreen UML Diagram

3.2 Map Loading

The map generation and management system utilizes a `MapLoader` class that implements a layered approach. This architecture provides modular and maintainable map management with clear separation of concerns and efficient handling of game elements.

The system employs:

- A dual-layer structure consisting of:
  - Base layer for floor tiles
  - Object layer for walls, items, and interactive elements
- 2D integer arrays representing tile types
- One method works and adapts for all levels and their respective properties file
- **Special Elements:** Tracks player start position, exits, enemies, and shadows... Allowing for smart reusage in other implementations
- **Collision System:** Methods for detecting walls, traps, and collectibles
- **Animation System:** Implements powerup animations using sine wave scaling
- **Enemy System:** Supports various enemy types (ghost, blob, spider, bat) and shadow enemies

Loading Process: The map loading process follows a sequential approach:

1. Load level-specific properties file
2. Set map dimensions based on level requirements
3. Initialize base and object layers
4. Create floor layer with basic tiles
5. Add objects, walls, and enemies from properties

Unified Tile Management: The implementation of different tile types within a single `MapLoader` class offers significant performance and maintenance benefits. By managing all tiles through unified 2D arrays (`baseLayer` and `objectLayer`) and a texture mapping system (`tileRegions`), the approach reduces memory overhead and simplifies collision detection. This centralized design allows for efficient batch rendering of the entire map and streamlines the process of adding new tile types, as they only require a new texture region and corresponding integer identifier. In contrast, individual classes for each tile type would introduce unnecessary complexity and potential performance bottlenecks due to multiple object instantiations and separate method calls.

3.3 Enemies

The `Enemy` class implements enemy behavior through an animation and movement system:

Core Architecture

- Position tracking using Vector2
- Dual animation states: stationary and chasing
- Configurable constants for hitbox threshold, animation speed, movement speed, and detection range

Key Components

- **Enemy Types:** Supports multiple and random enemy variants (ghost, blob or slime, spider, bat) with unique sprite animations
- **Animation System:** Uses LibGDX's Animation class with separate frame sets for stationary and chasing states
- **Movement Logic:** Implements player detection and pursuit mechanics with obstacle collision
- **Collision Detection:** Checks for player contact and map obstacles during movement

This design provides efficient enemy behavior management while maintaining flexibility for different enemy types and movement patterns.

3.4 Character Movement

The game implements character movement and camera control through a coordinated system spanning the `MazeRunnerGame` and `GameScreen` classes:

Core Components

- Vector-based position tracking using `characterX` and `characterY`
- Configurable movement speed with sprint multiplier option
- Direction-based animation system using LibGDX's Animation class
- Collision detection with map elements

Movement Implementation The movement system follows this process:

1. Input detection (WASD/arrow keys)
2. Position update calculation with delta time
3. Collision validation against map elements
4. Animation state updates based on movement direction
5. Position and camera synchronization

3.5 Camera System:

- OrthographicCamera for game world and separate HUD camera
- Zoom functionality with persistent zoom levels
- Viewport scaling adapting the aspect ratio
- Top-down camera behavior while maintaining the character in the center of the screen

Viewport Management The viewport system handles:

- Automatic resizing while maintaining aspect ratio
- Separate viewport logic for game world and HUD elements
- Zoom controls with minimum/maximum constraints
- Dynamic scaling based on screen dimensions

This architecture provides smooth character control while maintaining proper display scaling across different screen sizes and zoom levels.