HO CHI MINH CITY UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

REPORT

Subject: Object Oriented Programming - 23CLC03

Group Members:

- Trần Tất Trí - 20126030
- Lê Anh Duy - 23127011
- Nguyễn Thanh Gia Bảo - 23127158
- Lê Trương Bảo Ngọc - 23127237

Super Mario Game

Instructor(s):

- Ths. Nguyễn Lê Hoàng Dũng
- Ths. Nguyễn Minh Huy
- Ths. Phạm Bá Thái

ACKNOWLEDGEMENTS

We would like to express our sincere thanks to the Faculty of Information Technology, University of Science, Vietnam National University, Ho Chi Minh City for creating favorable conditions for us to study and complete this report on the subject OOP - Object-Oriented Programming.

In particular, we would like to express our deep gratitude to:

- Mr. Nguyễn Lê Hoàng Dũng
- Mr. Nguyễn Minh Huy
- Mr. Phạm Bá Thái

For imparting knowledge and guiding us during the process of writing the assignment.

We would like to sincerely thank the teachers for their attention and help in the process of writing this report.

Sincerely thank you!

Ho Chi Minh, December 13, 2024

Group Information

- **Lê Anh Duy** - 23127011

- **Trần Tất Trí** - 20126030
- **Nguyễn Thanh Gia Bảo** - 23127158
- **Lê Trương Bảo Ngọc** - 23127237

Evaluate the Level of Completion

Order	Object	Level of Completion	Note
1			
2			
3			
4			
5			

Introduction

Requirements

Design

Implementation

Assets

Fonts

Map

1. Creating the Map

The map for our Super Mario game was created using a combination of design tools and predefined assets. We utilized the **Tiled Map Application** to visually design the layout of the map. Tiled is an open-source tool that provides a user-friendly interface for designing 2D game levels with layers, objects, and tiles.

Each tile or object on the map corresponds to a specific class in our codebase. These classes are defined in `item-object.tsx` and `tiles.tsx`. By assigning classes to blocks, we ensure that each tile inherits properties like texture, collision behavior, and interactions.

2. Using the Tiled Map Application

The Tiled Map Application was crucial in creating a professional and scalable map design. Below are the steps we followed to utilize Tiled effectively:

a. **Setting Up the Project**:

- Imported the tileset graphics that were pre-designed for our game. Each tileset image represents a collection of blocks (cloud, flagpole, soil, etc).
- Configured the tile dimensions 16x16 pixels to match the block size in our game engine.

b. **Designing the Map Layout**:

- Created layers for different components of the map, such as the background, platforms, and interactive objects.
- Placed tiles and objects on the grid to construct the game levels.

c. **Defining Object Classes**:

- Each block or object was assigned a class based on its role in the game. For example:
  - **Ground blocks** were linked to the `Tile` class.
  - **Interactive objects** like coins or power-ups were linked to specific classes defined in `item-object.tsx`.

d. **Exporting the Map**:

- Once the map design was complete, it was exported as a `.tsx` file. This format preserves all the metadata, such as object positions, layer information, and tile properties.

3. Naming Conventions

Consistent naming conventions were adopted to maintain clarity and organization:

- **Map files** were named based on their purpose, such as `map-1-3.tsx`, `map-2-1.tsx`, or `map_interface.tsx`.
- **Objects** in the map were assigned meaningful identifiers that corresponded to their respective classes (e.g., `coin`, `enemy`, `block`). For animated objects, #x,y,z will be added at the end of the name class, with (x,y) being the coordinates of the move to, and z being the position of the base camera.

This systematic naming approach made it easier to load and manipulate map data in the game engine.

4. Loading the Map

The map data from `.tsx` files is loaded into the game using the `loadMap` method in `Map.cpp`. Below is an explanation of how the process works:

a. **Parsing the .tsx File**:

- The `Tileset` utility reads the `.tsx` file and extracts relevant data such as tile positions, object classes, and metadata.

b. **Initializing the Map**:

- The number of rows, columns, and tile dimensions (`tilewidth`, `tileheight`) are retrieved from the `.tsx` file and stored in member variables like `m_row`, `m_col`, and `m_block_size`.
- The `QuadTree` is initialized with dimensions based on the map size for efficient collision detection.

c. **Populating the Map**:

- For each object in the `.tsx` file:

    - **Entities** such as enemies or interactive objects are instantiated using the `EntityFactory` class.
    - **Tiles** like ground blocks or obstacles are created using the `TileFactory` class.
- All tiles and entities are added to the appropriate data structures (`map`, `tilePos`, `EntityManager`).

d. **Setting Player Position**:

- The player's initial position is retrieved from the `"player_pos"` metadata in the `.tsx` file.
- The player's attributes (position, size, velocity, and mode) are reset using the `resetPlayer` method.

Sounds

Textures

Save & Load Game

1. Idea

The **Save and Load Game** functionality is a vital component of our game, ensuring players can preserve their progress and resume gameplay at their convenience. This feature not only enhances the user experience but also provides flexibility and continuity in gameplay. It is designed to handle player-specific data, game configurations, and progression records seamlessly.

2. Save Game Description

The **Save Game** functionality captures the current state of the game and stores it in a structured file format. This includes:

1. **Player Metadata**:
    - The player's name is recorded as the primary identifier. Each player has a dedicated save file to avoid conflicts.
    - Game statistics, such as the player's score, the number of coins collected, and the remaining time, are saved to track achievements and performance.
2. **Game Configuration**:
    - Settings such as audio volume, control mappings, and unlocked levels are preserved to ensure consistency across gaming sessions.
3. **Level Progress**:
    - The current level and the player's position within it are stored, enabling the game to resume precisely where the player left off.
    - High scores or records for completed levels are saved to encourage competition and improvement.
4. **Save File Format**:
    - The game state is stored in a lightweight and human-readable JSON format. This ensures compatibility and ease of debugging if required.

- Each player has a unique save file located in a dedicated directory (e.g., `Users/PlayerName.json`).

The **Save Game** process is triggered automatically at key moments, such as:

- Completing a level.
- Exiting the game.
- Manually saving through the game's menu.

3. Load Game Description

The **Load Game** functionality retrieves the saved data and restores the game state. Key components of this process include:

1. **Reading Player Data**:
    - The system checks for the existence of a save file corresponding to the player's name. If no file exists, a new game session is initialized with default settings.
2. **Restoring Game State**:
    - Player-specific information, such as score, coins, and level progress, is loaded into the game environment.
    - Game configurations, including controls and volume settings, are applied to match the player's previous session.
3. **Level Initialization**:
    - The current level and the player's position within it are restored. This ensures that players can continue from exactly where they left off.
    - If no valid save data is found for the level, the game defaults to the first level or a tutorial stage.
4. **Error Handling**:
    - If the save file is corrupted or missing crucial data, the game gracefully handles the situation by initializing default values and notifying the player.

4. Key Features of Save and Load System

1. **Automatic and Manual Saving**:
    - The system provides flexibility with both automatic saves at critical points and manual saves via the game menu.
2. **Player-Specific Save Files**:
    - By organizing saves under player-specific directories, multiple players can enjoy the game on the same system without overwriting each other's progress.
3. **Progressive Unlocking**:
    - Each completed level is recorded, allowing players to revisit unlocked stages or move forward in the game.

4. **Seamless Integration**:
    - The save and load processes are integrated into the game engine, ensuring minimal interruptions to the gameplay experience.
5. **User-Friendly File Format**:
    - The JSON format allows for easy storage and retrieval of data while maintaining readability for developers.

5. Challenges and Future Improvements

1. **Corrupted Save Files**:
    - To address potential corruption, implementing a backup save system would ensure data integrity.
2. **Cloud Saving**:
    - Future iterations could introduce cloud-based save functionality, enabling players to access their progress across multiple devices.
3. **Encryption**:
    - To protect player data, especially in competitive settings, encryption could be applied to save files.
4. **Save Slots**:
    - Allowing multiple save slots per player would give users more flexibility in managing their progress.

Conclusion

The **Save and Load Game** functionality is essential for a modern gaming experience. By carefully designing this system to capture all relevant game state data, we ensure that players can enjoy a seamless and personalized experience. Looking ahead, enhancements such as cloud saving and additional security measures will further improve this feature's robustness and appeal.

Features

Testing

Challenges

Evaluation

Conclusion

References