COMP 345 – S

Assignment #1 – Report

Team members:

Kyo Young Lim (29293566)

Jarred Tsibidis-Goldberg (40245559)

Victor Iliescu (40258989)

Mohammadreza Abolhassani (40039616)

---

1. Implementation Details This game was built using **C++** with the **SFML** graphics library. We used SFML for rendering, event handling, and real-time interaction. The project was structured using **object-oriented programming (OOP)** principles, making it modular and easier to manage.

2. How to Start the Game

- Compile and run Tower_Defense_1.cpp.
- The game starts in **Initial Setup Mode**, where you enter the grid size.
- Click the **Play** button to move to the **Map Editor Mode**.
- In **Map Editor Mode**, place the **entry** and **exit tiles** by clicking on edge tiles.
- Draw a **valid path** by dragging the mouse.
- Once a valid path is drawn, the game enters **Play Mode**.
- In **Play Mode**, you can select towers from the UI and **drag them onto the map**.
- Press **T or Enter** to start the next wave of monsters.

3. Game Structure and Features

- **Game Class (Game.h, Game.cpp)**
    - Manages game states (setup, editor, play mode, pause).
    - Uses SFML's **RenderWindow** for drawing and handling user input.
    - Implements **event polling** for mouse clicks, key presses, and UI interactions.
    - Updates all objects every frame using the **game loop**.
- **Map System (Tile.h, Tile.cpp)**
    - Uses a **2D vector** to store grid-based tiles.
    - Each tile is an **SFML sprite**.
    - Allows the player to create paths dynamically with mouse input.
    - Ensures the path is **valid** before switching to play mode.
- **Tower System (Tower.h, Tower.cpp)**
    - Implements three tower types: **Basic, Sniper, and Rapid**.
    - Towers use **range-based targeting** to attack monsters.
    - Each tower has attributes like **damage, attack speed,** and **cost**.
    - Uses **SFML textures and sprites** for visual representation.
    - Towers automatically attack enemies using a **cooldown system**.
- **Monster System (Monster.h, Monster.cpp)**
    - Monsters move along the **predefined path**.
    - Each monster has attributes like **speed, health, strength, and reward**.
    - Different types: **Normal, Rogue, Tank, Swarm, Elite**.
    - Uses **SFML vector operations** for movement calculations.
    - If a monster reaches the exit, it reduces the player's wealth.
- **Monster Generator (Monster_Generator.h, Monster_Generator.cpp)**

    - Spawns monsters **based on game level**.
    - Randomizes **monster types per wave**.
    - Uses **C++ STL vectors** to store and shuffle monster waves.
    - Adjusts difficulty dynamically by increasing **health, speed, and strength**.
- **Entity Base Class (Entity.h)**
    - Parent class for all objects (monsters, towers, tiles).
    - Provides common functionality like **positioning, scaling, and movement**.
    - Implements **SFML's Drawable interface** to simplify rendering.

4. Additional Features

- **Drag and Drop Mechanics:** Towers can be selected and **dragged onto valid tiles**.
- **Path Validation:** Ensures a connected path before starting the game.
- **Wave System:** Press **T or Enter** to trigger a new wave.
- **Game Over Handling:** The game stops when the player runs out of money.
- **UI Elements:**
    - Displays current **wealth, level, and warnings**.
    - Uses **SFML Text and Fonts** for clear on-screen instructions.
    - Buttons for **starting the game and placing towers**.

5. Compliance with Stated Problem and Game Rules Our implementation closely follows the project requirements. We implemented a **grid-based map system** that allows for **custom path creation**, ensuring that only valid paths are accepted. The **tower system** supports different attack types and upgrades, while the **monster generator** dynamically increases difficulty, in line with the assignment's requirement for progressively harder waves. The game also enforces **player resource management**, as required by the problem statement. All essential mechanics, such as **tower placement, pathing, and wave-based progression**, work as specified.

6. Simplicity and Appropriateness of the Solution The code is structured using **object-oriented principles**, making it modular and easy to extend. Each component (game, towers, monsters, map) is handled by separate classes, ensuring clarity. SFML's **event handling and rendering** features were used effectively, making interactions smooth and intuitive. The **drag-and-drop interface for tower placement** simplifies user interactions. The balance between functionality and usability ensures that our solution remains efficient without unnecessary complexity. The result is a fully functional and **easily expandable** Tower Defense game.