

1. Overview:

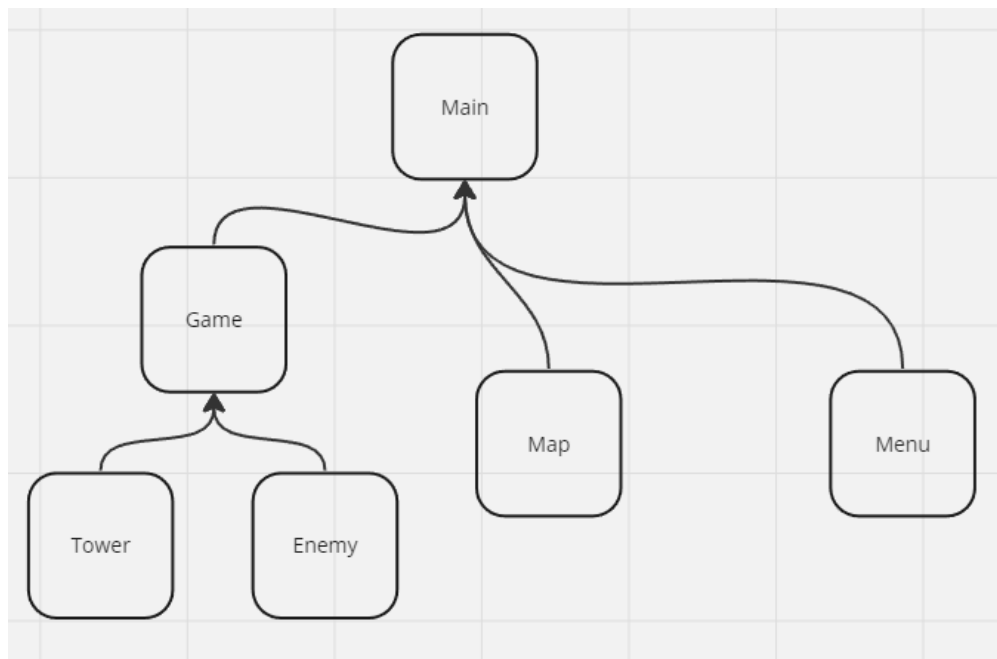
Our project is a tower defense game written in C++ using the SFML library. The goal of the game is to survive 10 rounds of enemies with increasing difficulty each round to win the game. The enemies will attempt to reach the end of a predetermined path. The player can stop them from doing so by building towers in exchange for credits, which are earned through defeating enemies.

Towers are also upgradable to deal with enemies in the latter part of the game. There are multiple types of enemies, towers, and functionalities such as hit points, attack damage, attack radius, ...

2. Software structure:

2.1. Overall architecture:

Enemy and Tower interact with each other via Game, Map is used for rendering the Enemy path and places where Towers can be placed. Game handles the interactions between Enemy and Tower, as well as rendering Enemy and Tower.



2.2. External libraries:

This project uses SFML for graphics and UI design. SFML allows us to easily display graphics and implement interactive elements using C++ without needing to learn extra functions.

2.3. Game:

Game acts as the most important class of the project, utilizing all the classes to simulate, update and draw the required entities. The main function works primarily just by initializing the game variable, which automatically starts itself and rounds round 1 for enemies and towers to be implemented into. It has several variables and functions, but some important ones include:

Variables:

- **gameWindow** : This is the window where everything is rendered and shown in a visual sense.
- **clock** : This keeps track of the real-time amount the game has been activated for.
- **enemies** : This is a vector that holds the current enemies that are on screen.
- **towers** : This is a vector that holds the current towers that are on screen.
- **Run_clock** : This is the clock that the game works on. For the purposes of our game, it restarts every second and thus causes the game to update and the enemies and towers to attack each other every second.

Functions:

- **Game(GridMap)** : This is the basic constructor function for the class. It also automatically creates the window required to render the game and automatically starts the tower and enemy placements into round 1.

- `run()` : This acts as our main loop where every second, we ask the program to `update()`, `render()`, spawn enemies, and so on.
- `update()` : This function handles updating the game's wave, loss requirement, enemy movement, and so on.
- `render()` : This function handles the drawing of enemy and tower sprites. The map rendering is done using Map class.
- `handleTowerEnemyInteractions()` : This function checks if each tower is in the enemy's range, and then attacks as required. The same is done for enemies to towers.

2.4. Tower and Enemy:

Tower has the following attributes: name, damage, HP, range, cost, damage over time, and position. Tower can be placed on a tile on the Map, where it will attack any nearby Enemy.

The Tower class includes 5 subclasses, they are as follows:

- Plant: A basic tower that has moderate attack and range with low price, has decent HP (this is the baseline for all towers and all the following statements are in comparison to the Plant subclass)
- Water: A long range tower with low attack and diminished HP, slightly higher price
- Fire: A tower with lowered base attack but 5 seconds of damage over time, much higher cost
- Ground: A tower with no attack parameters but high HP, purely a roadblock for enemies
- Magic: A tower that applies moderate attack to all enemies within its range. Slightly increased range and double the HP at the expense of 5 times the price

The Tower should be automatically removed by the Game class when `getHP` is called every game tick to make sure no 0HP towers remain on the map.

Enemy has the following attributes: HP, speed, attack, coins, and range. Enemy communicates with Tower through the Game object. Enemy moves on every frame and attacks once every second. When an Enemy's HP reaches 0, it executes the function die() and marks itself as dead.

- PlantEnemy is a basic Enemy. It moves and attacks any nearby Tower.
- TreeEnemy spawns 3 PlantEnemy when it dies.
- BombEnemy, when attacking a Tower, also deals damage to nearby Towers.
- BossEnemy has a large amount of HP and deals massive damage.

2.5. Map:

Map is a 16 by 16 grid plane. There are three different types of tiles inside the grid: non-interactable tiles, interactable tiles, and lane tiles. Towers can be placed on interactable tiles. Lane tiles are where the enemy will spawn and move along. The type of each tile is predetermined in the map layout text file. Map class can read the aforementioned text file, generate the grid map, and handle mouse input on the map.

2.6. User Interface:

When the game is built, players will be greeted with a main menu which has two options to either start or quit the game. Main menu class shows the options on the screen and handles the user's mouse input

Side Menu is included in the map folder due to them being closely related to each other. Side Menu keeps track of the position of the map and then is set to be on the right side of the map, and is the main way for the player to play the game. On the side menu is a build button for the player to enter build mode, after clicking the button, the player can click on interact-able cells to place towers. The player can also see their health points, gold, and score on the side menu

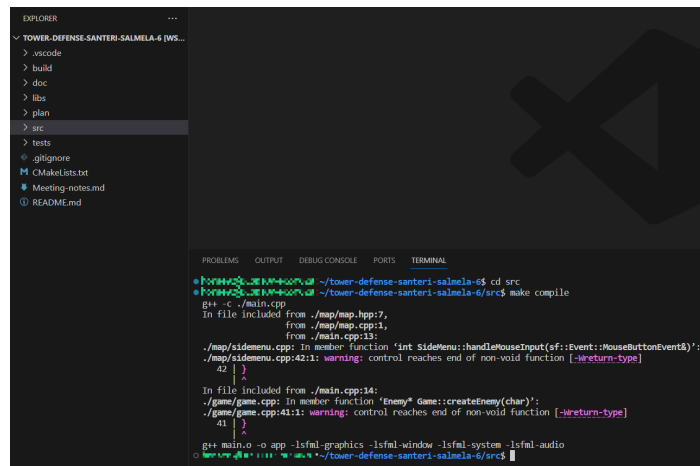
3. How to build and use software:

In order to build and use software, the user has to install SFML on their WSL system. A tutorial on how to install can be found here:

<https://www.sfml-dev.org/tutorials/2.6/start-linux.php>

The Makefile has already been configured to link the necessary SFML libraries for the program to function.

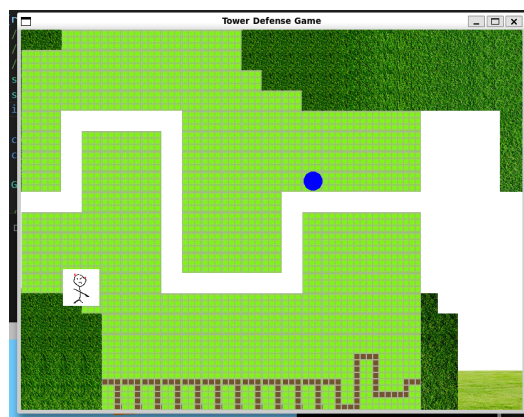
To run the program, open the project folder in VS Code, change directory to “src” and input “make compile”.



```
EXPLORER
TOWER-DEFENSE-SANTERI-SALMELA-6 (WSL)
  .vscode
  build
  doc
  libs
  plan
  src
  tests
  glibignore
  OKMakeLists.txt
  Meeting-notes.md
  README.md

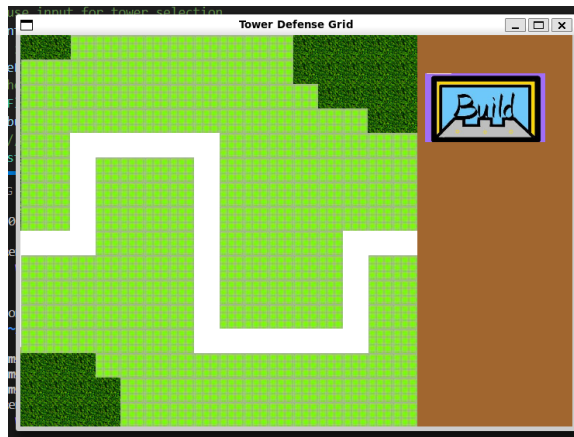
PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL
~/tower-defense-santeri-salmela-6$ cd src
~/tower-defense-santeri-salmela-6/src$ make compile
g++ -c ./main.cpp
In file included from ./map/map.hpp:7,
                 from ./main.cpp:13:
./map/sidemenu.hpp: In member function 'int SideMenu::handleMouseInput(sf::Event::MouseButtonEvent&)':
./map/sidemenu.hpp:42:1: warning: control reaches end of non-void function [-Wreturn-type]
42 | }
   | ^
In file included from ./main.cpp:14:
./game/game.hpp: In member function 'Enemy* Game::createEnemy(char)':
./game/game.hpp:41:1: warning: control reaches end of non-void function [-Wreturn-type]
41 | }
   | ^
g++ main.o -o app -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
~/tower-defense-santeri-salmela-6/src$
```

If there are no errors shown, that means the program has compiled successfully, and can be run using “make run”.



4. Testing:

- original-main.cpp: main file used for testing the capabilities and libraries of SFML.
- tower-test.cpp: tests getters for tower stats in Towers class to make sure they report correct stats to game when calling on a tower subclass.
- enemytest.cpp: testing enemy movement from checkpoint from checkpoint, as well as enemy destruction



(map_test.cpp)

- map_test.cpp:
 - Testing loadMap(): loads, then reads map.txt and gets information of the map tiles as well as information about the checkpoints on the map for enemy class. The function returns a 2d vector of the map data and a vector of checkpoints coordinates.
 - Testing draw() (map class): draw the 16x16 grid map based on the map data vector and draw the selected background texture.
 - Testing handleMouseInput() (map class): Mouse input on the map is checked to see whether or not it was on an interactable tile, if yes then the texture of that particular grid is then changed.

- Testing draw() (side menu): draw a rectangular side menu and a button within the side menu.
- Testing handleMouseInput() (side menu): Register mouse inputs on the button.
- Testing setPosition(): set the side menu next to the map.



(menu_test.cpp)

- menu_test.cpp: testing the draw function which draws a window with two 2 buttons, testing handleInput() which registers mouse inputs on the two buttons.
- final-test.cpp: combination test of all the classes implemented

5. Work log:

	Hoang	Dung	Aditya	Xiong
Week 1	Implemented basic Enemy movement and basic subclasses definition (6 hours)	Implemented basic map class (6 hours)	Implemented basic game class. (6 hours)	Implemented basic Tower class with HP, Cost, and Damage definitions (6h)
Week 2	Implemented Enemy HP and die() function (4 hours)	Implemented map loading function and mouse input handling function (6 hours)	Added more practical functionality to the game class. (4.5 hours)	Added Plant and Fire Tower basic headers, added other relevant parameters for towers. (3h)
Week 3	Implemented attack() and die() function for TreeEnemy (6 hours)	Implemented function to handle map and tower texture (3 hours)	Assisted in fixing enemy, tower and map classes alongside game class. (12 hours)	Implemented TowerPos to set and track tower locations. Implemented basic attack function for towers. Started work on preliminary asset designs. (6h)
Week 4	Implemented BombEnemy attack() (2 hours)	Map class bug fixing and added map test. (10 hours)	Misc. bug fixing and game class tweaking. Reworked checkpoint system and map file reading. (9 hours)	Completed UI assets and Tower assets. Implemented all Tower subclasses. (8h)
Week 5	Bug fixes and finalization (14 hours)	Implemented side menu class and menu class. (6 hours)	Reimplementation of game class and miscellaneous bug fixing. (5 hours)	Cleaning up assets. Removed redundant code from Tower and rewrote the attack to apply damage over time. (8h)
We all worked on the documentation during the last week.				