Tower Defense

Group: tower_defence_santeri_salmela_5
Mikhail Romanov, Aki Oura, Leo Teodosin and Kalle Lindgren

December 2023

1. Overview

We have built developed our own tower defense game. Game has fully own graphics and a lot of features. Our game has 8 different kinds of enemies and 6 different kinds of towers. Enemies and towers have all their own animations when attacking, moving or dying. Game has money system where player gets cash by surviving rounds, player can use that cash to buy new towers or to upgrade existing ones. Also, there is mechanic where damage between towers and enemies depends on objects type and on the type of object that it's attacking

Game starts with main menu, where user can choose level to play. User can choose between few hard coded maps or randomly generated map. If user chooses randomly generated, game will randomly generate fully new level that player have never played before.

After main menu game will start and players mission is to survive as long as he/she can. User wins game by reaching 30 rounds or loses by letting to many enemies get to the finish.

List of features:

- Main menu
- Animation of objects
- Own graphics
- Pre-existing levels
- Randomly generated levels
- 6 types of towers
- 8 types of enemies
- Mouse control
- Different damage levels depending on type of object
- Upgradable towers
- Towers can be damaged by enemies

Photo of main menu

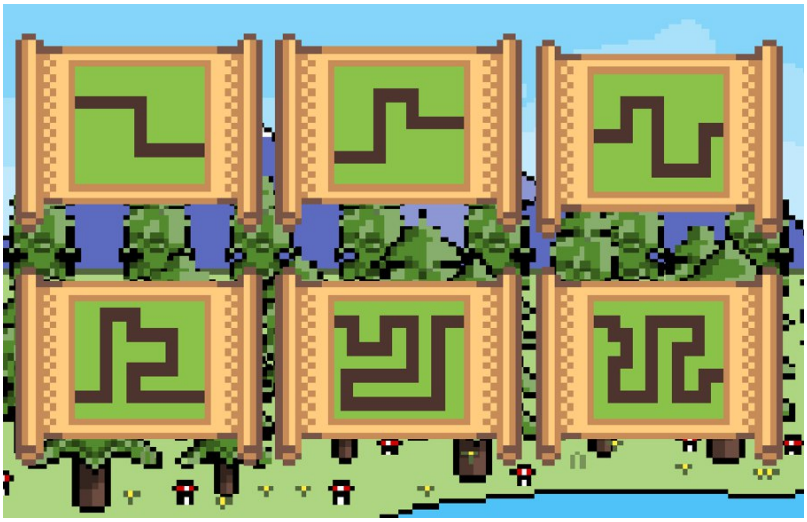
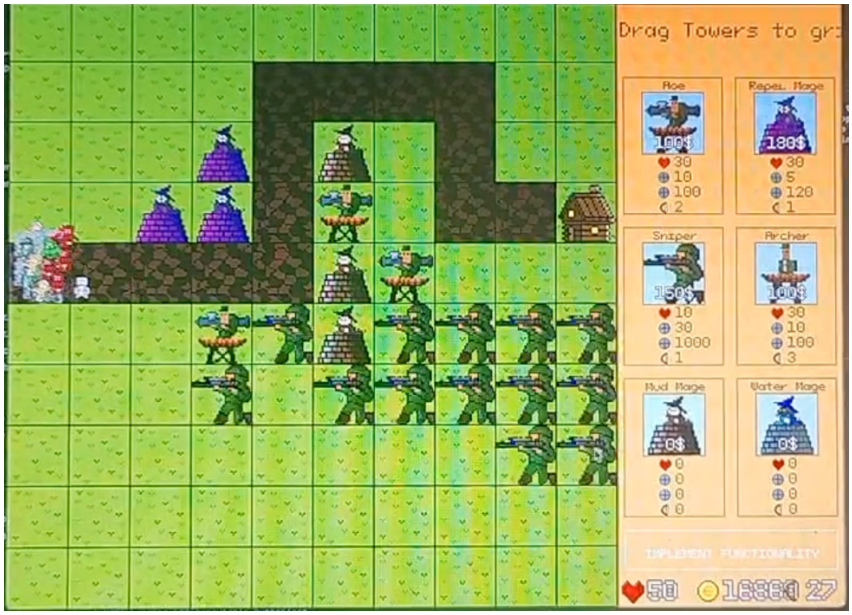


Photo of in game action



2. Software structure

In our project, the Game class is the central component orchestrating the entire game. It collaborates with the Level class for in-game logic, the Render class for rendering textures, MainMenu for initial player interaction, ChooseLevelMenu for selecting levels, and SideMenu for buying towers and displaying stats.

The game starts with MainMenu, prompting players to choose between a pre-existing or random level. Choosing random generates a new level using the Level class, while pre-existing levels are selected through ChooseLevelMenu from six options.

Once the game begins, the left side displays a grid managed and updated by the Render class, serving as the battlefield for enemies and tower placement. On the right, SideMenu generates a menu for buying towers and displays essential stats. Both Render and SideMenu classes efficiently load and manage textures using the ResourceHandler class, contributing to a smoother gameplay experience.

The Level class handles in-game logic and stores the current grid, enemies, towers, etc. It relies on two classes: Square and Object. The Square class manages individual squares in the level's grid, storing occupancy and other details. The Object class serves as a basic constructor for all game objects. It has two subclasses: Tower and Enemy, which inherit from Object and act as more specific constructors for towers and enemies, respectively.

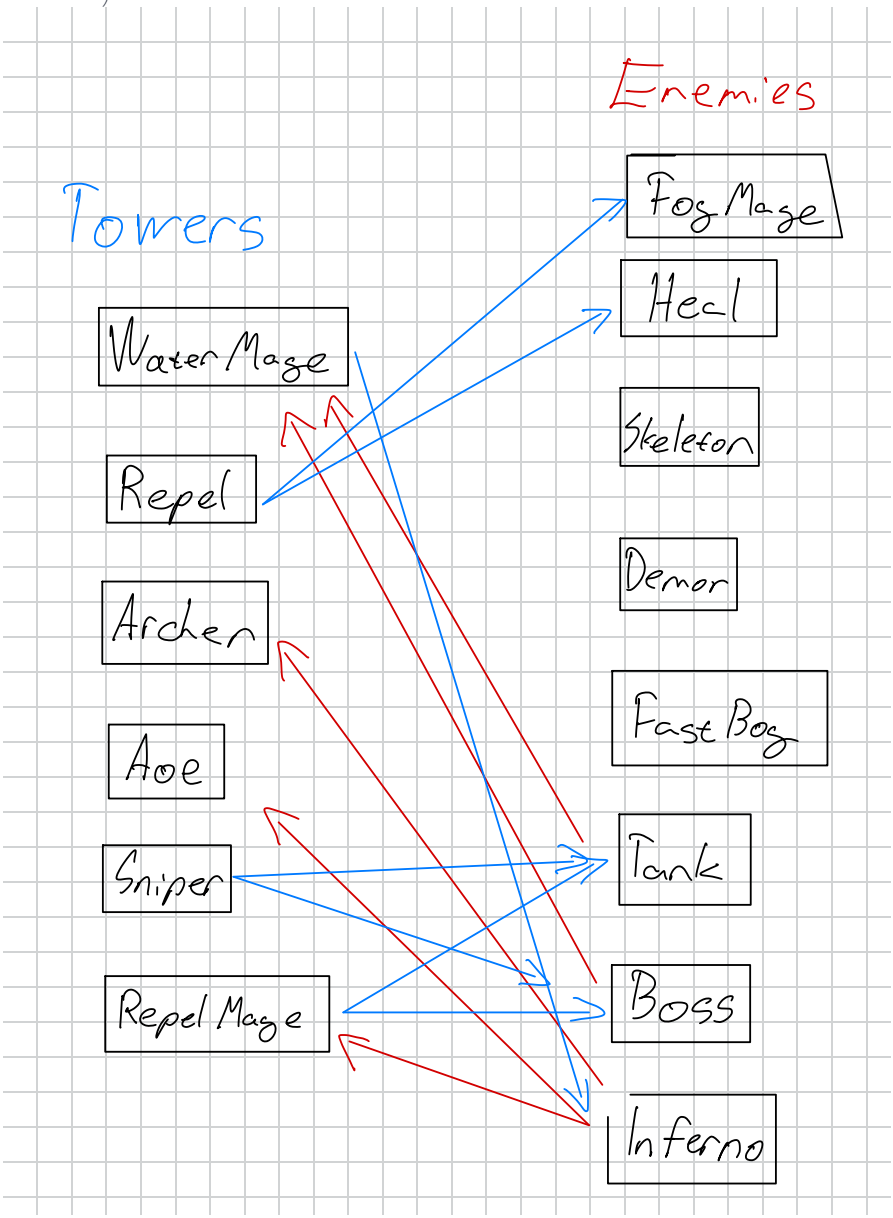
Under the Tower and Enemy classes, there are a total of 14 subclasses, each representing a specific type of tower or enemy in the game. These subclasses inherit from either the Tower or Enemy class, depending on their type. This inheritance hierarchy allows for a modular and extensible design, making it easy to add new tower and enemy types to the game.

Additionally, there are two standalone classes: Vector2D, which manages in-game coordinates, and AttackTypes, which manages types of objects and includes functions to determine damage between objects. These classes play essential roles, with Vector2D ensuring consistent handling of coordinates throughout the game, and AttackTypes providing a central mechanism for managing and calculating damage interactions between different game objects.

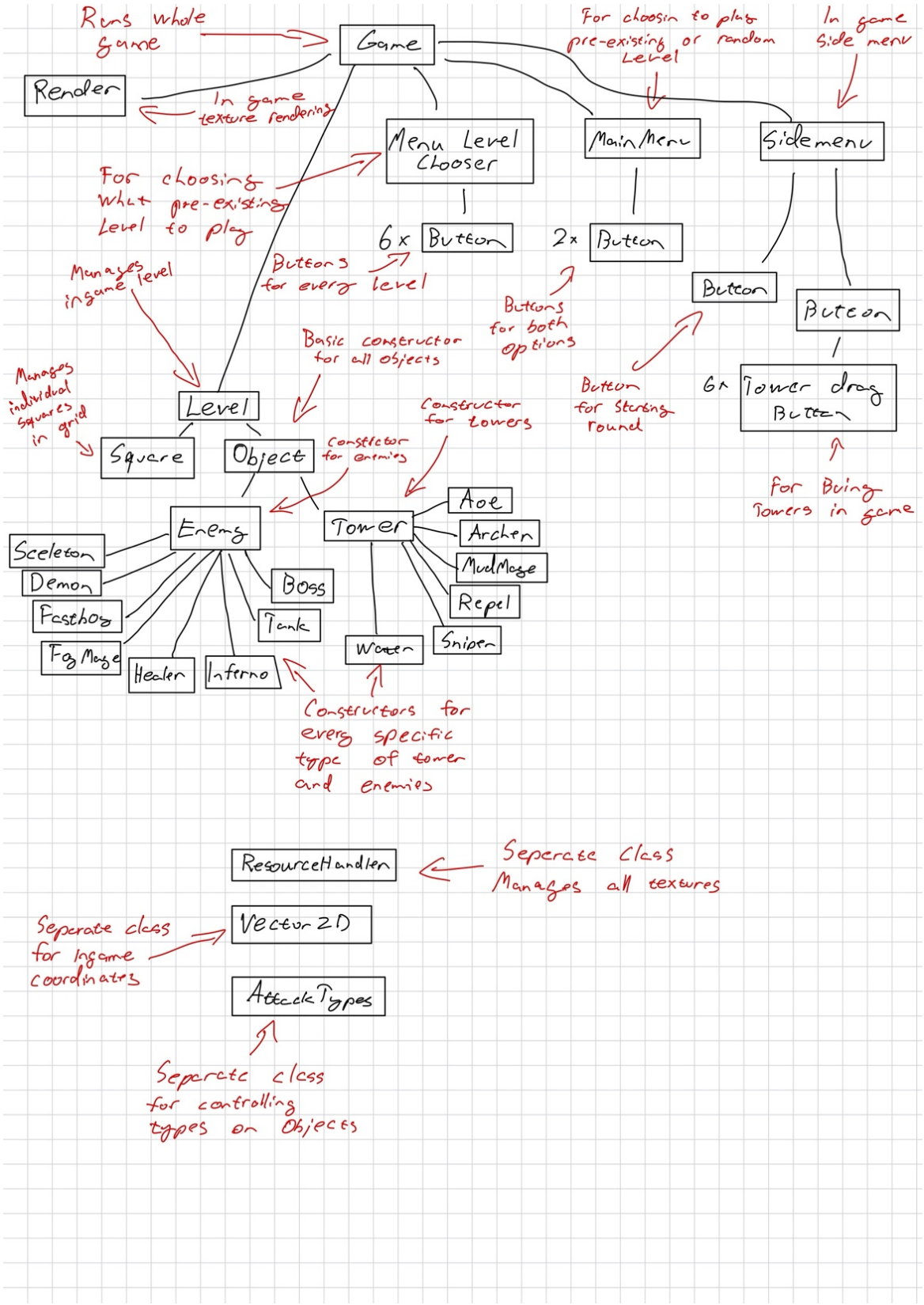
Our game also has structure where damage given by one object to another one is dependent on the type of the object giving damage and type of the object receiving damage. For example, Tower type “Water Mage” gives more damage to “Inferno” type enemy, but also it receives more damage, when attacked by Enemy type “Boss Knight” or “Tank”.

Detailed picture of type based damage structure

(Blue arrow means tower is giving more damage to enemy and red arrow means another way amount)



Detailed picture of structure



3. Instructions for building and using the software

Building instructions

Firstly, user must download zip file of the game and unzip it. If user is on macOS or on Linux, after unzipping user must just open folder with unzipped files in the terminal. If user is on Windows, user must move unzipped files to local Linux environment and open folder with game files in local Linux terminal.

After opening files user must run next command in the terminal

- sudo apt update
- sudo apt install \ libxrandr-dev \ libxcursor-dev \ libudev-dev \ libopenal-dev \ libflac-dev \ libvorbis-dev \ libgl1-mesa-dev \ libegl1-mesa-dev \ libdrm-dev \ libgbm-dev \ libfreetype6-dev \ libsfml-dev \ cmake
- mkdir build
- cd build
- cmake ..
- cmake -DCMAKE_BUILD_TYPE=Debug ..
- cmake --build .

On this point all libraries that project uses should be downloaded and project should be already built. User can start the game by going to the build folder of the project and executing "./MainExec" in the terminal. After that game will start and we will require user to have fun.

If there were any troubles or error while executing commands mentioned above, we will kindly ask user to contact development team.

Usage instructions

Game starts with main menu, where user should choose which type of level they want to play. User can choose to play already existing level, after what in front of user's eyes will appear new window where user then can choose which specific level of 6 existing user wants to play. Or user can choose to play randomly generated level, in which case game will generate fully new and fully random level and game will start.

After game has started, user will see level on the screen with side menu on the right side of it. On the bottom of the side menu, user will see stats of the game, as a current round, money situation and lives remaining. With money that user has, user can buy towers from side menu by clicking on then and dragging to the play field. All towers have different stats as health, damage, etc. That can be seen to the right side from tower. In the side menu is also "Start round" button that will start new round when pressed. Button will start new round that will generate some specific number of enemies (depending on current round). Enemies will spawn on left side on in game level on the first piece of road and enemies' mission is to get to the house that is located somewhere on the right side of the in-game level. If they get to the house user will lose lives (one enemy takes on life).

Depending on enemy's type enemies can only move or they have some ability as well. Enemy's ability can be ability to attack towers, heal other enemies or make fog around them that will decrease towers attack speed. Also, Boss enemy will spawn new smaller enemies when it dies.

Towers can't move inside of game, but they can attack enemies. Depending on tower type, they can attack enemies by dealing damage to them, slowing down enemies or by pushing them back. Also, all towers can be upgraded by clicking first on the tower and then clicking on "upgrade" button. Upgrading will add damage, health, range and attack speed to tower. One tower can be upgraded only twice.

All objects that can attack other objects by dealing damage to them deals different amount of damage depending on what type they are and what type of object that they are attacking. User wins game by reaching 30 rounds. And loses game when users live amount reaches 0.

After winning or losing user will see specific screen, after what user will be pushed back to the main menu.

4. Testing

In our project, we maintain a dedicated folder called "tests" containing all test cases for our game. The tests are primarily focused on classes such as Level, Square, Object, and Drawable derived classes (Menus and Buttons). Other classes indirectly undergo testing as part of evaluating the functionality of these core classes. Therefore, classes beyond the specified four do not have individual test suites.

All tests are executable from the main.cpp file, which is configured to run all tests and display the total error count that occurred during testing. To facilitate testing, our CMake file generates two executables named "TestRunner" and "ButtonTest" that can be launched from build folder by calling "./TestRunner" or "./ButtonTest" command in the terminal. While TestRunner runs the unit tests for Level, Square and Object classes, the purpose of "ButtonTest" was to test the functionality as well as aid in visual design of the Drawable derived classes.

Our testing strategy covers comprehensive evaluations of all functions within the targeted Level, Square and Object classes. Each test case is designed to assess various scenarios, including those in which the functions should operate successfully and situations where the functions are expected to fail. After executing a test, the outcome is verified, and the test result—whether a pass or fail—is printed. In the event of a failure, the associated error is logged, contributing to the overall error count. On the other hand, due to the need of user input, window object and the need to evaluate the visual appearance, each Drawable derived class was tested one-by-one and correct outcomes were left to the developer to judge.

This systematic approach to testing ensures a thorough examination of the core functionality provided by the Level, Square, Object, and Drawable derived classes. By evaluating both successful and failure scenarios, our tests contribute to the robustness of the codebase, providing confidence in the reliability and correctness of our game implementation.

5. Work log

Overall work log

Mikhail: Developed fully level class, helped with object class development and connected object, level and game class together. Made animation for objects. And developed resource handler with Kalle.

Leo: Developed object class and all subclasses with different abilities to each object.

Kalle: Developed GUI of the game, methods that allow user to interact with the game (for example method to drag towers to games grid) and made main menu of the game with help of Aki.

Aki: Made all textures for the game, and they animations. Also made UI of the game with Kalle and helped in design of the GUI.

Week by week work log

Week 43: All members of the team participated in planning the project, and researched information about external libraries.

Week 44: Mikhail started working on the Level and Square classes, Leo started Object class, Kalle started working on SFML library connection to the project and Aki started drawing textures for the game.

Week 45: Mikhail and Leo continued working on the classes that they started on the previous week, implementing tests at the side. Aki continued with textures and Kalle started developing GUI.

Week 46: Mikhail moved to work on connecting his level class to Leo's object class, and Leo moved to developing enemy and tower classes. Kalle finished first version of GUI and Aki finished almost all textures.

Week 47: Mikhail finished with connecting level and object class together and moved on connecting level and object classes to Kalle's game class. Leo continues working on subclasses of enemy and tower classes. Kalle started working on towers placement mechanism. And Aki started work on the design of the main menu.

Week 48: Mikhail made animation mechanism for all textures, made logic that calculates how many and what type enemies to spawn each round and started writing documentation for the project. Leo finished work on all subclasses of the object class. Kalle finished towers drag mechanism and moved to main menu development. Aki continued with main menu and maid textures for "You lost" and "You won" screens.

Week 49: All team worked on finishing touches of the project and troubleshooting problems. Team also commented all their code for Doxygen standard and reviewed documentation written by Mikhail.