Tower Defense Documentation

Autumn 2022

ELEC - A7151

Viivi Alitalo

Antonio Skondras

Samuli Öhman

Edris Hakimi

Advisor: Duong Tran

Overview:

-what the software does, what it doesn't do? (this can be taken/updated from the project plan)

Our project was to create a game of tower defense. Tower defense is a game where the player aims to defend their territory from enemy attackers. In the game enemies usually move in waves from some position on the map to another through a predetermined path visible to the player. Whilst the enemies try to get to the end of their path the player tries to prevent it from happening. They do this by placing towers along the enemy's path to attack and hopefully destroy the enemies. Basically, the players objective is to survive by preventing the enemies from getting to the end. If the enemies reach the end of their path the player loses.

Our original plan was to complete the basic requirements of the project and then go for the further additional features if we had the time and resources. As a team we were able to fulfill the requirements for the project. We also built the classes so that further development could be done.

Our game has a starting screen and a screen where you can decide which map you want to play. There are two maps you can choose from. On the map there is a single path the enemies follow. On the map there are nodes that are loaded from a file (extra feature). The enemies go from node to node and hence know where to turn and so on. There are three types of enemies: easy enemies which are killed immediately from contact; hard enemies which require multiple hits to kill; and splitting enemies that after "dying" split into 3 easy enemies. Enemies all look different in order for you to know what kind of enemy you are faced with. The enemies come in waves and the player can see which wave they are on from the right game bar. This game is like survival mode. Each wave can be thought of as a level, so you can compete with friends on who can get to the furthest wave (there are currently 100 in total).

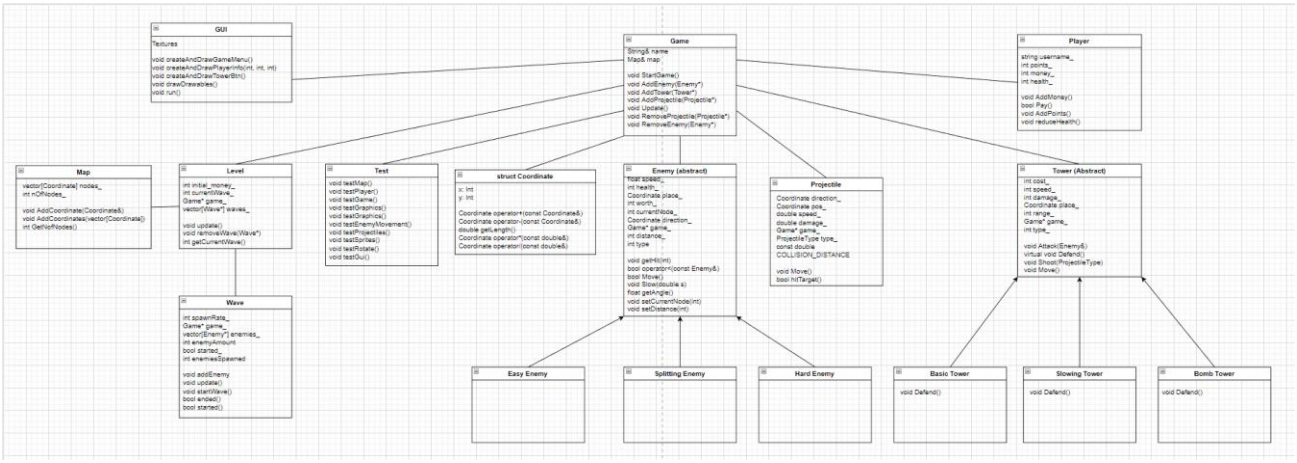
The Player can also see on their screen the amount of money they have, how many lives they have left and the towers and their prices. The towers which are too expensive for the player to buy currently are colored red. There are also five types of towers. A basic tower which shoots enemies in its range, a slowing tower which slows down enemies and a bomb tower which deals huge damage. The towers shoot out projectiles toward the enemies which do not always hit. This is because we wanted to make the game more realistic so that the projectiles do not change direction mid air but are directed toward the position of the enemy in the moment the projectile is shot. Also, the towers shoot at the enemy in their range that is the furthest on the path. Each tower has their own look to them and also shoot out projectiles of their own kind.

As the game goes on the player receives money for each enemy that it kills. The harder the enemy is to kill the more money the player receives for killing it. The player then with this money pay for towers to prevent the enemies from getting to the end of their path. The player has fifteen lives in the beginning and every enemy that reaches the end of the path decreases these lives. When the player no longer has enough lives the game ends and the player will see a game over page where they can decide if they want to restart or go back to the main menu.

Software structure:

-overall architecture, class relationships diagrams, interfaces to external libraries

We have done an updated UML diagram to show the class relationships and the methods inside them. The diagram is just an overview and has most important methods. Methods like get are not shown since they do not give additional value to the diagram. You can zoom into the diagram in order to get a more detailed look.



The external library we used in the end was SFML. SFML is a simple and fast multimedia library which provides an interface to the components of your PC to ease the development of games and multimedia applications. Our original plan was to use qt but, in the end, decided against it. After Antonio had familiarized himself with qt, he realized that it does not work well in sync with our coding platform vs code.

Instructions for building and using the software:

-How to compile the program ('make' should be sufficient), as taken from the git repository. If external libraries are needed, describe the requirements here

-How to use the software: a basic user guide

Compile:

1. Clone the project onto your computer.
2. Add folders to the project in directory .vscode:
    a. C_cpp_properties.json
    b. Launch.json
    c. Settings.json
    d. Tasks.json
    e. These can be copied from any C++ module we have done in this course.
3. Use your VS-code terminal and use command: sudo apt-get install libsfml-dev
4. Press Cntrl+Shift+P and "CMake: Configure" the project.
5. Press Cntrl+Shift+P and "CMake: Debug" the project.
    a. If this does not work, first try using the program by clicking "(gdb) Launch" in the Run and Debug window of VS Code

b. “Failed to open X11 display; make sure the DISPLAY environment variable is set correctly. Aborted”
c. If you get this error message try:
    i. Run PowerShell as administrator, use command: wsl --update
    ii. Sudo apt-get update
    iii. Sudo apt-get upgrade
    iv. You will need to restart WSL for the update to take effect, use command: wsl --shutdown
    v. Type command: wsl –list –verbose
    vi. You should see the number 2 for you installed distro under the heading VERSION in the response.
    vii. If everything is correct until this point use command: sudo apt update
    viii. Now install gedit: sudo apt install gedit -y
d. “SFML Failed to load image, reason: Unable to open file.”
    i. If you get this error message the problem might be that your computer cant find the images our program uses.
    ii. You could try to change where the program calls the images and add “../” to the beginning of them. This way the program can find them easier.

How to use our software:

After using our compile instructions, a user should be able to run our project. When running it the player will be introduced to our main menu. There the user can decide the map they would like to play. When they decide the map, they will be directed to a new page. On that page will be the map of their choice and a right-side bar with all useful information. There they will find their lives, the wave they are on, the money that they have and the towers that they can buy. Almost instantly the waves of enemies will begin, and the player can start playing. The player can place towers by clicking the tower and then clicking somewhere on the map where they want to place it. When the player loses all their lives they will be redirected to a game over page where they can either decide to play again or go back to the main menu. That is a basic user guide of our software.

Testing:

-how the different modules in software were tested, description of the methods and outcomes.

We had a few different methods of testing our software. Mostly we would make test methods for different classes in the test class. Also, visual means of testing were used.

In the beginning we would mostly create test methods in the test class that would print out values that we could then make sure are the correct ones. We would use all the methods from the class we were testing and see if it was working correctly. Sometimes we would find mistakes and we would go back to working on the class and when we thought we were done

test it again with the tests or modify them to fit the purpose. Some of these tests are still present in the test class, but others were deleted when there was no use for them anymore.

In addition, Samuli would create tests where he would draw the different features he was testing. He would draw on a black background different colored circles which represented the different features. For example, there is a test in our project called testEnemyMovement which has blue circles for nodes (where the enemy has to go); green circles for towers; a red circle for the enemy; pink for the final node and turquoise circles for the projectiles shot by the towers. The animation shows how the enemy moves from node to node, how the towers shoot projectiles at the enemy and how the enemy dies after being hit enough times. This way he could test the enemy move function, tower defend function and multiple other functions which work together.

Antonio made the GUI and hence had a lot of tests which had to do with the visual side of things. He would mostly draw parts of the GUI to see what it looks like in order to know if something needed to be fixed. Since the GUI is mostly about looking aesthetically pleasing, the testing was a lot about if it looked like what we desired it to be.

Work log:

-a detailed description of the division of work and everyone's responsibilities. For each week, a description of what was done and roughly how many hours were used for the project by each project member.

Next, we have compiled per week what we have done and about how many hours it took us. It is a very rough estimate of the hours used and the tasks done. We worked together and hence the line of what everyone did is quite light.

Week:	**Due dates:**	**To do:**
Week 45	**11.10.** 23.59 Project plan submission to git	• Antonio started to familiarize himself with qt and created the Git for the project. (10hours <) • Samuli created the coordinate and map classes. He also created the test class and added his own tests into it. Samuli was also a huge help in setting up the environment and helping others do the same. (14hours) • Viivi created the player and main classes. She also made a new function into the test class to test her player class and was the secretary at the group meeting. Created UML and most of the plan. (10hours) • Edris started to plan the graphic design. He designed the enemies and what the map should look like. (4 hours) • All team members cloned the repository and created the project plan.

Week 46	Project plan review with your advisor	• Antonio made a test which tests drawing of textures and sprites, how enemies move on map paths and generating paths. In addition, preliminary planning for the inside functions for the GUI were made. Change from qt to smfl. (10hours <) • Samuli added the SMFL libraries to the main application. Modified the Enemy class so that enemies can move on the map. He also made the Projectile class and the recognition of collisions of enemies and projectiles. Related to this he updated the test class for projectiles and simple GUI. In general, Samuli also helped other group members. (10hours) • Viivi did not have a lot of time this week due to work rush, but she did complete everything that was set for her to do this week. She completed the abstract class Enemy and its enemies HardEnemy, EasyEnemy and SplittingEnemy. Like everyone else she created her own tests for these to make sure they worked as intended. She also held the meeting again and wrote up the notes. (8hours) • Edris has made three graphical maps in PNG format. He also has implemented enemies and towers graphically. He has been making the level class and tested it. (8-10hours)
Week 47		• This was the week that we were supposed to get most of the aspects of the game done, but they took longer than expected. Also, everyone was very busy with other schoolwork and studying for exams since it is the last week of school before exam week. • Antonio worked on GUI functions, as in tested how the players inputs work. He has also worked on the panel that the player will eventually play with. (11hours) • Samuli did a lot of work last week so that he could do a little less this week. Samuli has worked with projectiles and thought about how hitting enemies could work. He has also been helping everyone with their parts. (3 hours) • Viivi did the abstract class Tower and its subclasses SlowingTower, BasicTower and BombTower. She also started to work on the towers hitting the enemies with projectiles. (8hours) • Edris has made 2 more graphical maps in PNG format. He has also implemented new enemies. (6 hours)
Week 48		• Ando worked on the GUI. (13hours) • Samuli helped others on their parts and worked on finalizing the game. (8hours) • Viivi finished hitting the enemies with projectiles with Samuli. She also started making the documentation. (12hours)

		• Edris was able to design the game menu picture and implement it. (2hours)
Week 49	**Week 49 and 50** Project demo to advisor **9.10.** 23.59 Project final commit to git	• Ando did everything he could do with the GUI. Buttons for towers and putting them on the map. Map choosing and game over page. (24hours) • Samuli finalized the game. Fixed problems. Added functionalities for towers, made waves and so on. (24hours) • Viivi helped with the game and did the documentation. Created splitting enemy. Helped fix problems. (24hours) • Edris was not able to do anything. (0 hours)

Important: *The project documentation must be committed to the Git repository in the doc/ directory.*