

# Documentation Cats VS Rats TD

## Group 1

Henrik 




Antti 

Kasper 




Otso 

## 1. Overview

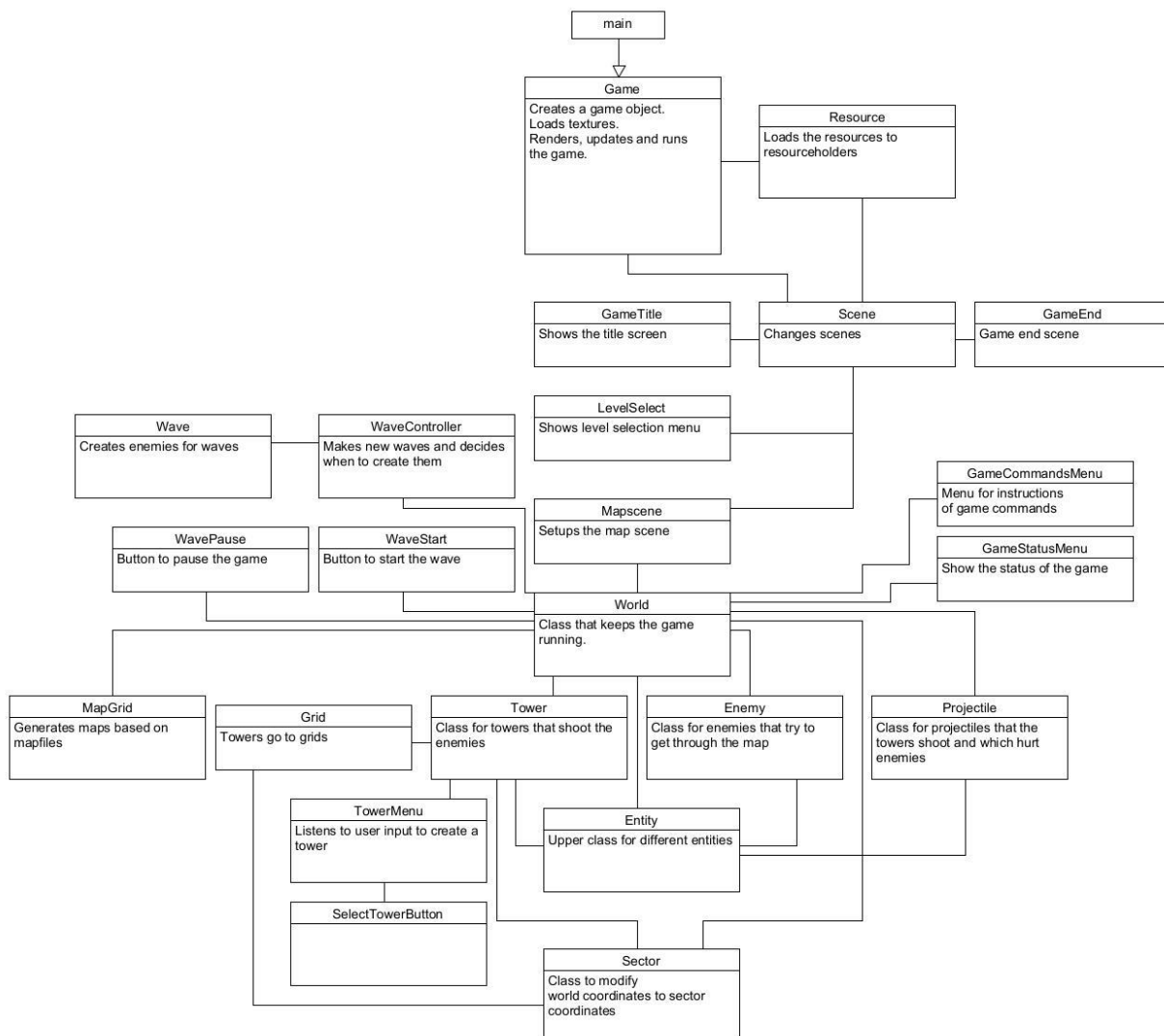
Our project is a Tower Defense game named Cats VS Rats TD (Tower Defense). Our theme is cats who try to defend against rats from passing through the path and prevent them from damaging the house at the end of the path. If a rat reaches the end of the path lives are lost and if all lives are lost, the game is lost. The game has three different rats and three different cats to defend against the rats. The towers cost money which you can earn by killing rats. You can also sell towers that you've placed, but you only get 80% of the tower's purchase price back. The towers can be upgraded once. Upgrading the towers enhances some feature of the tower.

Rat	Hit points	Value	Speed	Picture
Basic rat	4	20	Normal	
Fat rat	10	50	Slow	
Fast rat	2	10	Fast	

Upgraded values are in brackets

Cat tower	Damage	Number of enemies affected	Cost	Upgrade cost	Shooting speed	Specialty	Picture
Gun cat	2 (4)	1	300	100	Normal (Fast)	None	
Freeze cat	1 (2)	1	400	100	Normal (Fast)	Slows down enemies	
Bomb cat	1	3 (5)	500	100	Slow (Normal)	Does splash damage	

## 2. Software structure



Picture 1. UML diagram of the class relationships.

The class relationships of our game can be seen in the UML diagram (1). The game uses an external library called SFML which is a multimedia library that we use for graphics and sound effects. Game class handles starting up the game, loading textures to texture holders, rendering the game and processing events.

The game class also calls scene update, which calls update in other classes. The game has different scenes, one for example the level selection menu and one for the game map. The game world consists of entities like towers, enemies and projectiles that the towers shoot. World class handles the game world, it draws the map, towers, enemies and projectiles.

Our game uses advanced features like smart pointers with entities. Shared pointers are useful for enemy objects because multiple other entities interact with them. In game this means that multiple towers and projectiles affect a single enemy simultaneously.

A template class handles loading different media files dynamically. It is done for all assets before GUI is loaded so any errors should be immediately visible. For this resource class and some others we used the

book SFML Game Development as a reference. The book written by Artur Moreira, Henrik Vogelius and Jan Heller describes how a simple game is built from scratch and it fitted our purposes perfectly.

More detailed description of what each of the classes' functions do is told in their Doxygen documentation

### 3. Instructions

1. Clone the repository
2. Navigate to the root directory of the repository
3. Run `"cmake ."`
4. Run `"make"`
5. Run `"./MyGame"`

If some packages are missing install those. However, the SFML library should be downloaded automatically from its GitHub repository when CMake is configured.

Do not resize the game window.

### 4. How to use the software

#### Instructions:

1. You are first greeted with a welcome screen; press enter to continue.
2. Select the map you want with arrow keys and press enter.
3. Place towers
  - a. Select the tile you want by clicking it with left mouse button
  - b. Press number 1 to place a gun cat, number 2 to place a freeze cat and number 3 to place a bomb cat.
  - c. You can sell a tower by clicking it with the right mouse button. You get back 80% of the purchase price.
4. When you are ready, start the wave by clicking "start wave" button.
5. You can pause the game if you want by pressing "pause game" button.
6. You can see the wave number, enemies left, hit points you have left and how much money you have on the right-hand side of the screen.
7. If too many enemies get through you lose the game.

### 5. Testing

We didn't use any unit tests. Instead, we tested the game by ourselves. This isn't ideal but we didn't have enough time to create tests. We also tested the game with dynamic analyzer tools valgrind and Google address sanitizer and in error situations we ran the program in debug mode.

For example, when Projectile class for towers projectiles used raw pointers, valgrind and address sanitizers showed conveniently the stack trace. It could be seen that enemies were deleted from stack before projectiles reached them. Valgrind showed memory leaks in an external audio library for example. Address sanitizer on the other hand had code line numbers missing. The best results were achieved when these tools were used together to spot the source of errors in our code.

## 6. Detailed description of division of work and everyone's responsibilities

At first, we all tried to understand how to use SFML and tried to create base code before we could divide the work. After we got up to speed with coding, we divided the tasks on the fly with the group members. This worked well as we're friends with each other. In some things we divided to pairs of two.

The division of work was as follows:

Henrik's division of the work:

- Drew the textures and graphics
- Made part of the reading of a map from a file with Antti
- Made entities with Antti
- Movement of the enemies with Antti
- Created documentation

Antti's division of the work:

- Made entities with Henrik
- Movement of the enemies with Henrik
- Damage mechanics and projectiles
- Made part of the reading of a map from a file with Henrik

Kasper's division of the work:

- Made most of the reading of a map from a file and displaying of the map.
- Placing of the towers
- UI with Otso
- Buying and selling of the towers.
- Pausing of the game

Otso's division of the work:

- Created most of the UI elements like the level select menu with Kasper.
- Made freeze cat's mechanics.
- Created resource holder
- Using of the sounds and textures.
- Using of a font

## 7. What was done week by week

- Week 43 (24.10-30.10)
  - Filled the project questionnaire
- Week 45 (7.11-13.11)
  - Had our first meeting
  - Made GitLab repository for our game
  - Created and submitted the project plan
  - Everyone used about 4 hours on this
- Week 46 (14.11-20.11)
  - Had plan review with our advisor
  - Started to play with SFML and started to code the project
  - Everyone used about 5 hours on this
- Week 47 (21.11-27.11)
  - Henrik used 3 hours
  - Antti used 3 hours
  - Kasper used 3 hours
  - Otso used 3 hours
  - Started to work on resources class and on creating the game world.
- Week 48 (28.11-4.12)
  - Henrik used around 1 hour
  - Antti used around 1 hour
  - Kasper used around 2 hours
  - Otso used around 2 hours
  - Tried to use the resources class and draw the map.
- Week 49 (5.12-11.12)
  - Coded most of the project
  - Created documentation
  - Created textures
  - Fixed crashing and memory issues caused by Projectile class
  - Henrik used around 41 hours
  - Antti used around 47 hours
  - Kasper used around 35 hours
  - Otso used around 35 hours
- Week 50 (12.12-18.12)
  - Project demo to advisor (not happened yet)