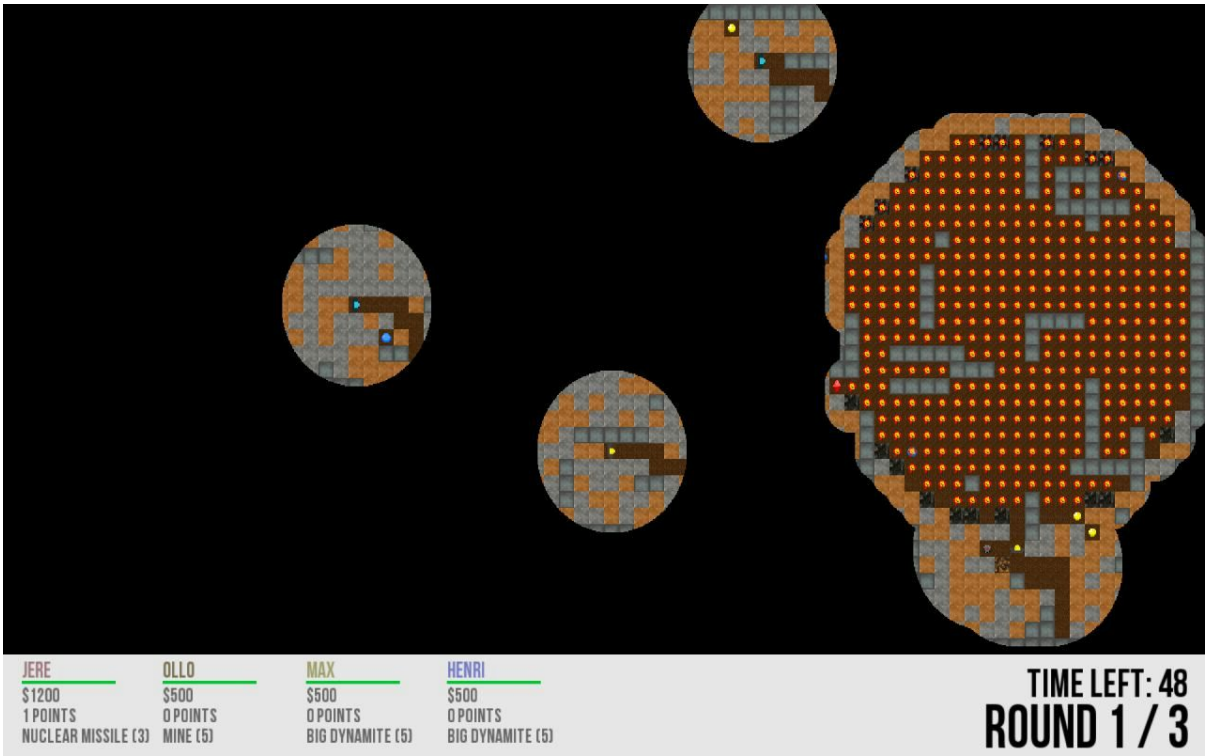
C++ Project Documentation, Minebombers

Henri Nurmi (345545), Olli Rauramo (431433), Jere Vaara (294450), Max Huttunen (348283)



Overview

We have implemented all the minimum requirements except for the single player mode, which we didn't really consider plausible. Minebombers is best played with other live players; developing an AI that would be sophisticated enough to be a worthy opponent would be out of the scope of this project. However, to compensate for the lack of a single player campaign mode, we have made the game playable by up to four players at a time, and implemented several extra features:

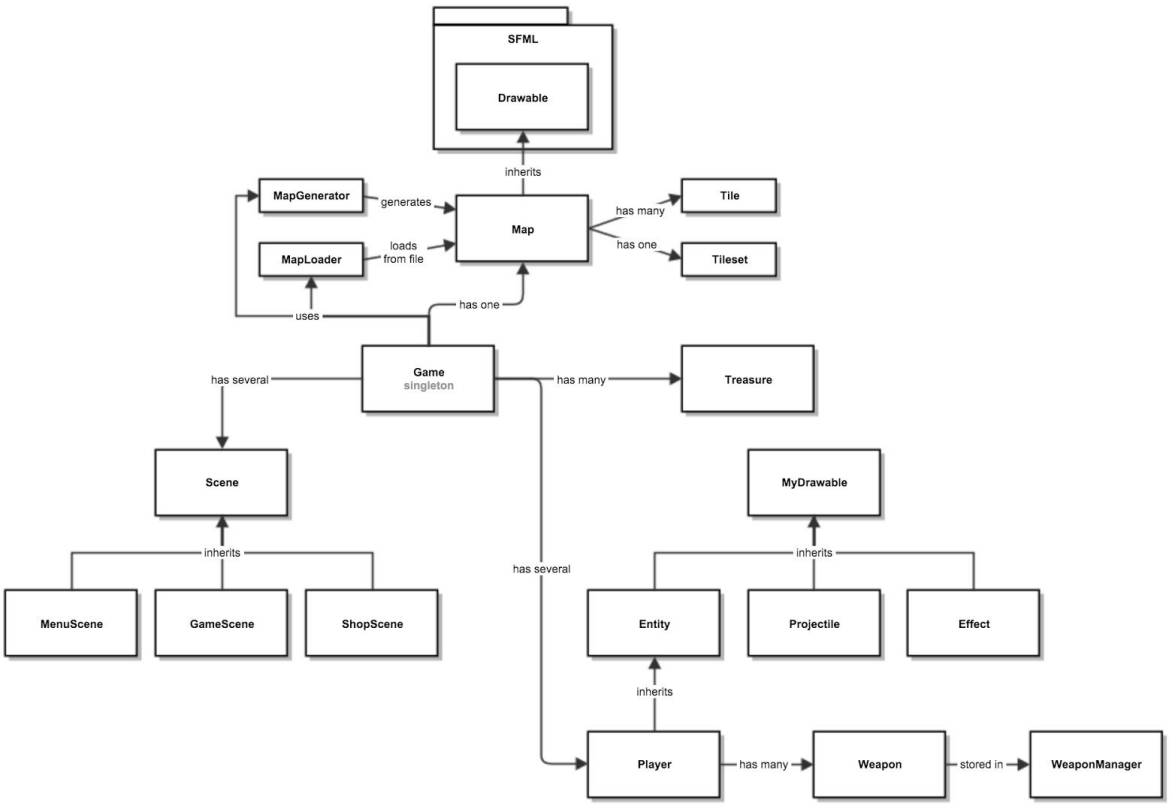
- map editor
- audio
- more weapons
- configuration file
- game is playable by up to 4 players on a single keyboard

When starting the program, the user can choose the amount of players from 2 to 4 and the preferred map. The multiplayer mode consists of three rounds by default, which come to an end once the timer runs out or only one player remains. After a round has ended, new weapons and ammunition can be bought from the store with in-game money, which is earned by collecting treasures. The amount of money received from a treasure varies. A player has 100 health points which are decreased by varying amounts depending on the weapon that is used. In addition to money, a player is also rewarded points for killing other players (10 points) and collecting treasures (1 point). If a player kills themselves, they receive 5 negative points and lose all their money. At the end of the three rounds, the player with most points wins the entire match. Many parameters related to the game can be configured in a separate configuration file.

Structure

We relied heavily on object oriented programming in our implementation. We have used class inheritance and some singletons in order to achieve a clearly structured and easily maintainable code base. The only external library we have used is SFML. It takes care of everything: window management, graphics and sound. Below is a diagram

illustrating the main classes and their relationships with each other:



Description of Software Logic

There are three different scenes in the game: the menu scene, the game scene and the shop scene. A scene is something that defines what is drawn onto the main canvas, it also receives and handles all user input. Although the logic of the GameScene is mostly abstracted into the Game singleton, the MenuScene and ShopScene manage their own internal state. This is because their own state doesn't have to be maintained globally across the program. In addition to keeping track of the state of GameScene, the Game singleton class keeps track of the global program state. It has an instance of each scene and is used to change the currently visible scene which is also stored in the class. Game stores the map, all players, treasures, projectiles and effects, and provides methods to interact with these. MyDrawable is a class that represents everything that can be drawn onto a specific tile on the map. This is why players, projectiles and effects inherit this main class. Sprites, textures and sounds are managed by the ResourceManager singleton class. Maps are loaded either from files using the MapLoader class or

generated randomly using the MapGenerator class. MapGenerator generates maps with an alternating algorithm in a smart way: it tries to build coherent walls and the different ratios of walls, hard tiles and soft tiles vary randomly.

Build Instructions

The program can be built with make. The SFML library is needed however, the install instructions for which can be found here: http://www.sfml-dev.org/tutorials/2.0/start-linux.php
The build command (in src/minebombers folder): make CONF=Release

Usage Instructions

The game is controlled with the keyboard only. In the menu, the preferred game mode can be selected with the up and down arrow keys, and the map can be selected with the left and right arrow keys. If “random map” is selected, a new map is generated at the start of each round. The match is started by pressing enter and the menu can always be returned to by pressing escape, which aborts the game. The controls for the players in multiplayer mode are illustrated in the table below. The keys for moving are also used to navigate the shop, the shoot command acting as enter or select. After each round there is a five second delay before moving to the shop, in which the scoreboard is shown. The shop returns to the next round after a fixed timer or after all players agree to moving on and pressing enter. Sound can be toggled with F10.

	Move	**Shoot**	**Change weapon**	**Use pick**
Player 1	arrow keys	right alt	right shift	Dash
Player 2	WASD	left alt	left shift	<
Player 3	TGFH	B	V	C
Player 4	IKJL	M	N	;

Configuration file

The game contains a configuration file. All configurations are located in mb.config. The structure is <key>=<value> and the value can be a string or an integer.

Testing

Due to the simple nature of the game, we did not write any unit tests or other automated tests. The program is simple to test visually by using it for a while since it doesn't have that many different use scenarios or combinations. On the last day of development all team members tested the game thoroughly to find any bugs or other issues and we believe this is enough to ensure that the program will be bug-free.

Work Log

We have been using git actively throughout the project, which is why the more detailed work log can be viewed from the commit log. As we've mostly been working together either face-to-face or over Skype, pair programming has been an effective approach. However, although we have tried to keep the commits even when programming in pairs or as a group, the relative effort cannot be immediately measured from the amount of commits or changes.

It is hard to exactly separate the work between the team members as everyone has been working on everything, but certain main areas of work can be indicated :

- Henri: map, map editor, map loader, game logic, config, tiles
- Olli: weapons, sprites
- Jere: shop scene, status bar, entities, sprites, animations, sounds
- Max: map revelation, map generator, menu scene, shop scene, music

Due to other responsibilities all team members had, the main amount of work was done on week 50, but the main structure along with some initial work was done already earlier between weeks 47 and 49.

MB MapEditor

We recommend that you don't use the map editor but if you really want to, here's how: if you want to create a new map you have to delete the map.mb file. The editor always saves to a file with this name and also loads the map from this file. You can build the editor (in src/MB MapEditor folder): make CONF=Release

Key Bindings

- **Space**: insert tile
- **Arrows**: move
- **L**: next tile
- **H**: previous tile
- **W**: set tile type to wall
- **F**: set tile type to floor
- **R**: set tile type to rock/sand
- **S**: save the map to map.mb

Awesome MB Map File Format (.mb)

1. Magic number: 0x424d
2. Map name length: 2 bytes
3. Map name: n bytes
4. Tileset name length: 2 bytes
5. Tileset name: n bytes

6. Map width: 2 bytes
7. Map height: 2 bytes

8. Tilesize width: 2 bytes
9. Tilesize height: 2 bytes

10. map width * map height tiles
    tile id: 2 bytes
    tile level: 2 bytes
    tile type: 2 bytes