Brawl Stars

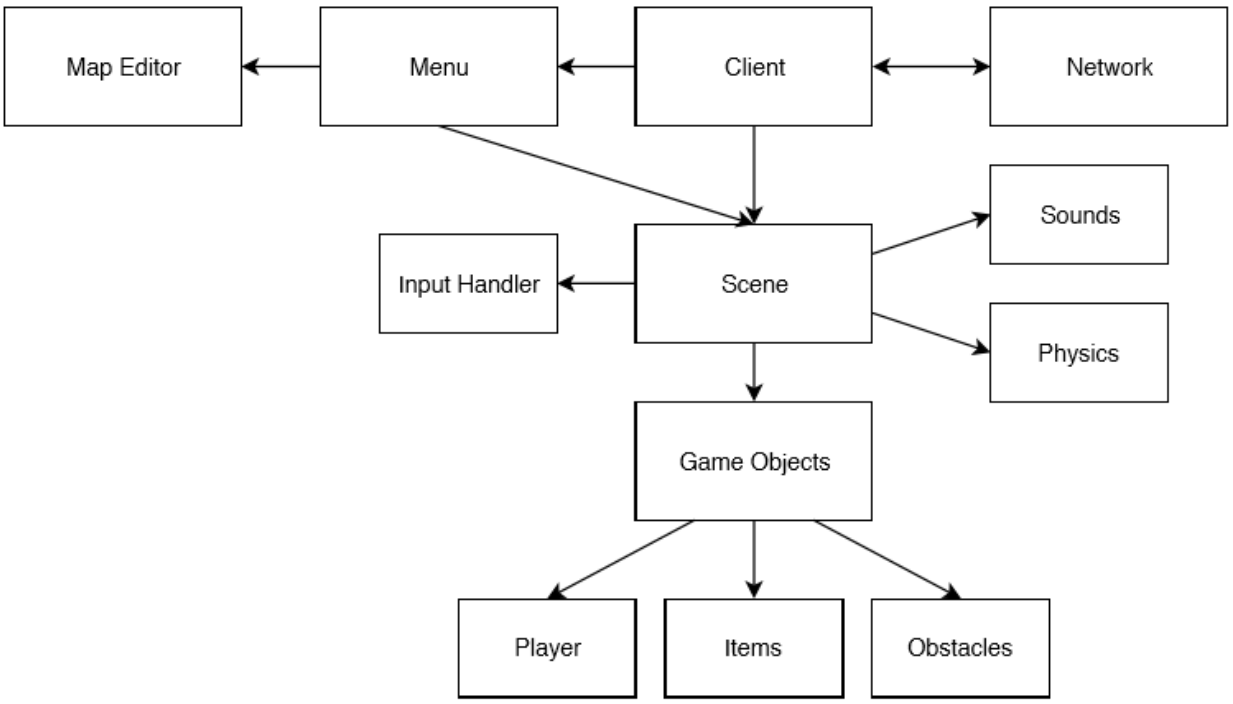
Software Project for ELEC-A7151 Object oriented programming with C++

Simo Hakanummi, Toni Lyttinen, Toni Ojala & Mikko Suhonen

1. General description

- A Brawl Stars-like game, where you have different game modes and characters
    - Free for All: The players will fight against each other and there will be a required amount of score (kills) and the first one to achieve it will win the game.
    - Capture the Flag: 2 teams will fight against each other while trying to steal the enemy team's flag from their base and bring it back to their own base. The first team to achieve required score will win the game.
    - Team Deathmatch: 2 teams will fight against each other. The first team to achieve required score (kills) will win the game.
    - Co-Op: All the player will work together to survive as long as they can while fighting against randomly spawned monsters. The goal is to achieve highest possible score.
    - About characters: every character will have common characteristics but each one will have a unique abilities.
- The plan is to make a real-time game that is playable via network against and/or with other players
- Perspective will be a view from above with 3D-like 2D graphics
- The main art style will be pixelated objects
- Each character will have its special and unique abilities
- The game will have self-drawn graphics
- The game will have self-composed soundtrack, effects etc.
- The game will have a simple map editor, where one can set the size of the map, the background theme and insert objects.
- The goal is to create an audio-visually pleasing game that is fun to play with your friends
- Generally, the idea is to create a solid base structure for the game that can be extended and developed even after the course.

2. Class hierarchy



This is a rough sketch of our idea that will probably change during the project. The UML chart will function as the starting point for our development.

- Client will serve as the main window and the center of our game. It holds the menus, the game scene with its objects and will communicate with the network class.
- Network class will hold all the needed implementation to make the game playable via network.
- Menu classes will hold the needed options and settings for each part of the software.
- Scene class will handle the game progress and display all the graphical objects and play audio accordingly.
- Input handler will detect keyboard inputs etc. and pass the information to the scene (and its objects)
- Sounds will contain information about the audio and the related settings. It will pass the information about audio to the Scene.

- Physics will handle the collision detection and collisions between different objects in the scene. Possible movement related effects (e.g. crowd control) are handled by the Physics class accordingly. The movement in the game will consist of friction and acceleration.
- Game Objects will contain all the different objects in the game world. This will function as an abstract parent class for the different subclasses (e.g. Player, Items, Obstacles).
- Map Loader will read and generate the map from a file. The map files will hold map specific information about the map size, background and object specific statements. The files will be divided into separate blocks that will describe each section accordingly.

3. Work schedule

1. Client will be the main priority, and everything is built around it. Start researching network implementation.
2. Scene and menu will be the first additions to the client.
3. Combine the very basic scene and menu to enable launching an empty map from the menu that will function as the base for scene-focused development.
4. Start implementing all the assets related to the scene (audio, physics, different game objects, input handler).
5. When we have a very basic foundation for the game, we will start implementing the network system.
6. Continuous implementation of all the remaining features.
7. Fine-tuning the game.

4. Testing plan

The game will be tested mainly by running the game manually and testing all the features and functionality. We plan to use Valgrind to detect any possible memory leaks and faults.

5. Libraries

For what we have researched so far, we've concluded that we can implement all the currently planned features using SFML libraries. In case we need other external libraries, we will add them to our implementation.

6. Work and its distribution

The work will be divided with individual interests in mind. No feature implementation will be entirely locked in place meaning that if one's interests change during the project, one can start working on another feature. We trust that we can work in such complex environment because we spend a lot of time face-to-face.

Initial emphasis by feature:

Mikko: physics, map loader, input handling, network

Toni L.: audio, graphics/animations, network

Toni O.: network, menu, game objects, development tools/workflow

Simo: game objects, scene, audio-visual design

Generally, bugs should be addressed and fixed before implementing new features (especially bugs related to core mechanics of the game).

7. Useful links and related material

https://www.sfml-dev.org/

https://subscription.packtpub.com/book/game_development/9781849696845