Laser Squad

Project Plan

Members

Kristian Salo, 481014

Katariina Korhonen, 526021

Heikki Jääskeläinen, 525763

Joonas Pulkkinen, 527347

Instructor

Pasi Sarolahti

Course

ELEC-A7150

C++ -programming

Introduction

Laser squad is a turn-based tactics video game originally released in the 1990s. It has several different missions where the player tries to complete objectives such as eliminating the enemies or rescuing a hostage. A player has a team of characters, which can perform move, shoot, pick up or turn operations, thus using the player's action points. The characters are controlled individually.

The purpose of this project is to implement a similar but simplified version of the original laser squad game. The fundamental gameplay mechanics will be largely similar to the original version, but the game will have less diversity in terms of mission, weapon and character options. The game will be implemented as a two-player game, in which the players take turn in controlling their characters.

General Overview

The implemented game will first open up to a menu, where the players are given a chance to choose their teams: at least the characters and weapons are chosen at this point. Also, the mission and the controls are explained. At least one mission will be implemented, but if there is time, others may be done as well. The menu will be navigated with the keyboard, with instructions on the screen.

After the teams are initialized, the actual gameplay will start. The players' characters are spawned on the map and one of the players starts performing operations on his/her characters. The players take turns in controlling their characters until the mission is completed or the other team is eliminated. A turn lasts until the player has run out of action points, or decides to end the turn before that, in which case the action points are saved for the next turn.

A character will at minimum be capable of picking up items, including ammo or food, moving and shooting. A mouse is used to control the movement and attacks of a character, and a keyboard will be utilized for perform operations, e.g. picking items up or switching characters. A mouse will also be used to move the view of the map, since only a part of it will be displayed at a given time.

The GUI of the game will consist of a view of the map with the characters and items in it. It will also have a sidebar, which will show information about the player's action points and the current character's HP, weaponry and ammunition. Also, information about possible items to pick up is displayed there. Below is an example screen capture of a laser squad game. The GUI of this project will be similar to that, but naturally it won't be as impressive graphically and will have less options for character actions and items.

Figure 1. A screen capture of a laser squad game. The basic layout of the GUI in this project will be similar to this: a map view with buttons to move it and a sidebar displaying information about the current player and character. Source: http://www.abandonwaredos.com/abandonware-game.php?abandonware=Laser+Squad&gid=2515

Program Structure

The program is planned to be modular, i.e. different functionalities are separated into different classes. This should allow the program to be easily expanded, if additional features are to be added later in the development. The figure below presents the planned class structure in a UML-style diagram. Only the essential methods and fields are presented, since the planning is still at an early phase.

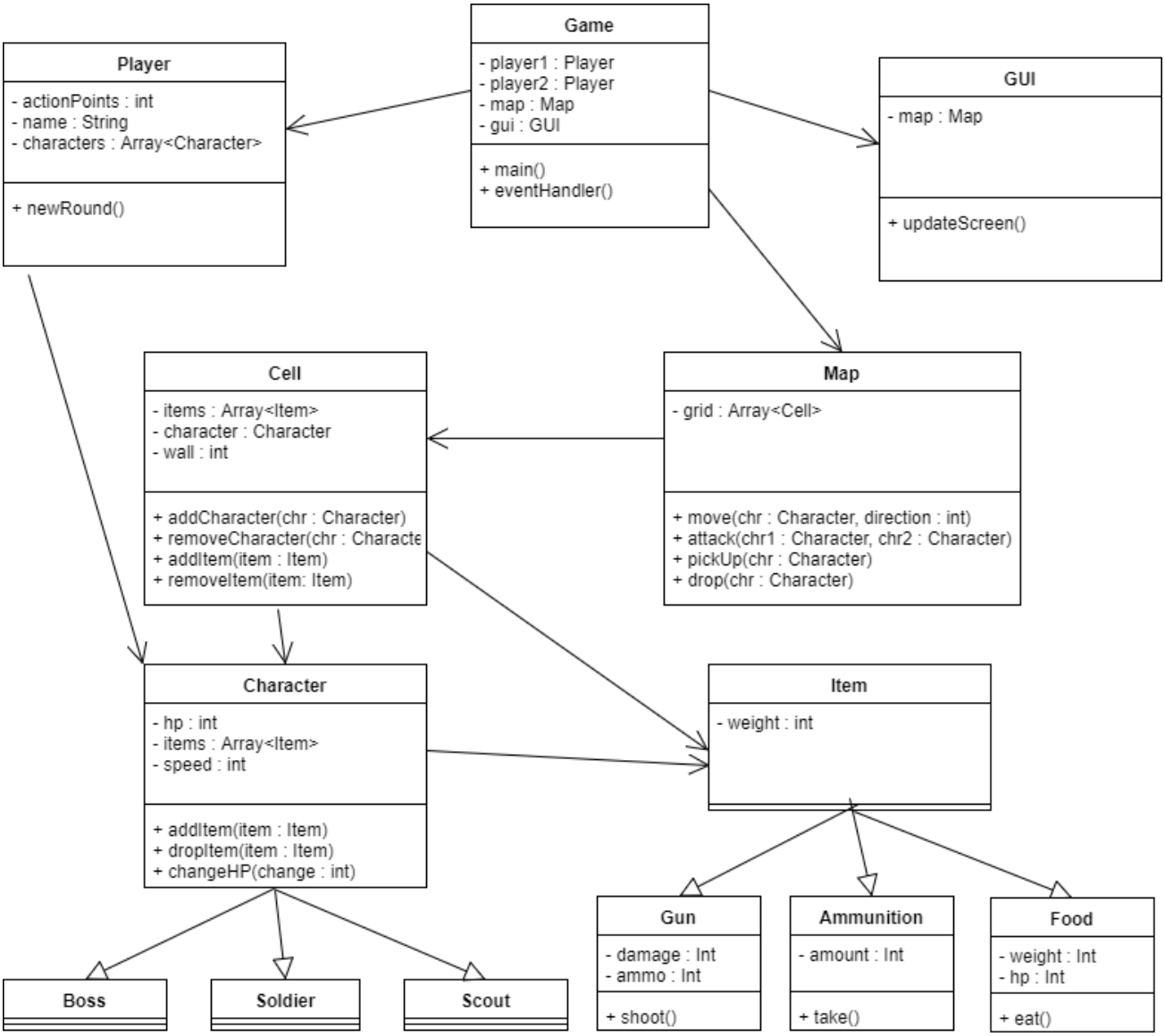

Figure 2. An UML-diagram of the planned class relationships in the Laser Squad game.

- Game

  The game class is going to be the top-level logic for our game, taking care of initializing, running and ending the game. It will hold references to a map, 2 players and a GUI class that is used to draw the game. Game will keep track of the current players' turns and respond to events, such as mouse clicks with appropriate functions from classes character, cell and map.

- Player

  Each Player will have their own Player class, that will keep hold of their current action points, their name and the characters (units) they have at their disposal.

- GUI

The GUI class will be used to draw the game, utilizing some the SFML graphics library for the rendering. updateScreen-function will be used to draw every frame and be called periodically from the Game classes main-loop. The class will also have some functions for moving the camera.

- Map

  The Map class instance will represent the playing field of the game. It holds a 2-dimensional array of cells, that hold all the necessary info on the current tile of the map. Map-class will also have functions that enable characters to act on the map. Move-function allows a character to move on the board according to its speed and remaining action points. A* algorithm will be implemented for this. Attack lets a character to attack a target in range instead. Pickup and drop lets a character interact with items that are in the same Cell as the character.

- Cell

  A cell is used to hold all the information about the point, such as items and characters in it. It has functions to add or remove them.

- Character

  Character is a virtual class used to portray units on the map. All characters have common variables such as speed, health and inventory. Its children will have different initial and max stats. Character has functions to remove and add items to its inventory. It also has a changeHp-function that allows the character to heal, take damage and die.

- Item

  A virtual class that has at least 3 different children: Gun, ammunition and food. Each item has a weight that effects a character's movement speed. Gun, ammunition and food all have different uses but they are kept under the same parent class to keep other classes tidier.

Files and File Formats

The program is going to use standard .cpp files source code files and .hpp/.h header files for building the program. Different classes are supposed to be implemented into separate files for clarity and readability. Either standard Make or CMake is going to be utilized to build the finished program.

The game’s starting position is going to be read from a text file, where the starting position is stored and which can be modified to change the layout of the map. Different levels can also be written in the text file, which will all be read at the start of the program and stored in a container to be used if next level’s map layout is needed. The GUI will then draw the map as stored in the container (2-dimensional list or array).

Unit Testing

Unit testing will be done to the major algorithms that are needed to run the game. The game will need algorithms such as calculating the path of movement for the units, algorithm to change pixel coordinates to game board coordinates and others.

The basic functionality of the game can be seen when running the game and seeing how the map and units look when drawn by the GUI. Bug testing will need to be done by actually playing the game and checking that everything works as planned.

Use of C++ Features

Several advanced features of C++ will be utilized in this project, including exception handling, virtual classes, smart pointers and suitable containers. Exception handling is only needed in rare cases where something unexpected would happen. One case would be when the game tries to load the map from a file, which could fail if the file doesn't exist or the contents are somehow modified to not load the map properly. In that case the program would end in 'Couldn't find text file' or 'Text file doesn't produce a valid map' type of exceptions.

'Character' is planned to be a virtual class which derives the subclasses for the different units, e.g. boss, soldier, scout. That is also the case for Item. Smart pointers will be utilized throughout the whole program to ensure correct memory management. Also, the containers used will be chosen according to the application, e.g. if elements are added/removed or if the size is changed.

Distribution of Work

The class implementations of the image above are divided into each group member so that the workload would be distributed as evenly as possible. The strengths of each group member have also been taken into consideration when deciding the distribution of work.

The distribution of work among the group is the following:

Heikki: Map, Cell

Joonas: Game, reading map from file, etc.

Katariina: Character, Item

Kristian: GUI, Game, Player

Every group member should implement his/her own classes including the methods inside the class. Telegram group chat and private slack channel are used as the main communication

channels during the project. Also, at least a couple of group meeting are to be arranged to discuss the progress and problematic parts.

Schedule

The project schedule is planned to follow the general deadlines for the course. The intention is to have a more or less functional game ready for the mid-term meeting with the advisor at the end of November. In this way, we get the most out of the meeting and can focus on finishing up the game during the final half of the project.

Before the mid-term meeting .hpp and .cpp files should be implemented for all classes. This means that the group has four weeks to complete the initial class implementations and a project plan. The first week is for writing the project plan and the remaining three weeks can be used for writing the code itself. Roughly 30% of the time is estimated to be used for writing the project plan, another 30% for implementing the header files and the last 40% are for .cpp file implementations.