# Project plan C++

Project Manager and Front-end developer: Ella Eilola
System architect: Jaakko Koskela
Back-end developer: Leo Kivikunnas
Software Engineer: Eero Hiltunen

## Project description

Our project will be a tile-matching game in the style of Candy Crush/Bejeweled. We will implement all of the basic functionality required for the game to be playable, such as basic game class structure, graphical user-interface, user input through the GUI, basic game mechanics and physics. These aspects include the implementation of scores, time info, randomized levels, random new blocks, special blocks, several default maps (read from a file), start menu, pause menu, validity of move checking, and map updater.

In addition to this we will implement the following extra features: high score listing, in-game sounds, multiplayer (online/local), special game modes, map editor and AI (capable of playing the game against a human player)

Figure 1: tile-matching game

## Architecture

The program is divided into multiple classes. Each class is defined and implemented in its own .hpp and .cpp -file. This allows for easier parallel editing, since group members can work on separate files at the same without the risk of overlapping code. This type of a structure also simplifies the analysis of a complex program structure.

When the program is launched an instance of the TileMatching class is created. This TileMatching class has a variable stating the state of the program, the program can be in *menu* or *game* state. The GUI window is initialized in the class GUIWindow. The state determines which GUI editing class is used, either StartMenuGUI or GameGUI. This enables the use of a single window throughout the whole program. When the TileMatching Class is created also all the other relevant class instances are initialized, initially it is in the *start menu* state.

Once the game is started, the startGame method of the TileMatching class is called with the appropriate parameters and its state is set to “ingame”. The Game class will handle all the functionality related to running the game. This offers a clear division between the functionalities of these two classes.

When the game is started, the map is loaded from a file and filled with random tiles, after the initialization is ready, the game will begin by moving onto a loop structure that will loop for user input until a valid input is given and then the appropriate method calls are made, this include for example checking a move, making a move, moving to the menu and updating the map.

When the game runs, the Games loop structure will run and the GUI will wait for user input. If the game is paused, the pause menus loop structure will be called and input regarding the pause menu will be handled there. After initial setup the program will flow from one loop structure to another.

There are two possible input methods to choose from. Firstly, we could read the users button presses from the keyboard and let the user traverse the different map squares this way and change tiles for example by pressing spacebar on two subsequent and adjacent tiles while the player is located on the tile. However for this procject the user input will be handled mainly by mouse clicks, since this is more convenient for the player.

Each class that needs to have a graphical representation, such as Game, start menu and pause menu, needs to have their own GUI class to handle user input (mouse clicks) in that context. To read the users input, loop structures need to be implemented in the GUI Classes. This would mean that inputs need to be passed to the Game class. The structure of the game mode classes is still unclear. However, division of the game modes to different classes, allows for setting e.g. win conditions for different modes.

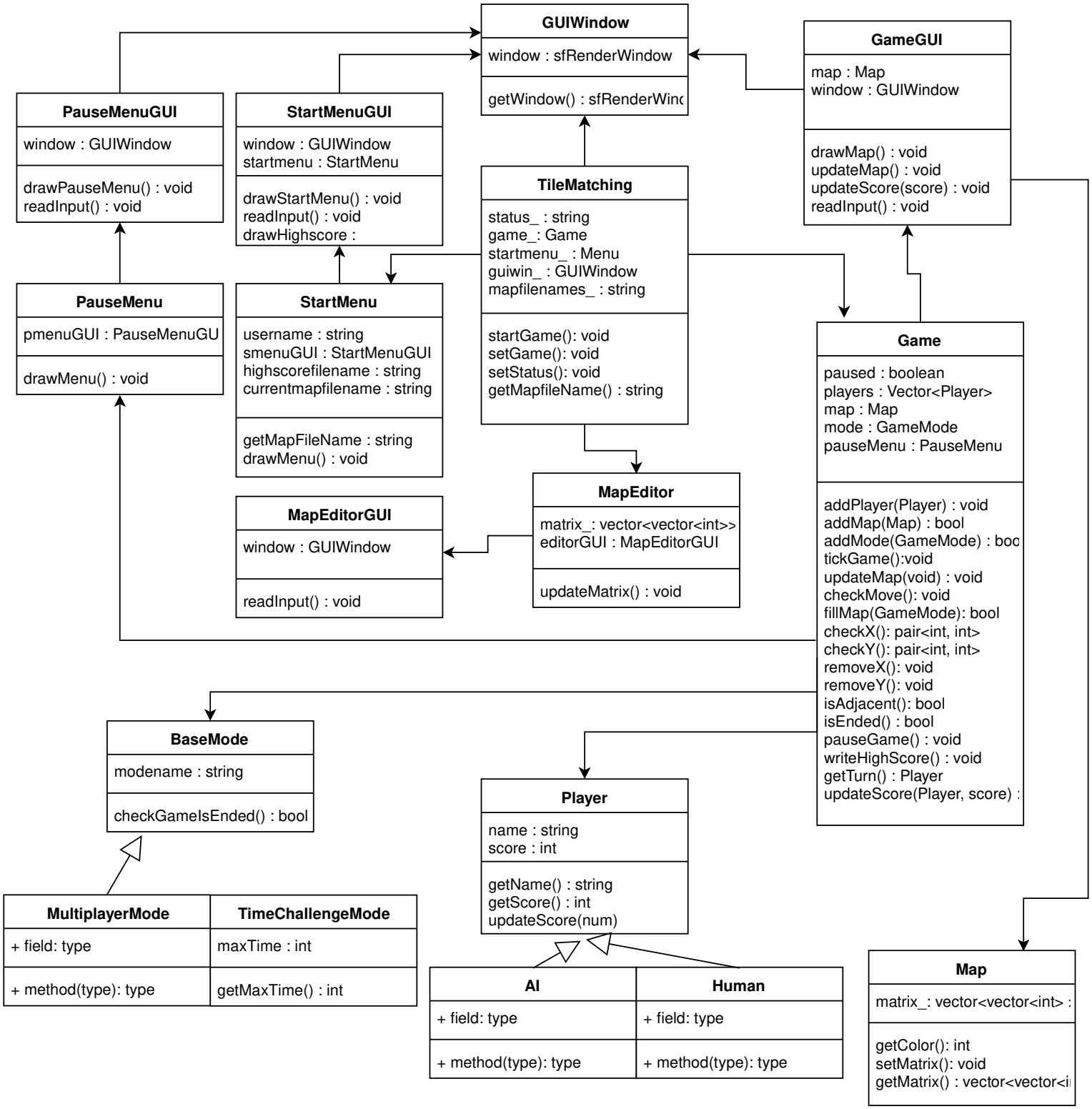


Figure 2: UML graph

## Libraries

For this project we will need at least a graphic library and a multimedia library to handle certain sound effects. For the graphic library we will most probably use the SFML library and we will decide the multimedia library when we decide to add the sounds to our game.

## Roles

The distribution of roles in our group is based on group members skills and personal preferences. Therefore, they’ve been the main factors when evaluating suitable roles for each member that allows us to create the best possible work environment for this project. Thus, Ella will be taking care of the project manager tasks, which include distribution of tasks, problem solving and scheduling. She will be responsible of designing the user-interface and front-end in general. Jaakko is the one responsible of delivering suitable back-end architecture which allows flexible development and debugging of the entire software. Leo is the main back-end developer of the project and his goal is thus to ensure the performance of the back-end. Eero will be making sure that the user requirements are met in the final product and that agreed development method is followed accordingly.

The following roles are assigned this accurately to distribute the burden of software development equally to all members of the group, rather than ending up with someone bearing too much  responsibility. It also needs to be noted that every member of the group will take part in the basic programming work.

# Preliminary schedule

Week 45: Project plan submitted
Week 46: First commits and distribution of tasks
Week 47: Basic back-end for Game, Map and Player classes
Week 48: GUI
Week 49: Game modes (incl. Map editor)
Week 50: Finalizing and documentation