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 such as score, time info, randomized levels, random new blocks, special blocks, several default maps (read from a file), start menu, pause menu, move validity check and map updater etc.

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

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 the menu -state or in the game –state. This determines which GUI is used. 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 TileMatching classe’s startGame method is called with the appropriate parameters and it’s state is set to “in game”. 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 valid input is given and then the appropriate method calls are made, this include for example checking a move, making a move, moving to he menu and updating the map.

When the game runs the Games loop structure will run and 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. Otherwise the user input would be handled mainly by the mouse clicks.

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 Game and Menu Classes. This will make input checking easier since the games classes can be called immediately. This would mean that the input checking and the game board updating would be done immediately in the Game class and the GUI class would be called only to update the graphics.

MapEditor is thought of as a subclass of the GameMode class, with its own specific rules related to the map editor.

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.

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