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

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 menu–state.

Once the game is started (TileMatching classes startGame method is called with the relevant parameters, the Game class will handle all the functionality)

When the game is started the map is loaded and filled with random tiles, after the initialization is ready, the game will begin by moving onto a loop structure that will detect user input.

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 need to have a graphical representation, such as Game, start menu and pause menu needs to have their own GUI class to handle user input in that context (can both the menus have the same GUI or do they need separate ones?)

To read the users input, loop structures need to be implemented either in the Game/Menu Class or alternatively in the respective GUI classes.

The loop structures can possibly be implemented in the Game and in the menu structures this would 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. This, however, would make the Game and menu classes quite cluttered.

Another approach would be to detect inputs from the GUI classes, do the input checking and board updating from the game class through a method call (the GUI would need the Game instance as its member variable) and then the Game class would call the GUI for graphics updates. This approach would make the classes and their functions better defined, but this would increase the back and forth between the different classes

MapEditor is thought of as a subclass of the GameMode class, with its own specific rules

Wrap the whole class base to a TileMatching namespace

After every game instance the relevant information is stored and all the destructors are called

Can the GAME GUI handle the PAUSEMENU – probably not

Can the button presses / mouse clicks be detected from the game class??

Usecase?

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.

Design rationale

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

Distribution of roles