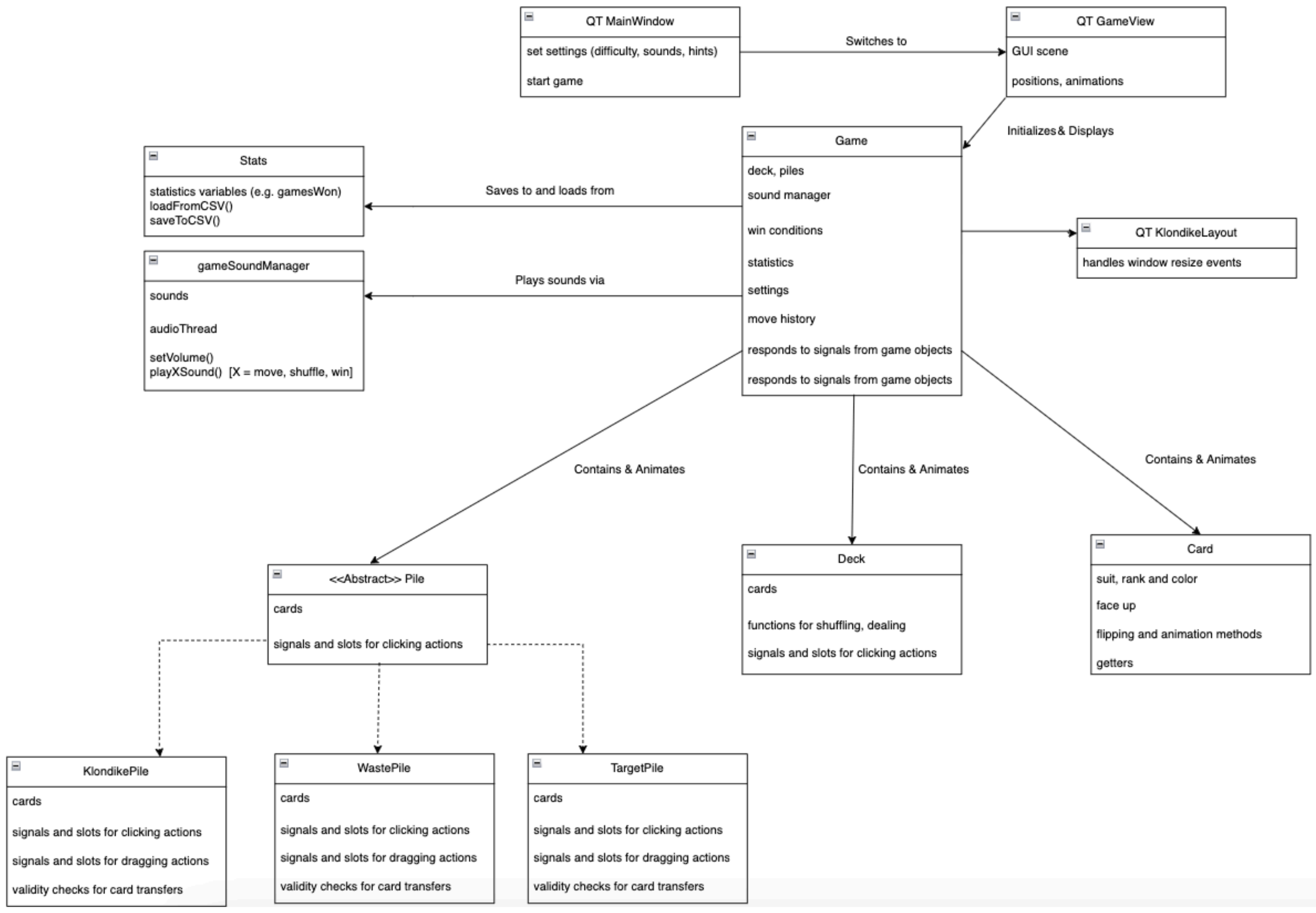
Overview

The program is a simple klondike style solitaire game, with Qt being used for the graphical elements. The game features animations and sound effects and can be played with either dragging or click-to-auto-move controls. The player can undo previous actions and ask the game for a possible move hint. Statistics of the player are collected throughout multiple games and sessions and can be viewed through the statistics panel. The game also has a few settings for the player to change: they can enable/disable hints and hard mode (making the draw pile show 3 cards at a time and disabling auto-move) as well as adjust the volume of the audio. The settings are also persistent over multiple sessions and can be changed in the middle of a game.

Software structure



QT MainWindow
set settings (difficulty, sounds, hints)
start game
Switches to
QT GameView
GUI scene
positions, animations
Initializes & Displays
Game
deck, piles
sound manager
win conditions
statistics
settings
move history
responds to signals from game objects
responds to signals from game objects
Stats
statistics variables (e.g. gamesWon)
loadFromCSV()
saveToCSV()
Saves to and loads from
gameSoundManager
sounds
audioThread
setVolume()
playXSound() [X = move, shuffle, win]
Plays sounds via
QT KlondikeLayout
handles window resize events
Contains & Animates
Contains & Animates
Contains & Animates
<<Abstract>> Pile
cards
signals and slots for clicking actions
Deck
cards
functions for shuffling, dealing
signals and slots for clicking actions
Card
suit, rank and color
face up
flipping and animation methods
getters
KlondikePile
cards
signals and slots for clicking actions
signals and slots for dragging actions
validity checks for card transfers
WastePile
cards
signals and slots for clicking actions
signals and slots for dragging actions
validity checks for card transfers
TargetPile
cards
signals and slots for clicking actions
signals and slots for dragging actions
validity checks for card transfers

The project emphasizes modularity throughout its general architecture. It comprises of three pure QT window classes: mainWindow, gameView and KlondikeLayout, purely responsible for visualizing the game and menus. GameWindow contains the scene, which is the top-level object in the QT hierarchy system, and thus handles the bulk of the graphics-related memory management. KlondikeLayout defines the layout of the window, and handles window resize events.

At the core we find the Game class, which contains and initializes the entire game instance, along with the relevant objects (cards, deck and piles). On start, the game initializes the objects, sets up the tableau and deals cards from the deck into piles at random. After the setup, the program basically enters a loop of players initiating a move by clicking or dragging cards, checking if the move is valid, and if so, making the necessary move both logically and graphically. Finally, it checks the winning conditions of the game, and displays a win screen with the statistics if the game has been won.

The other game artifact classes are responsible for handling their respective functionalities like shuffling, move validation and transfers, card flipping, et.c. Each class also has a function for updating its graphical representation based on the logical content. KlondikePile is an exception, and contains an auxiliary gui class KlondikeLayout

There exist two auxiliary class: gameSoundManager, that implements the miniAudio Library, a light-weight audio library for playing sounds on card clicks, deck recycles and winning. It uses multithreading for minimizing delay when playing sounds while performing other operations. Statistics class contains functionality for continuously collecting statistics, which is persistent via storage in a .csv file.

The structure is modular for easily adding new Solitaire game modes. One simply has to define new game and pile classes with different rules for valid moves, and different win conditions. While we did not have time to make Game an abstract class, this would be the way to continue forward.

Instructions for building and using the software

Testing

Each core class of the game was tested initially using unit tests. The graphical features were mainly tested through trial and error to see that it looked okay. For example dragging and dropping was tested with trial and error. We used the catch2 library for our unit tests and they can be found under the test folder.

Card:
The card class was tested such that the basic properties of a card functioned (e.g. suit, color and rank). After that, flipping the card is tested (facedown -> faceup -> facedown). Lastly, its string representation is tested. All tests passed.

Deck:
Initialization and drawing to the waste pile was first tested, then moving cards back to the deck was tested. All tests passed.

Klondike Pile:
Card validity is checked (so what can be placed on this pile) and some visual aspects that could be tested were tested. All tests passed.

Target Pile:
Card validity is tested (so same suit and increasing rank). Then some visual behaviour is tested. All tests passed.

Waste Pile:
Behaviour when drawing from a deck and visual aspects. All tests passed.

Game:
Here a lot of interactions are tested. Initialization (Deck, Waste Pile, Target Piles and Klondike Piles). Moving and scoring is tested by drawing some cards and moving them around. Winning is tested. All tests passed.

Valgrind:

Valgrind was used to check for memory leaks. There were no memory leaks from our code. Some memory was still reserved when the program was terminated, but that is due to the external libraries used.

Work log

Sprint 0:
Sprint 0 was used to get the development environment to work with each member of the group. Everyone installed the Qt6 framework and configured together the cmakelists.txt file to allow smooth compilation. Ukko configured the Qt creator. First hello-world window was created.

Sprint 1
Sprint 1 was spent creating the logical classes such as Card, Pile, TargetPile, KlondikePile, WastePile, Deck and Game. The first iteration of MainWindow was created as well as graphical wrappers for the logical classes.

Divided tasks:
- Kasper worked on card/piles classes
- Robin worked on game class
- Daniel worked on GUI solutions
- Ukko worked on the mainwindow using qt creator

Sprint 2
First half of sprint 2 squashed the logical and graphical classes together and connected the elements to the main window. The other major task in sprint 2 was to get the elements moving on the screen and make the application work like a proper game of solitaire.

Divided tasks:
-Kasper and Daniel squashed the graphic and logic classes
-Robin worked on the game logic
-Ukko interconnected the graphics elements
-Collaboration with GUI related solutions

Sprint 3
Sprint 3 added a lot of polishing and additional features to the game. Sounds and animations were added, statistic gathering, settings and GUI navigation options.