Design Document for Sudoku Project

by Jay Gao, 2016011251

Introduction

Purpose

Write a QT gui and a Sudoku core to generate a sudoku game.

Scope

Programming training

System overview

This project generates a Sudoku chess board and then let the players play this game.

System Architectural Design

Architectural Design

This project is divided into two parts. One is the Sudoku board generator, and the other one is the GUI controller.

Decomposition Design

Generate class

This class generates a Sudoku board with a single solution. It uses *map* algorithm to generate a new class according to an existed Sudoku. Thus, the sudoku has a singal solution. It firstly generate a random figure from one to ten to map the other figure. Then it removes some numbers in random order, while testing if there is still only one solution after deleting this number. If not, this number will be returned.

mylabel class

This class is inherited from `QLabel`. Since there is no `click()` to reply the clicking of the players, I creat a new class which can not only have the main features of `QLabel` but also can have the `click()` as the `SIGNAL`. As a concequence, the click of the chess board can be monitored.

dialog class

This class is in charge of the main usage of the UI. The main functions are as follows.

- `timerEvent` It is in charge of the time. By using it, we can record the time of the game and change the state of the game. (begin, not begin, pause, end, replay)

- `paintEvent` It draws the grid of the chess board.
- `QMediaPlayer *player` It adds the background music tor the game.
- `bool isFinshed` It judge the end of the game.
- `QSignalMapper` It can transfer arguments to the SLOT function.
- `std::vector<std::vector<int>>>history` It stores the history so we can realize undo and redo.
- `QComboBox *comBox` It can choose the levels.
- `playMusic()` It add BGM to the game.
- `pauseIt()` It can pause the game.
- `replayIt()` It can replay the game.
- `gridClick(int)` It is the SLOT function of the click of the chess board.
- `numberClick(int)` It can fill the number to the grids.
- `gridDelete()` It can delete the number of the grids.
- `markGrid()` It can mark the grid.
- `helpMe()` It can help you play the game.
- `undo()` undo
- `redo()` redo
- `chooseLevel()` choose level

Data Design

Data Description

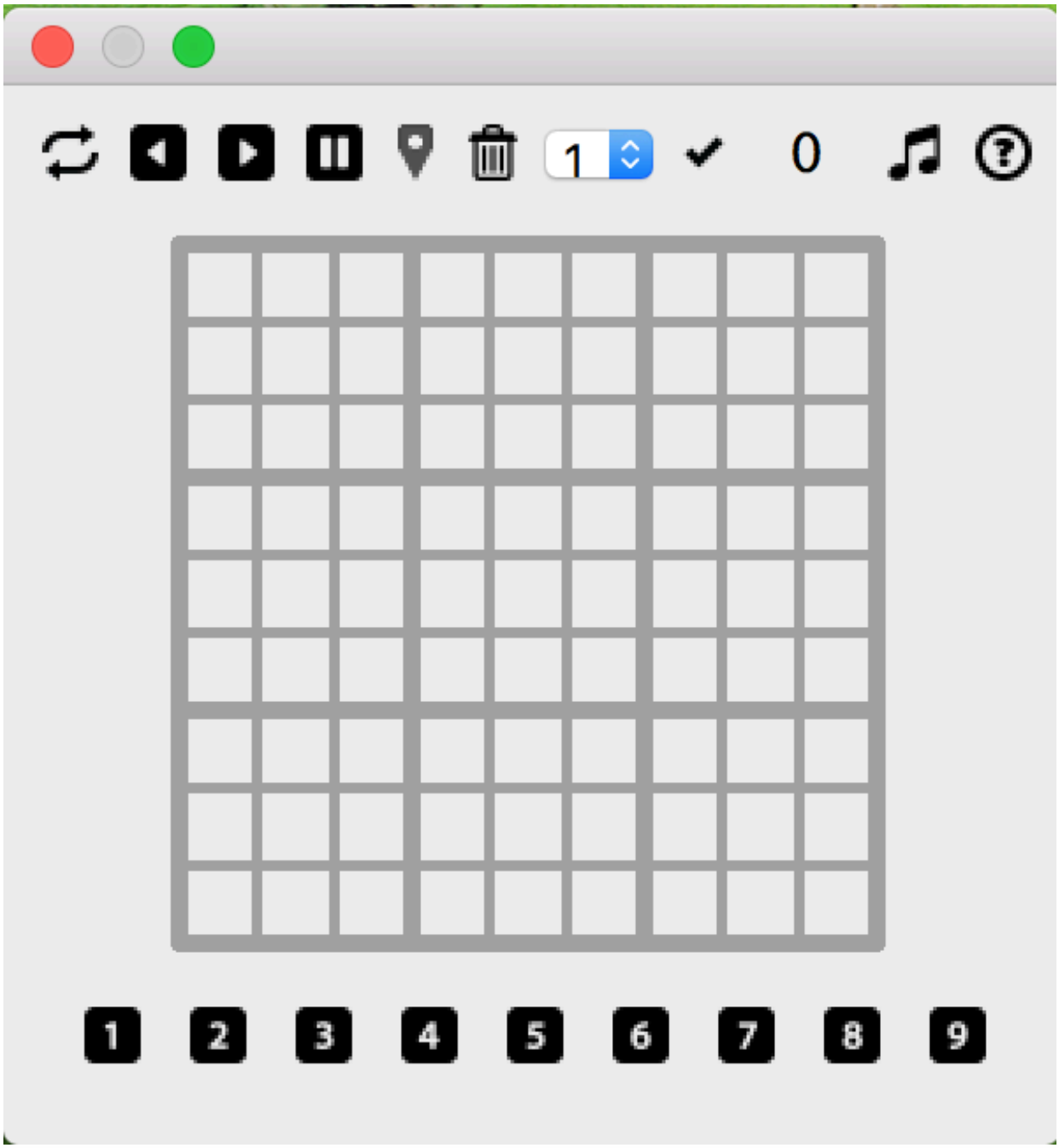
For multiple numbers may be stored in one grid, an advanced way of storing the board must be there. I use binary bits in a `int` variable to show which numbers are in this grid. The `history` variable is very useful since it complex declaration `std::vector<std::vector<int>>>`. Every operating can be store in one `std::vector<int>` which contains three `int`. The 1st is represent for the opration ,the 2nd is for the grid and the 3rd is for the number changed in the grid.

Human Interface Design

Overview

All functional slots are connected to the `Dialog` class itself, and then it operates on the separate grids or the buttons. Button pushing actions are connected to the slot functions and the style of them is design by me (I think they are beautiful~).

Screen Images


1
0
1
2
3
4
5
6
7
8
9