Homework 4

We will be building a game of CHUTES & LADDERS.

See Canvas for due dates

130 points

Objectives:

- Implement a GUI with elements that interact with one another
- Understand timer mechanics and how to advance state automatically
- Become more familiar with the observer pattern in general (signals/slots in Qt)

Credit:

- **Deliverable 1:** HW4_design.pdf
- **Deliverable 2:** All the .h, .cpp, .pro, .ui files for your Qt project (**do not include your .pro.user file**). Github link if you are working with a partner.

Instructions:

You may work in a pair though I encourage that you attempt this project on your own. If you are working with a partner, create a **private** github repository and add **chaitanyab2311** and **sreeshanath** as collaborators to it. You will submit this link along with the rest of your submission on Gradescope.

YOU MAY NOT POST ANY OF THIS CODE IN A PUBLIC GIT REPOSITORY

Deliverable 1: Design document

First, carefully read the entirety of this design document.You will make a plan of which classes are in charge of what, how they will communicate (what methods they will have, what signals will they emit, what slots will they have), and what data they will control (what fields will they have). You should also indicate what signals will be emitted by UI elements and which objects will respond to these signals.

1. Use the plot project that we'll be working on in class from Weeks 8, 9 & 10 as well as the Qt documentation to help guide your design.

2. Your design document does not need to be a 100% final plan, but should account for all features that you plan on having.

Game rules (modified from original):

Note: Each of the operations are explained in detail in Program Flow.

1. Each player has a pawn assigned to them. These pawns will reside in a container outside the board.
2. To get placed on the board, the player should get a 6 upon rolling the die(s). They get placed on tile 1 to get started.
3. In each turn the player can do one of the following:

- undo a previous move
- roll the die(s) and play the move based on the number generated
- re-roll the die for a different number
- quit the game

4. If the player does not make their move within 10 seconds, their turn should be skipped and the next player will have a chance to play.
5. If the player lands on a tile that is the bottom of the ladder, they should automatically be moved to the top of the ladder.
6. If the player lands on the top of the chute, they should automatically be transported to the bottom of the chute.
7. The player to reach tile 100 first is declared the winner.

Program Flow:

Number of Players

Your application should load a dialog at the start that allows the user to choose the number of players for a given game. You can have a minimum of 2 to a maximum of 4 players. Users shouldn't have the option to choose outside of this range.

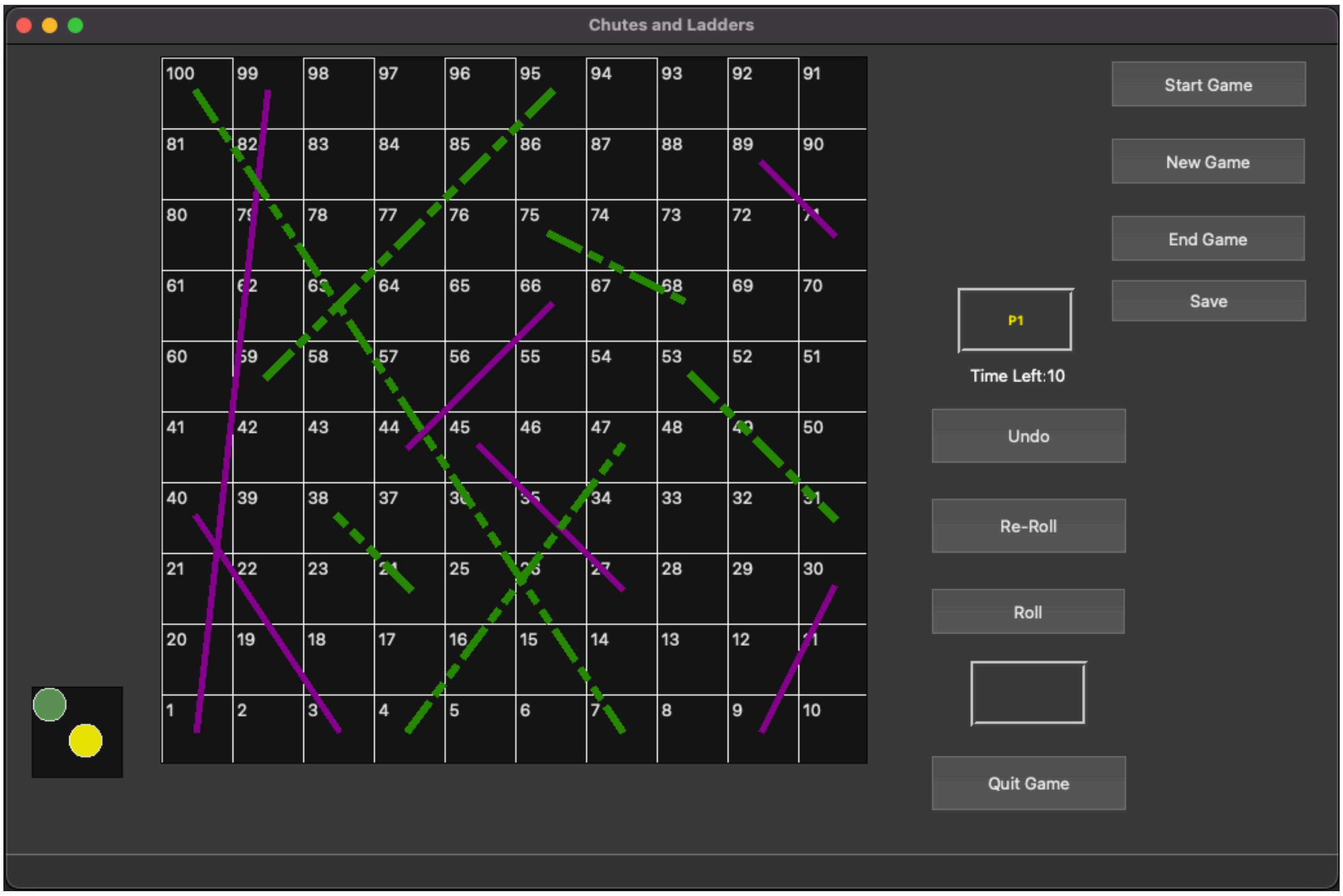
Hint: sliders

Player Information

We intend to store player information to allow us to track their scores and be able to generate a leaderboard. For this purpose, the users should be able to enter names only for the number of players they have chosen for a given game. For eg: If the user chose a 2-player game, then the next prompt should allow the user to enter names only for 2 players.

Board initialization

The board is then loaded onto window. This board should be randomly generated with a minimum of 6 chutes and 7 ladders. They can be placed anywhere on the board and can be of variable random lengths. The only caveat is that no tile on the board can serve as an endpoint to more than any one chute or ladder. Below is an example of the board:



Here the darkMagenta solid lines denotes the chutes and the darkGreen dashed lines denotes the ladders. We have used a `QPen` object to define the look of the chutes and ladders. You can choose your options to determine the appearance of the chutes and ladders.

Our playing field:

We will play on a 2d square field.

Graph: Underneath our playing field is a graph that tracks the tiles on the board. Each tile has a number associated with it and a player can land on any tile on the board depending on the number from the rolled die(s). You may want to track if the tile is an endpoint for a chute or ladder. It will help with board initialization.

Rolling the die

We're simulating rolling 2 die(s) as in the original board game. You may choose to represent this using either 1 or 2 labels/textboxes.

- If you are working with 2 die(s) - generate a number between 1 & 6 for each die and you will sum these numbers to get the final count of moves to make
- If you are working with 1 die - generate a number between 1 & 12 and that would be the number of tiles the player can move forward to.

Start the game

User needs to click on the start button when they are ready to play. The players will play this game in a round robin fashion. Each player has 10 seconds to make their move. All the player pawns will reside in a widget outside of the board. To get their pawn placed on the board, the player has to roll the die(s) and get an exact 6. This will place them on tile 1 on the board. Except for this move, the player should be able to undo all moves that they may play in the future.

Player Turns

You must indicate on the UI who the current player is. You can use color coding to link a player to their pawn as seen in the image above.

If the player is unhappy with their move in the previous turn, they can undo it - even if it means moving up the chute or down the ladder. Each player has a maximum of 3 undos in a single game. Undo-ing a previous move in a turn is optional for the player. They can proceed to the next step without this.

The player can choose to roll the die(s). If the player is satisfied with the number rolled by the die(s) they can choose to make the move to the next tile. If not, they can re-roll the die. A player can re-roll the die for a maximum of 5 times in a game.

A player could use multiple undos and re-rolls in a turn. However, if the player reaches the limit for undos or re-rolls, those options should be disabled for the player's remaining turns in that game.

The player also has the option to quit the game. If upon a player quitting, the number of players is reduced below the minimum, the game should automatically end with a warning dialog with an appropriate message to let the user know.

Irrespective of their sequence of actions, each player has 10 seconds to complete their turn else their turn is skipped. You should display a timer on the UI. It could be simulated using a label that updates the text/number each second. Choose a medium to indicate to the user if their turn was skipped.

Visualizing player movement

When the player's pawn is being moved on the board, it should step on each tile along the way till it reaches the destination tile. This is applicable when the pawn is moving forward in a regular move or moving backward while undo-ing a move. The only exception is when the player is moving up/down the ladder or chute.

When the pawn is being moved on the board, all actions for the player should be disabled till the move is completed. Note here that when the move is completed, the player's turn is considered done and it is the next player's chance to play.

OPTIONAL : You can add appropriate labels/dialogs to indicate when the player is moving up a ladder or down the chute.

Ending the game

The game ends in one of the following situations:

- If a player reaches tile 100, the game ends with declaring that player the winner.
- If the number of players reaches a value below the minimum
- User chooses to click on "End Game"

Leaderboard

You will track the player name, number of games won, number of games played, and % of wins for each player in a csv file. For this game, treat the csv as a read-write database for player statistics.

The user should be able to click on a button to view the leaderboard They should be able to view all the statistics for each player, stored in the csv file, arranged in a descending order of % wins. You can choose which widget to use to display this information.

If the number of player exceeds 10, you can choose to display only the top 10 players. You will, however, retain information about all players that have played the game across multiple sessions in the csv file.

Extra Credit (20 points)

For extra credit you need to do both of the following tasks:

1. Save the game state: If the user decides to save the game to be played later, you should save the below in csv file(s):
   a. players' information and state: names, positions, number of undos, number of re-rolls for each
   b. board state: since each board is randomly generated, you should save the positions of the chutes and ladders on the board
2. Load a game from the given csv(s): You should be able to load a game with the information that was stored in the previous task and the players should be able to proceed with the game from where it was left off.

Object Design:

The design of this program and the underlying objects is entirely up to you.

Tips:

1. Items that do not inherit QObject or include the Q_OBJECT macro cannot emit signals, nor have slots.
2. QGraphicsScene has an ItemAt function that you can use to tell what item was clicked in your scene. The QGraphicsItem must implement *both* the shape() and the boundingRect() methods correctly to be properly detected by ItemAt().To do this, you will need to subclass QGraphicsScene so that you can override whatever event methods (such as mousePressEvent) that are necessary.
3. Take a look at the QTimer documentation. You may choose to use QTimer, or you may use one of the alternatives; our game doesn’t require single-shot timers or signals (from the timer).

4. Use QDebug. The easiest way to use this class is to import it and direct whatever you want to print to qDebug(). (e.g. qDebug() << “click”;) QDebug handles correctly routing the output to the console so that you don’t end up with the weird behavior that can happen with cout, such as the console not updating until after you exit the application.
5. Feel free to use any parts of the plot project as inspiration.
6. We highly recommend using only QGraphicsScenes and QGraphicsItems to visualize your board, player pawns, chutes and ladders. We do not recommend using QTables or QGraphs. You may use QPixMap to give your game a better visual appearance but indulge in it only after completing a basic implementation of the game.

Rubric

Criteria	Description	Points
Design Document		10
UI	Contains all UI components and dialogs to accept number of players and player names	30
Rolling the dies(s)	The total should not be less than 1 or more than 12	5
Start the game	Player pawn is placed on board at the right time	10
Player turn actions		45
	Update current player information	5
	Undo	15
	Re-Roll the die(s)	5
	Make the move based on number rolled on die(s)	10
	Quit the game	10
	Disable player actions when pawn is moving on the board	5
End game	Games ends appropriately under specified circumstances	10
Leaderboard	View leader board with player stats	15
Extra Credit	Save and load games	20