University of Glasgow | School of Computing Science

Adavanced Programming(IT+)
Assessed Coursework

Option 1

Draughts

Jialiang Song / 2410536S

1/4/2020

Description

1. Aim

Develop a distributed system that can build and support multiplayer games (two-player games).

The system should contain a Java program that acts as a game server and a Java program that acts as a Swing-based game client. Each client should use sockets to communicate with the server. Clients should not communicate directly with each other, but through the server.

It should be design an application-level agreement for the system, determine the communication mode between the client and server, and then use it for implementation and test procedures. Every time the client connects, the server should start a new thread. Each thread should be responsible for serving a customer. Prevent any potential race conditions and deadlocks by using tools such as synchronization methods and locks.

Each player should be able to contact at least one other player through the server and play games involving multiple rounds or rounds. Allow ongoing games, that is, once a game has been played, allow the game to restart.

2. Rules

Fig. 1 Sample of chess board

1. The chess board is 8*8, 10*10, or 12*12, based on the different rules, and on this project, I will choose the 10*10.

2. Three rows of chess pieces, four in each row, a total of 12 pieces on each side

3. All of the pieces will only move in the dark (black) grid.

3. Mechanisms

1. Soldire’s move
   The soldier could only move forware one side diagonally, not backward.
2. Soldier’s jump and eat
   The soldier's jump and eat is: two black and white pieces are closely connected on a slant line. For example, when it is the turn of a certain party, there is an empty space before and after the opponent's piece. The chess pieces eat the skipped pieces and remove them from the board.
3. The soldier's continuous jump and eat
   A soldier's consecutive jump is to jump over the opponent's chess piece and then meet the skippable chess piece, then you can jump in succession, eat the skipped chess piece, and remove it from the chessboard once. [Note: The soldiers ca n’t move backwards, but when jumping or continuous jumping, they can jump back or eat children.
4. The rising of soldiers
   Before the start of the game, the pieces placed on the board by the two sides are soldiers. During the game, the soldiers go to or jump to the other party's bottom line and stop, then they can be promoted to "king". But must have one round gap before the new King could move.
   During the game, if the soldier does not stop when he reaches or jumps to the opponent's bottom line (that is, he passes by halfway), and he cannot be promoted to a "king".
5. King’s move
   King can advance and retreat on any slanted line, and there is no limit to the number of cells. (Movement similar to chess)
6. King’s jump and eat
   When the king and the opponent's pieces meet on the same slant line, no matter how many positions are apart, as long as there are empty positions before and after the opponent's piece, then King can jump over and eat the opponent's piece
7. King's consecutive jump and eat
   The situation of King's consecutive jump and that of soldiers' consecutive jumps is basically the same, except for unlimited distance.

Implementation

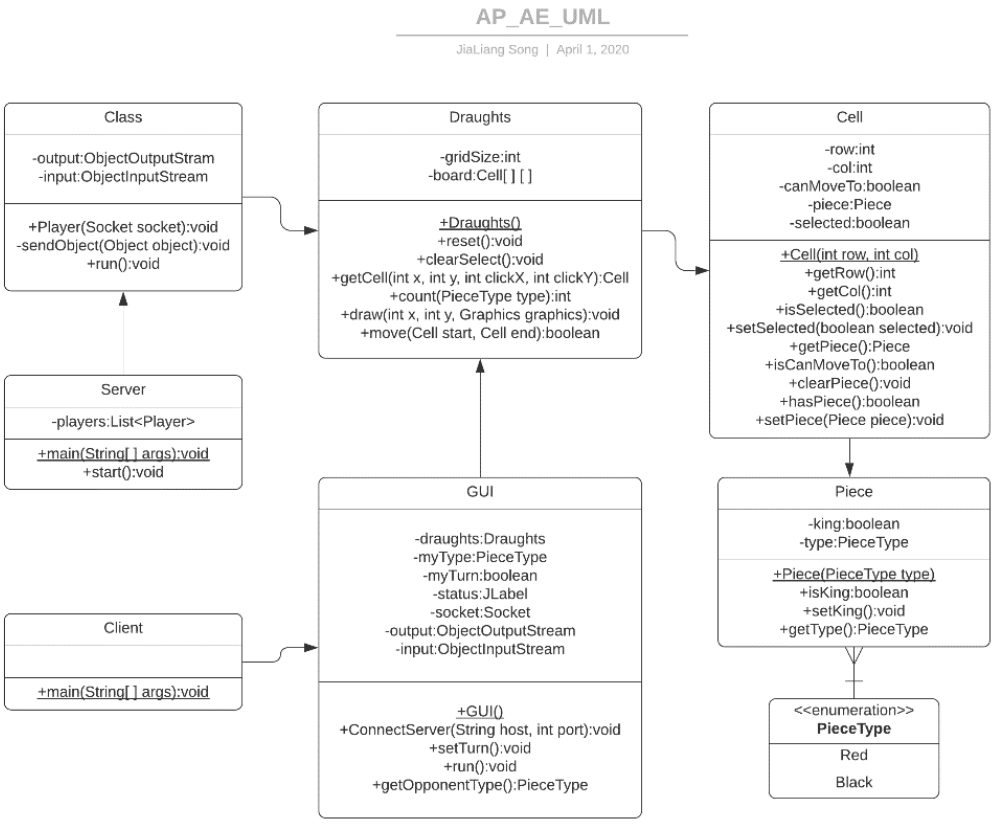

Fig.2 UML Diagram

PieceType:

This class using an Enum represent types of piece type, Red or Black.

Piece:

This class represent a piece, each piece has its own type (PieceType) and state (whether it is King...or Queen maybe? I/m not sure..There are different rules in difeerent instructions)

Cell:

This class represents a grid (cell) on the chessboard. Each grid has its own row and column numbers and you can set or remove pieces on the grid.

Draughts:

The class Draughts represent the entire chess board. It contains a 10*10 Cell matrix. It will also judge whether a vertain piece moves legally.

GUI:

The class GUI drew the chess board and handle user click events. Also responsible for interacting with the Server.

Client:

The Client class (is a Client...) that create and display the GUI.

Server:

This class contains the user list, responsible for accepting user connections and creating service threads for each user.

Player:

It is the the thread class responsible for interacting with users, and it will be created by the Server class.

protocal

1. First, the game is initialized. The initialization process will determines who is red and who is black.

   Server -> Client : StartRed
   // Tell the client that it's red

   And then,

   Client(Red) -> Server -> Client(Black) :StartBlack
   // Red Client talk to another Client that, "you are Black"

2. During the game
   The pieces of red and black will move alternately until the end.
   And after each operation (move), the system will send the Object of the chess board (Draughts) to another person.

   Client -> Server -> Clinet :Draughts Object

Appendix

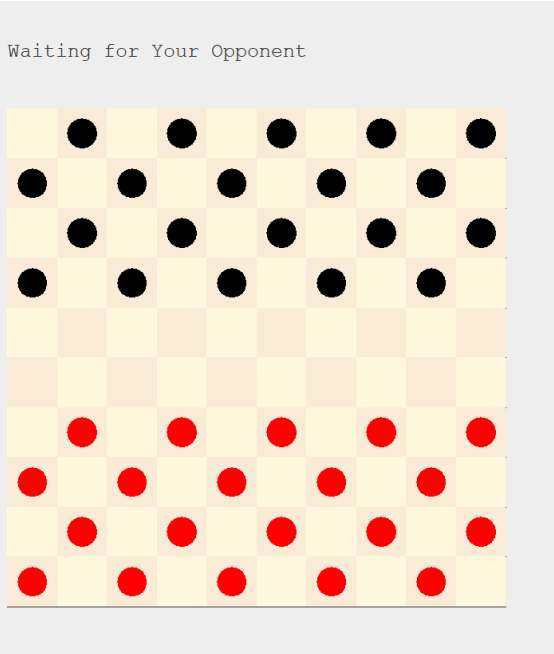

Fig.3 Initialize the game

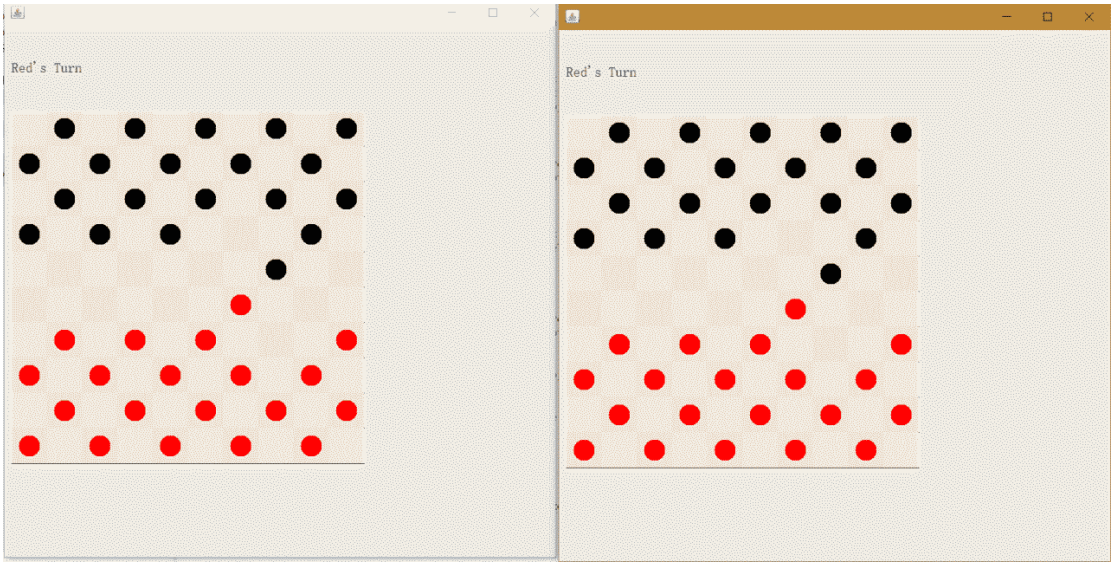

Fig.4 Game running