Izmir University Of Economics

SE360 – Advances In Software Development

Term Project

by

Doğa Ünal

Sinan Küçükyılmaz

Summary

Introduction

Our project is a classic chess game which can be played in single coputer with 2 players or from separate computers in LAN.

Brief Implementation

Every game session in the project starts with a ‘Game’ class instance. Game class has a chessboard which is implemented in ‘Chessboard’ class. Chessboard class represents a 8x8 chessboard. Chessboard class has a 2D array of ‘TableSquare’ instances which represent squares on a chessboard. TableSquares can be empty or contain a ‘Piece’ instance which represent any piece in a chessgame.

In GUI , TableSquares are implemented as ‘JButtons’ and Pieces are just background images of those TableSquares. According to 8x8 TableSquares array , drawing method of the chessboard displays them on the screen.

Implementing the squares as buttons made interaction management easy , espacially in LAN games.

a-Interaction With The Board

When a new game starts , a new chessboard gets created with their pieces on it in default sequence. If a player clicks a square which is also a button , ‘SquareListener’ class which is user defined actionlistener for TableSquares gets triggered. SquareListeners check whether the clicked square has a piece on it . If it has a piece , then displays available positions to move for that piece as blue squares. After that, when player decides where to move and click , then the piece gets remove from the current square and a new piece which is a same type of chess piece with deleted one gets created. The player can also change his mind and can click another piece to move.

In LAN games , when a player clicks a square , then ‘x’ and ‘y’ indexes are sent to the other player. Then , the receiver player process the message and gets those indexes. SquareListener of the square which is located in the array with those indexes invokes its ‘actionPerformed()’ method. Therefore , when the client clicks his own chessboard , it also triggers the server player’s chessboard as if they are playing in same computer.

b-Gameplay Logic And Algorithms

1 – Available Positions

Most algorithms in the game are based on ‘availablePos’ list in ‘Piece’ class. According to type of that piece (queen,king,pawn etc.) , the piece fills its list with those specific squares. For example , a knight can move as ‘L’ shape , so if a knight calculates their available positions , its ‘availablePos’ list will be filled with squares which are ‘L’ far from it.

2 – Movement

After selecting a chess piece , if the next clicked square is on its ‘availablePos’ list , then it can move there.

There can be special situations which a piece can not move . For example , there is a pawn which is in charge of protecting the king by blocking the way of a enemy queen. If the player tries to move that pawn by eating some enemy piece or by moving into an empty square , it will not move in any case.

3 – Is There A Check Position ?

Checking a situation of a chess piece threatening the king is important in chess logic. If that ‘check position’ occurs , then the player whose king is in danger has to escape that situation by eating the threatening piece , blocking its way or simply move the king somewhere else if it is possible.

In ‘Chessboard’ class , there is a method calles ‘isThereCheck()’ which is scans the array and checks every chess pieces if there is a chess piece which is a trouble for the enemy king. This method usually called after a piece is moved into another square. The piece which threatens the king also is saved into a field called ‘checker’ in ‘Chessboard’ class.

When there is a ‘check situation’ , then the info field which is located at the bottom of the board whill display a message such as “black knight threatening the king!” and movement of the unrelated pieces will be limited until that situation disappears.

4 – Is It Over ?

As if mentioned in section three , If ‘check position’ occurs , then the player whos king is in danger has to escape that situation by eating the threatening piece , blocking its way with another piece or simply move the king somewhere else. However , if it is not possible to use one of these solutions , then the king is trapped and the game is over. In class ‘Chessboar’ , a method class ‘endOfTheGame()’ checks those situations. Firstly , it checks whether the king can eat its foe or not. If it can eat the threat, then there is no ‘checkmate’. Secondly , it checks if the king can step away from the threat. If it can , then there is no ‘checkmate’ condition either. Finally , the method checks if there is any ally piece of the king can eat the threat or block its way. If there is , then no ‘checkmate’ again.