Final Deliverable

Design Document

Group 13

System Design:

For our final deliverable we have prepared a fully functional game of checkers. The game includes a single and multi-player experience as well as score boards to document and save the wins and losses. These saves will only be documented for single player games. Upon launch of the game the user will be taken to the main menu of the game, where the user will then choose whether he/she would like to go play the game, either in single or multi-player, view the score boards or quit.

The single player experience is played against a fairly smart AI. This AI takes advantage of the MiniMax AI system allowing for an intermediate difficulty. Whenever a user makes a mistake, like not moving the proper piece, the messages will be shown on the command prompt. As well, our boards show which piece on the board is the current starting position by a highlighted square. If the player wishes to click on another piece to move instead, the player must have the specific piece they wish to move highlighted.

The multi-player experience is quite similar to the single player experience, the only difference is that there no longer is an AI; rather the opponent is a human player. The errors will also be prompted on the command prompt.

The score board contains all the wins, losses and games played from the single player experience. The way that this scoring system works is that every end game, whether it be a win or loss, will always append a game played to the scoring file and then the win/losses will append accordingly. The data is saved onto a file under the src/ directory using Java Serialization. Within our testing units there is a score board tester, this simply gets the scores respectively, since we do not wish to overwrite any scores. If the file does not exist or is empty a new file will be created and all scores will be reset to zero.

Design Choices

Our project was written following the MVC Design Paradigm- Model, View, Controller. In this pattern, three distinct areas exist and interact with each other only as needed, being largely self-contained. In an ideal scenario, each component is usable as a standalone component without dependency on the others.

The Model is the game itself, holding the game logic, board state, AI algorithms, et cetera. This Model is not aware of the View or the Controller in any way.

The View is where the Swing library is utilized to represent the Model. The View is only aware of the Model's state and will make changes as the Model changes. This is the front-end of the program, what the User will see.

The Controller is the component with the highest level, being aware of both the Model and the View. This component's functionality involves receiving input from the user and giving it to the Model or the View, and can be considered the bridge between the User and the Program.

Within the Model, the game is launched and played. A main loop will alternate between which Player's turn it is, and wait for them to perform a movement. A thread sleep is added to prevent system lockup as the game waits for input to be given.

Input is checked at multiple levels- both upon selection of a Location, and request for a movement. The input is checked for validity of location and legality with respect to Checkers' rules.

The Player sends the request to the Board to move a piece (and if relevant, remove any intermediate pieces that would be jumped over), and the game proceeds.

The game will end on two conditions- A player has no remaining pieces, or a player can no longer move. At this time the winner will be determined, and if in Single-player mode, the Score component updated accordingly.

For an AI system, the Minimax algorithm was chosen for its effectiveness with symmetrical state-based games. In this system, a tree of possible movements is recursively searched, up to a maximum search depth of 4 movements between both players (two movements each). At each 'node', there exists some state of the board; each node must be analyzed for its 'score'.

Score is determined based on remaining pieces, number of kings, piece location, and ability of a Piece to move or jump. Because Checkers is symmetrical, the score of one player is equal to the negative value of the other player, which the search algorithm accounts for. Score is always taken from one player (The 'root' player's) perspective.

If a given node has a higher score than the last 'best' score, its score will be set to the highest. This score will be passed up the tree as the recursive function unwinds, and eventually the score will be assigned to some 'root' Node, which is a node that a player could reach from making exactly one movement. The best movement will then be given to the AI Player.

MiniMax is a very effective algorithm for games like checkers, however improvements could be made to the algorithm for a more difficult AI (for example, accounting for the game time it takes to reach some given node). For the purposes of our program, we found the current state to be very sufficient and is definitely a playable product.

One quirk of the MiniMax algorithm is the tendency in the late-game to play an overly defensive game, as a stalemate is considered to be neither unfavorable or favorable. However, to solve this issue would require a more detailed level of AI logic which we did not feel that we had sufficient preparation time for.

The Swing library was used to handle our View/Controller components. The View includes the frame and the various panels that are drawn upon it, including any buttons that each panel may carry.

A 'FrameSwitcher' controller is used to navigate between the panels, and a 'ModelController' controller to send commands to the Model itself from the Player.

The 'MouseController' is a listener that takes clicks from the user on the Game board and converts them into usable board coordinates for use in the ModelController.

All of this is launched from the MainFrameLauncher, which generates and instantiates the relevant controllers, initializes the panel and runs the GUI components.

We hope that you enjoy our end result as much as we do.

-James, Dylan, Zsanett, Daniel