Joshua Latvatalo

Huy Nguyen

JuAune Burgess

Design Document: Reversi Online Game

**Purpose:**

For this project, our main goal is to create a fully functioning online Reversi game. In this case fully functioning will entail a server that will allow a player go up against a low, medium, or hard level AI as well as let any variation of the aforementioned AIs compete against each other. The game itself is an 8x8 uncheckered board and initially the center four spaces will have to white and two black discs, each diagonal from their respective match. Each player will take turns making legal moves and in Reversi a legal move is defined by making a straight (horizontal, vertical, or diagonal) line between one disc of your color and the intended move space with any number of the opponent’s discs in between. When a move is made all of the opponent’s discs will be turned over and will take on the color of the other player’s discs. If a player cannot make a valid move their turn is forfeited. The game will end either when the entire grid is filled or when neither player can make a valid move. At this point the player with the most discs of their color on the board wins.

For client based interactions our client will be provided a GUI where they can request a connection to our server. In addition to this they will be provided options as far as the difficulty they would like to play and after that what move they would like to make. The board will be updated after each move is made, inform the player of how many discs each player currently has on the board, and eventually inform the client of a win or loss.

CLIENT

(Human or AI)

**High level Entities:**

Firstly, the client mentioned here is not actually coded into this project but is actually how we are defining the input that the GUI will be handling to properly create the requested game type and moves.

GUI

Next, the GUI is what the client interacts with for creating a connection to the server, sending moves, and updating the game board.

The server connects to the client and hosts the game which relies on the AI Engine and the Game Mechanics to send moves and information to and from the client.

The AI Engine is a class that, according to the difficulty level selected, will determine what moves will be made by the computer against its opponent.

Lastly, the Game Mechanics section is what holds things like the game board, legal moves, updating the board, etc. This is called upon by the AI Engine and the server when either sends a move.

**Low Level Designs:**

Starting from the bottom of the design, this section will describe the entities of the project in more depth.

Game Mechanics:

The game mechanics have one usage, and it is central to the whole project. The game mecahnics run basic functionality of Reversi. This is where you will find the game board, as well as functions covering things like making moves to checking if the game is finished and finding the winner. This part of the project will not directly interface with clients or external servers, but will be utilized by them through our server. The client will issue a command, which will come through the socket on our server, be interpreted, and then handed to the game mechanics to make the users commands a reality.

This entity will have a few internal variables necessary to implement the game play, the most important being the game board. The game board we will be implementing will be 8x8 matrix, that holds string variables, such as “B”, “W”, or “E”. The spaces on the board will be filled with these variables to signify what is in that space at the moment, “B” standing for black, “W” standing for white, and “E” standing for empty. The next internal variables that will be important are two vectors that store previous states of the game board. These vectors will be used to implement that undo/redo commands by simply storing the previous board states, and then overwriting the current one with the previous if a redo is called. The last variable will be a simple counter to keep track of the number of undo/redo’s used.

Some examples of functions found in this entity are:

makeMove()- will actually take the coordinates for the move being made, and adjust the game board accordingly.

checkMove()- will take the coordinates for a move being made, and check to see if it is a legal move.

findPossibleMoves()- will compile a list of possible moves so that checkMove() can use it.

Undo()- will overwrite the current game board with one stored in the vectors mentioned earlier.

Redo()- same as undo, just implemented a little differently.

checkEndGameCondition()- will find out if there are any more moves on the board to be made, and if not, end the game.

AI Engine:

The main purpose of the AI engine is to create a artificial intelligence for a user to play a game of Reversi against. This AI will have three difficulties, easy, medium, and hard, that will choose its moves based on differences in the algorithm. That algorithm will be a MIN-MAX function, and the differences in difficulty can be made through the difference in the limited depth of the function.

There is really one main function in this section, ChooseMove(). This function will take a list of possible moves, and use predetermined algorithms to choose the computers next move. It will then send the function to the server, and notify the client.

Server:

The server is the central point for the project. It's purpose is to connect the game mechanics, the AI engine, and the GUI. It accomplishes this task through two major functionalities, creating a socket and handling the user input.

The socket will be coded into this part. It will be set up to listen to a certain port, waiting for a connection from a user to come through the GUI. After the connection is set through the socket, the user can begin sending messages through, as well as allowing the server to send information to the user also.

The other main purpose of the server is to handle user input through that socket. Once the socket is established, it will begin handling the communications between the server and client. When the user enters a command, it will take that input, and then choose which command it is and use the proper functionality to handle it.

Functions in the section:

Connect()- the main socket function, it awaits a connection from a client and sets up the socket.

HandleMove()- the function used if the user has input a move, with call on game mechanics functions.

HandleUndo/Redo()- function for handling undo/redo

HandleDifficulty()- used to set the difficulty

HandleExit() - exits game

HandleDisplay()- handles display

GUI:

The GUI is the only part of this project that with interface directly with the user. It's main purpose is to provide the user with a nice interface to play the game from, instead of the command prompt being used for initial testing. The board will be fully displayed, with each square being a button, and will update after each move. It will also have a display for the current score, buttons for undo and redo, and a display for the number of undo/redo 's left. The other functionality behind the GUI is that is handles the client connections, rather than the user having to use telnet.

This GUI will be created using a c++ graphics library, possibly FLTK.

**Benefits, Assumptions, and Risks:**

Benefits

* Designed in this way, it keeps the client from having direct access to the game mechanics.
* By using the GUI, it makes a much better interface for the user, without the chance of miss typed inputs.
* By setting up everything behind the server, it allows for Human-AI games as well as AI-AI between server and server.
* By using a MIN-MAX function with a variable limited depth, it allows for the creation of varying difficulties without different algorithms.

Assumptions

* Users wont connect to our open port for nefarious means

Risks