|  |
| --- |
|  |

|  |
| --- |
| **Reversi Game Design Document** |
| Project 2 for CSCE 315 |
|  |
| This Reversi console game will allow users to login via telnet to a game server and play unlimited reversi games with a friend or against an AI. |
|  |
| **Laramie Goode, Chris Voss, Curtis Franks** |
| **6/15/2012** |
|  |

# 

# Purpose of Project

This project is an implementation of the game Reversi. The game is a strategic board game played with two players. Each player takes turns trying to capture the other player's pieces by forming a straight line from a starting piece of their color to an ending piece of their color. If this starting piece and ending piece form a line that contains the opposing player’s pieces, those opposing pieces are converted to the same color as the starting and ending piece of the line. The game ends when all the spots on the board are filled or when there are no more moves to be made. The only valid moves are one where a piece placed is adjacent to another piece and it forms a straight line either vertically, horizontally, or diagonally. The player with the most pieces by the end of the game is declared the winner.

With this project we will create a UNIX console Reversi game mechanism that allows for two people to play against each other or to utilize a player2 AI that will run off of the game server. The user will connect to the game server via an ip address and port and have command line access to begin a game.

# High Level Entities

The project is divided into three parts. The first part will be the game mechanics and will consist of two players taking turns making moves on the same machine. The next part will implement an AI so one player plays against the computer. The third part will implement the core game with the AI into a server program. A player will connect to the server via telnet and issue commands to the server which the server will handle appropriately.

## Simple Game Mechanics

The game will draw the board to the console with ASCII to represent the board and pieces. The game will have a board class which will have a matrix (array of arrays) of a class called Position. Position will have an enum type to represent whether or not the spot is empty, filled with a white piece, or with a black piece. Position will also hold the location of itself. A function will be used to determine whether or not a move is a valid move for the player. If a player is not able to issue a move, then the game will alert the current player and wait for a return character before it becomes the next player’s move. If neither player can make a move are all spots of the board are filled, then the game ends and shows the scores of each player.

We were fortunate to have many sources of reference including Wikipedia and a programmer named Richel Bilderbeek. Much of our code is based on a reverse game that Richel provides under the GNU General Public License program. We added undo, redo functionality and completely altered the menu system and coordinate system for the game board to match the parameters of the assignment. We additionally color formatted the console api and added additional available menu options.

## Artificial Intelligence

The game will have three levels of difficulty when playing against the AI. The Easy level of difficulty will have the AI pick random spots to place its piece on the board. The Medium level of difficulty will have the AI look ahead one move to determine the best move for the future. The Hard difficulty will have the AI look ahead two moves to determine the best move for the future. To achieve this, a MinMax search and Alpha-Beta search will be implemented. These searches will determine the possible move outcomes. The AI will choose the best move returned by the searches.

## Implementing into a Server

The game mechanics along with the AI will be put into a server along with the game loop. The server will have a simple parser to check for commands by the player. To connect to the server, the player will use Telnet with the hostname and port number as the arguments. Once connected, the player will receive messages from the server. To achieve this, the server will create a socket for the client. Once a client has connected to the server the server will loop receiving commands. When the game has started, the game loop will begin and not exit until another command such as exit/quit is issued or the game reaches an end state.

# Low Level Entities

## Game

The game is consisted of loops that receive input from the user. Once enough information is gathered, the game can start. The game’s data structures are a Reversi struct and an enum called Square. Square has the values empty, player1, and player2. Square is used to represent the individual tiles on the Reversi board. The board is a vector of vectors of Squares (matrix of squares). The Reversi struct has two more vector of vector of Squares. One of them is used to keep track of the game’s history. The other is used to keep track of the Redos in the game. The Reversi struct also has a string that stores the difficulty of the AI. The Reversi struct has set and get functions that return or set the Square object at the specific coordinates passed in the function. The struct has several Boolean functions to help determine valid moves (IsValidMoveUp, IsValidMoveUpLeft, etc). These functions also have matching void functions that actually perform the move by setting the Square with the correct player’s piece.

The communication will be through simple ASCII text. The commands are pretty simple.

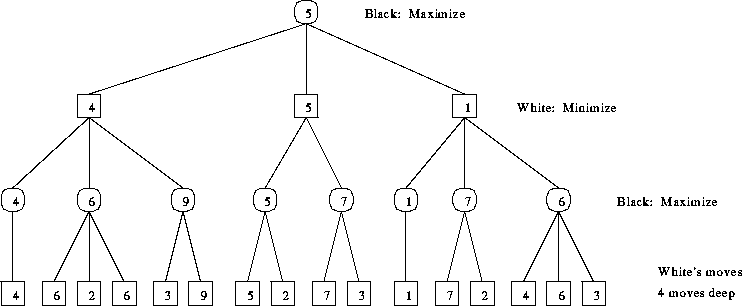
|  |  |
| --- | --- |
| *expr* | ::= *command* | *move* | *comment* |
| *command* | ::= EXIT | DISPLAY\_ON | DISPLAY\_OFF | EASY | MEDIUM | HARD  | BLACK | WHITE | UNDO | REDO | SHOW\_NEXT\_POS |
| *move* | ::= *column row* |
| *comment* | ::= ; \* |
| *identifier* | ::= *alpha* { ( *alpha* | *digit* ) } |
| *alpha* | ::= a | ... | z | A | ... | Z | \_ |
| *digit* | ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| *row* | ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| *column* | ::= a | b | c | d | e | f | g | h |

The commands are self-explanatory, except that SHOW\_NEXT\_POS will show the human player what are the available legal positions.

## Reversi (2)

## *Diagrams of core Reversi Game Design*Artificial Intelligence

## The AI will implement a Minimax search and Alpha-Beta search. The Minimax search finds the best move based upon which move will result in a board state that gives the human player the least amount of points or the greatest difference in score. The Alpha-beta search will decrease the number of nodes that are evaluated by the Minimax search. The Alpha-beta pruning algorithm stops evaluating a move when at least one possible move has been found that is worse than a previously examined move. This search returns the same move that the Minimax search algorithm would but gets rid of any branches that do not influence the final decision.



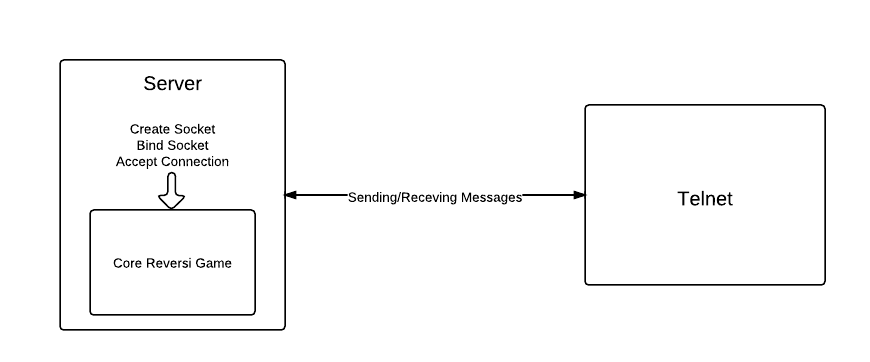
*An example of a* (Biermann)*Minimax tree*

## Server

The server will show these messages to the user:

|  |  |
| --- | --- |
| *ack* | ::= WELCOME | OK | ILLEGAL | BLACK | WHITE | *comment* |
| *comment* | ::= ; \* |

The server will create socket. After the socket is created, it will bind it to a hostname. An unused port will be used when binding the socket. The server will then wait for a client to connect. Once a client is connected, the server will send the ‘Welcome’ message to let the client know it has successfully connected. The server will then follow the same procedures as the core game mechanics. It will wait for commands and enter the game loop when the game is set up (after choice of color, difficulty, etc). The server will handle the commands and make sure the game receives input from the player. Telnet will be used to connect the player to the server and therefore, another client program is not needed to be developed. Telnet simply takes console input and sends it to the currently connected server.



*Server model*

# Works Cited

*Minimax and Alpha-Beta Template*http://www.pressibus.org/ataxx/autre/minimax/node2.html

*Reversi Console source code*http://www.richelbilderbeek.nl/GameReversiConsoleSource\_1\_0.htm

TAMU CSCE FacultyCSCE 315*TAMU*http://students.cse.tamu.edu/aalap/CSCE315\_s12/slides/Project2\_Design.pdf