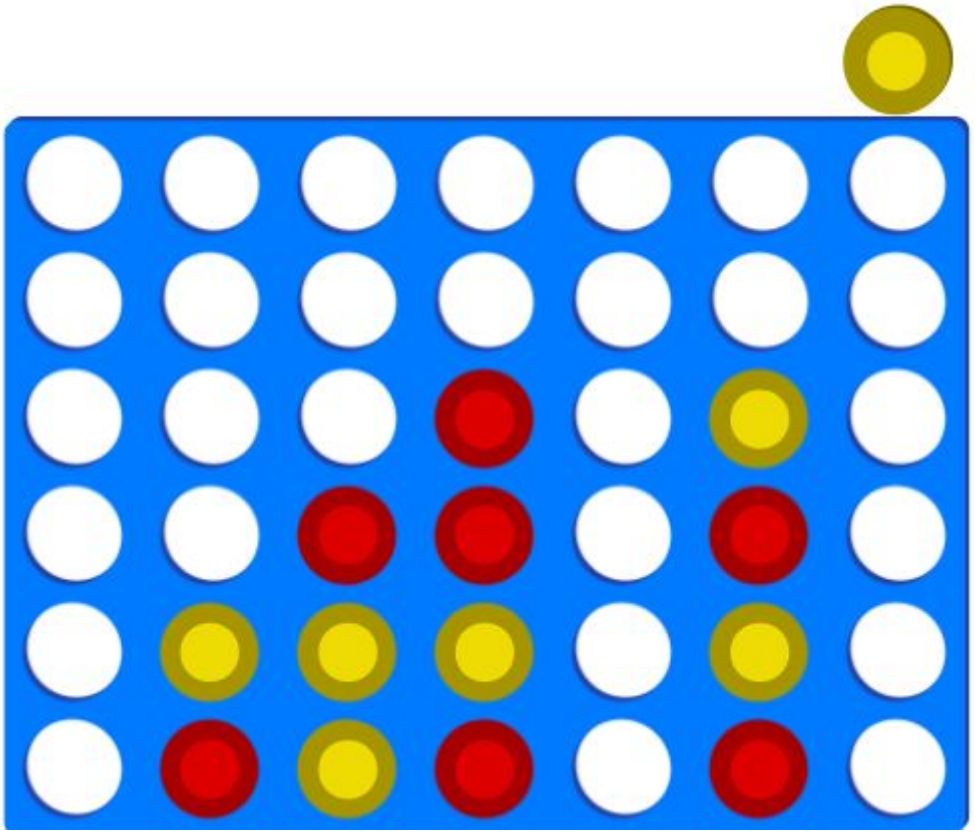
Technical Design Document
Connect Four Game



Designed by Daman Behl
Student ID(2091426)

TABLE OF CONTENTS

List of figures

Section 1: INTRODUCTION

1.1 Abstract

Connect four is a 2 player game, first commercially sold in 1974. Since the 1960's computer scientists have been obsessed with the idea of computers having Intelligence to beat human beings at board games such as chess, connect four, go etc. Even though Connect four is a solved game and doesn't require extensive research to build, it can be used as a stepping stone to understand game theory in computer science and inturn be used to create other games such as chess, go, mancala. This project also be used to implement algorithms which introduce us to algorithms used in Artificial Intelligence, one such algorithm has been used in game theory. This project consolidates concepts of any programming language that the user decides to build this game in. Selecting a random input for the computer to play defeats the purpose of this game and hence an algorithm has been used in this project which will be explained in the following segments.

1.2 Background

Connect Four is a solved game, a solved game is a game wherein the outcome of a game can be deduced from the position of the board, in simple terms, the outcome of the game can be predicted from the position of the board at any given time in the progression of the game. In this code implementation of Connect Four, Minimax algorithm or rather the alpha beta pruning variation of minimax algorithm has been used. Minimax algorithm is an algorithm used in game theory wherein given the assumption that both the players play perfectly, the algorithm maximizes the minimum gain. Minimax is a decision based approach for game theory which hits all the right notes for computerizing the play of a game, afterall every board game played is just a succession of decisions which the two players make as the game unfolds move after move. Connect Four is a perfect information game wherein there are no hidden rules of undefined variables of factors which might alter the course of the game, in simple term, both the players have the information about all the moves that the opposing player has made..The game has been previously solved by Victor Allis and James Dow Allis with a knowledge-based approach came up with nine strategies to solve the game.
Connect four can be solved using brute-force methods such as an 8-ply database. There are artificial intelligence algorithms that have been able to solve this problem such as minimax and negamax, with optimizations such as transposition tables which is a cache of previously achieved states in the game or in a decision based model, pruning of traversal of certain branches of a decision tree. Since this is a solved game, it has been deduced that given perfect play is in place for the first player, the first player will always win and if the first player puts the disc in adjacent

columns to the center column, the game ends up in a draw, thus the first player can almost always force a win provided perfect play is in effect.

There are various variations of the Connect four game. The variation used in this project is the 6 rows and 7 columns variation of the game other notable variations of the game are:

- PopOut: In this variation the game is similar to the traditional 6X7 connect four game, the only difference being that a player in his turn can remove a disc from the bottom of the board provided the disc the player wants to take out is his own. When the player takes out this disc, all the discs above will move one spot down.
- Pop 10: first the entire board is filled row by row until the whole board is filled out, the game progresses by both players taking turns removing their own discs from the bottom of the board. Now if a disc taken out was a part of a continuation of four discs, the player sets the disc aside and it is then out of play, when a player collects 10 such discs, that player is declared winner.
- 5-in-a-Row: The game is played on a 6 high and 9 wide grid. Two additional board columns, pre-populated pieces in alternating pattern, to the left and right sides of their standard 6 by 7 board and then as the name suggests, the player has to put together five pieces in a row.

1.3 Objectives

Below listed are the objectives of this project:

- To create a GUI(either a text based or a UX based) User indicative information display screen, that indicates the game board at all times, interactively ask the user for input.
- Show the user values input by them, and the turn played by the computer and to display it graphically on the screen so the user can make their future moves.
- Calculate the status of the game and the position of the board after each move is played.
- Make the computer smart enough to win a game against the user.
- Use core concepts of Java to achieve efficient clean code, using functions, smartly store data and to develop an algorithm that can beat the human player playing the game.

Section 2: Game Design

2.1 System Requirements:

- Installation of JRE and JDK on a supported operating system, eg any debian based or windows based operating system.
- Appending path of Java development kit in the environment variables of that particular operating system or if using debian based OS, adding the path to the executable in the ~./bashrc.
- Installation of an IDE with a maven or a standalone maven installation.
- User Input: Input of the column in which the user intends to put the disc in.

2.2 Other functional Requirements

- Intelligent play by the machine against the human opponent.
- Indication of coordinates where the disc was placed by the AI
- Error handling and seamless play
- Appropriate error messages after an invalid input has been done by the user.
- Recursive gameplay until the user does not want to play the game anymore.
- Instructions for the game unfolded to the user before the game initiates.
- Efficient use of logic, avoiding use of Brute-force approach and implementing a smart algorithm that can be configured for varying levels of difficulty on changing a single static variable.
- Clean output of the game board so that the user can clearly view the game.
- Accurate labeling of the board with the respective column number.

Section 3: Detailed Description

3.1 Working of the Game

- Initially the program must initialize the game board and print the game board. Then the program has to alert the user to initiate the game by asking the user to input the column in which the user wants to input the disc.
- Upon printing and notifying the user to input the column number in which the user wants to enter the disc, the program first validates the input entered by the user.
    - The input is checked for negative integers and a warning is displayed to the user specifying the same.
    - The input is checked for string values.
    - The input is checked for floating point values as they are invalid.
- Upon ingesting valid input, the game progresses further progresses and now the computer makes the move, The AI places the disc in the column of it's choosing and then notifies the user of the exact position where the user has placed the disc so that the user can make an informed choice on their next move.

- The algorithm used for mimicking Artificial Intelligence in this game is the Minimax algorithm, specifically the alpha beta pruning variation of Minimax algorithm.
- The minimax algorithm is a decision based game theory algorithm which is very popular and effective in 2 player games where the opposite player is trying to win and is playing optimally. In minimax, the player moving is trying to maximize his gain or heuristic score, board position, gradually while the Min player is trying to minimize or put the Max player in the position where his/her score is minimum.
- Minimax is a recursive algorithm, it takes a depth variable which decides the difficulty of the game in question, the depth variable is the number of steps that the computer traverses for all the possible columns available to it, this recursion stack memory decreases as the game further progresses and possible positions are put out of play.
- The input validator step is only required for the user input and not the column selected by the AI since that check is already integrated in the insertion process for the computer.
- User input is verified after it is taken from the user and then a separate function is called in order to drop a disc into the board, this step is common for both the computer and the player itself in a bid to reuse code, concepts of OOPS have been used where the entire code is broken down to individual functional units. This helps in reusing the same code for computer as well as the player and also testing the code.
- After the piece is dropped by the player, the board is sent to a win calculator function along with the piece which was put in recently to check if the player won.
- If the player wins, a congratulations message is displayed onto the screen, if the game ends in a draw a notification is shown accordingly. After the terminal position is achieved in the board, the user is asked whether he wants to play another game or not.
-

3.2 List of Functions

Following are the list of functions used:

- **public static void display(char[][] grid)**: This is a double looped function to print the grid onto the screen at any point during the progression of the game.
- **initializeGrid(char[][] grid)**: This function is used once when a new game starts, the grid is initialized and loaded with whitespaces.
- **public static void playTurn() :** This function initiates the gameplay between the computer and the player.The global class variables are populated as the game progresses, a controlled while loop is executed in this function till a player wins or the game ends up in a draw.
- **public static boolean validate(int column, char[][] grid):** This function validates the input entered by the user, ie. it checks if the column entered is

within the allowed bounds of the grid array and that no negative, floating point or string value is entered, also checks for stacked up columns and that the input does not correspond to a column that is already filled up. In case false is returned by the function.

- **public static boolean isWinner(char player, char[][] grid) :**check for winner by ingesting the grid and the player that made the last move in the game sequence.
- **public static Long[] minimax(char board[][], int depth, long alpha, long beta, boolean maximizingPlayer):** Minimax recursive algorithm to return the best possible column to maximize the minimum gain every time the computer makes a move, also returns the heuristic value of the board position.
- **public static int[] getValidLocation(char board[][]):** returns a list of columns that are not filled with discs. Used both in minimax to get the list of possible branches where a valid move can be made. In isTerminalNode used to check if all columns are full to avoid traversing the game further and break out.
- **public static long positionScore(char board[][], char piece):** returns the cumulative score of a game board at any point by iteratively adding the score of individual rows, columns and diagonals.
- **public static int windowScore(char window[], char piece):** returns the score of an individual widow formation taken separately from the board to the main function positionScore.
- **public static int countOccurences(char arr[], char find):** counts the occurrences of a particular piece in a windows array of 4 elements.
- **public static char[] getDiagonal(char[][] matrix, int row, int col):** To retrieve diagonals from a board grid.
- **public static int whereDoesThePuckStop(char board[][], int column):** This function returns the column where the last disc stopped to accurately depict the position of the board to the player playing so that they can make an informed choice.

3.3 Flowchart-Game

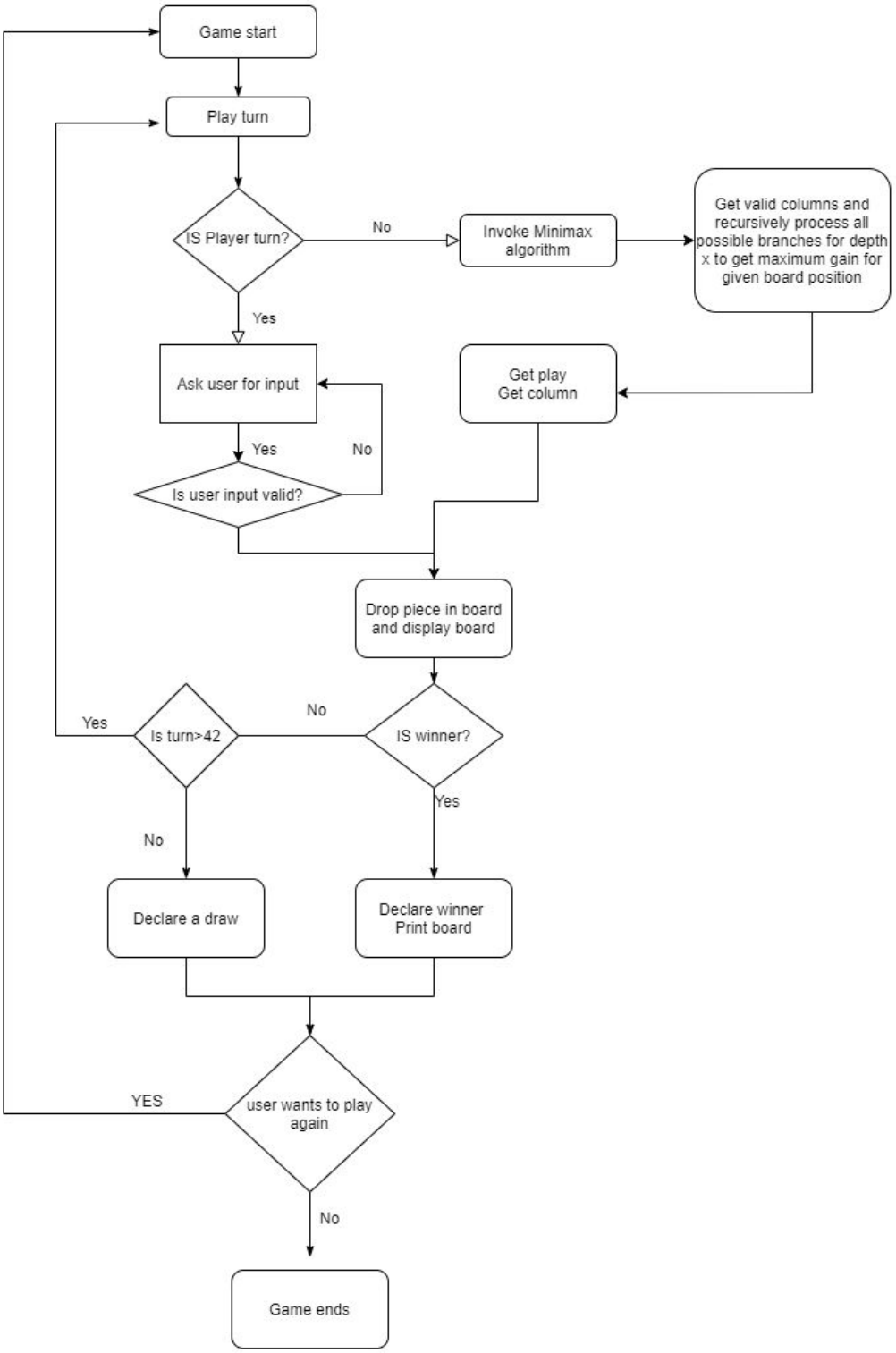


Fig 3.1 Game Flowchart

Section 4: MiniMax Algorithm

- Introduction: MiniMax is a decision based backtracking algorithm used in game theory to find the best possible move for a player in a two person game, given the player in the opposition plays optimally.
- Usage: Minimax is widely used in 2 player- turn based games such as chess, tic tac toe, mancala, go
- Goal: The basic essence of this algorithm is that there are two players, namely the Maximizer and the minimizer. The maximizer tries to maximize the score(or the overall heuristic value of the board with respect to the rules of the game and the scoring criteria).
- Variation: the variation used in this game is the **alpha beta pruning** variation of the algorithm, it is an optimization technique in which two extra variables are passed along to successive recursive runs of the game run for the computer's turn.
    - Alpha: the best value that the maximizing player can guarantee for the current level or above.
    - Beta: the best value that the minimizing player can guarantee at that level or levels above it.
    - Condition: if alpha>= beta, prune the subtree since there is a better option for the opposing player to choose from.

4.1 Pseudo code

```
function minimax(node, depth, isMaximizingPlayer, alpha, beta):

    if node is a leaf node :
        return value of the node

    if isMaximizingPlayer :
        bestVal = -INFINITY
        for each child node :
            value = minimax(node, depth+1, false, alpha, beta)
            bestVal = max( bestVal, value)
            alpha = max( alpha, bestVal)
            if beta <= alpha:
                break
        return bestVal

    else :
        bestVal = +INFINITY
        for each child node :
            value = minimax(node, depth+1, true, alpha, beta)
            bestVal = min( bestVal, value)
            beta = min( beta, bestVal)
            if beta <= alpha:
                break
        return bestVal
```

Source-(geeksforgeeks, Akshay L Aradhya) Minimax Algorithm in Game Theory | Set 4 (Alpha-Beta Pruning), code2Image converter - Carbon

4.2 Flowchart -Algorithm

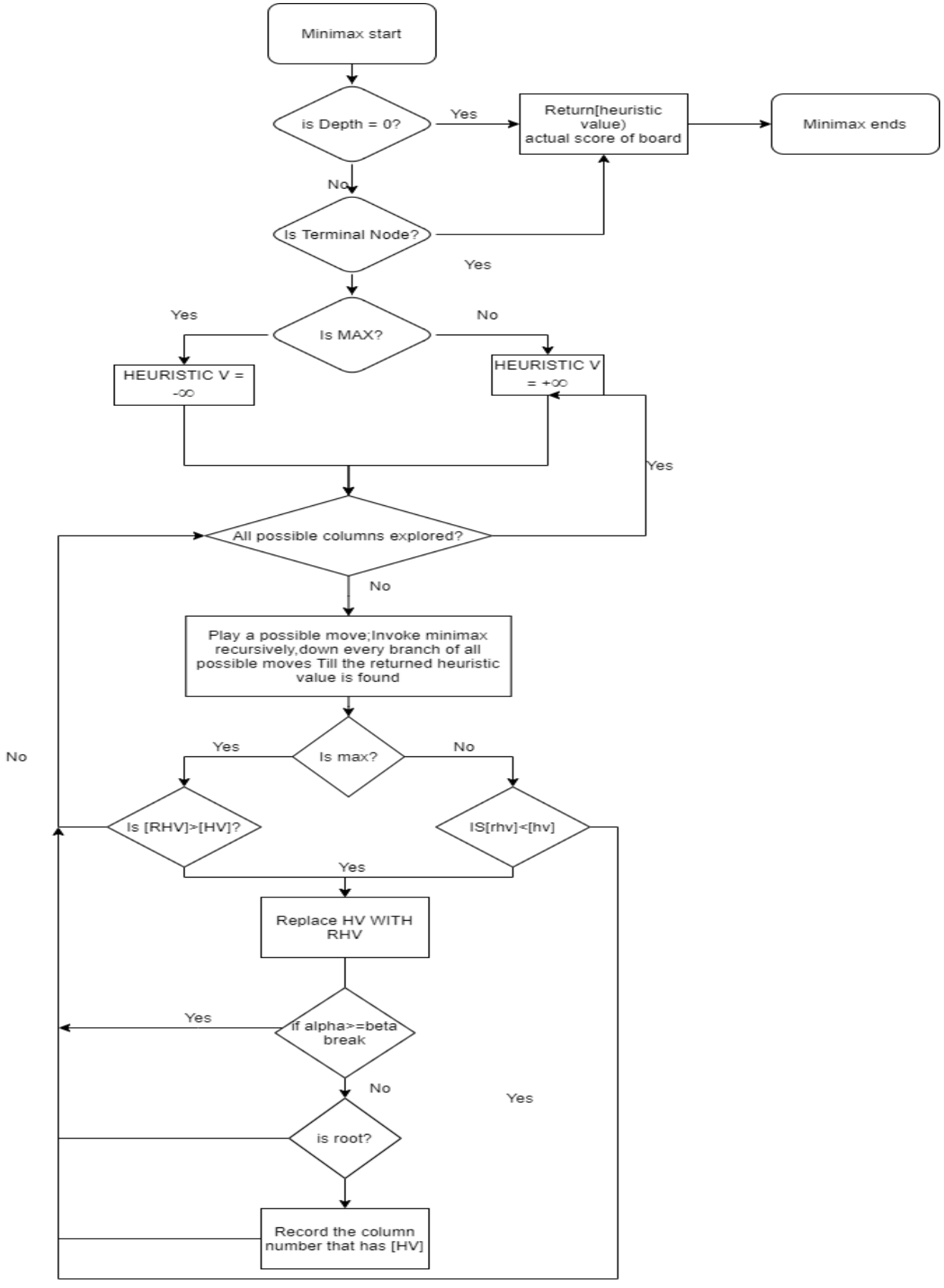


Fig 4.1 Minimax flowchart

Section 5: Game Progression

- Step 1: Initial state of the board, an empty grid is initialized

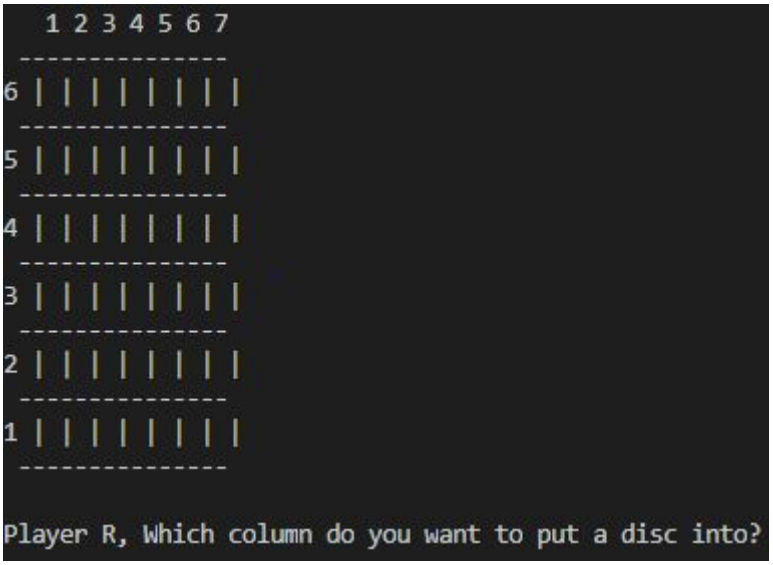


Fig 5.1 Initial Board state

- Step 2: The user is asked for input, after the input is validated and processed, the grid is displayed on the screen and then the AI makes a move and displays the specific move it made.

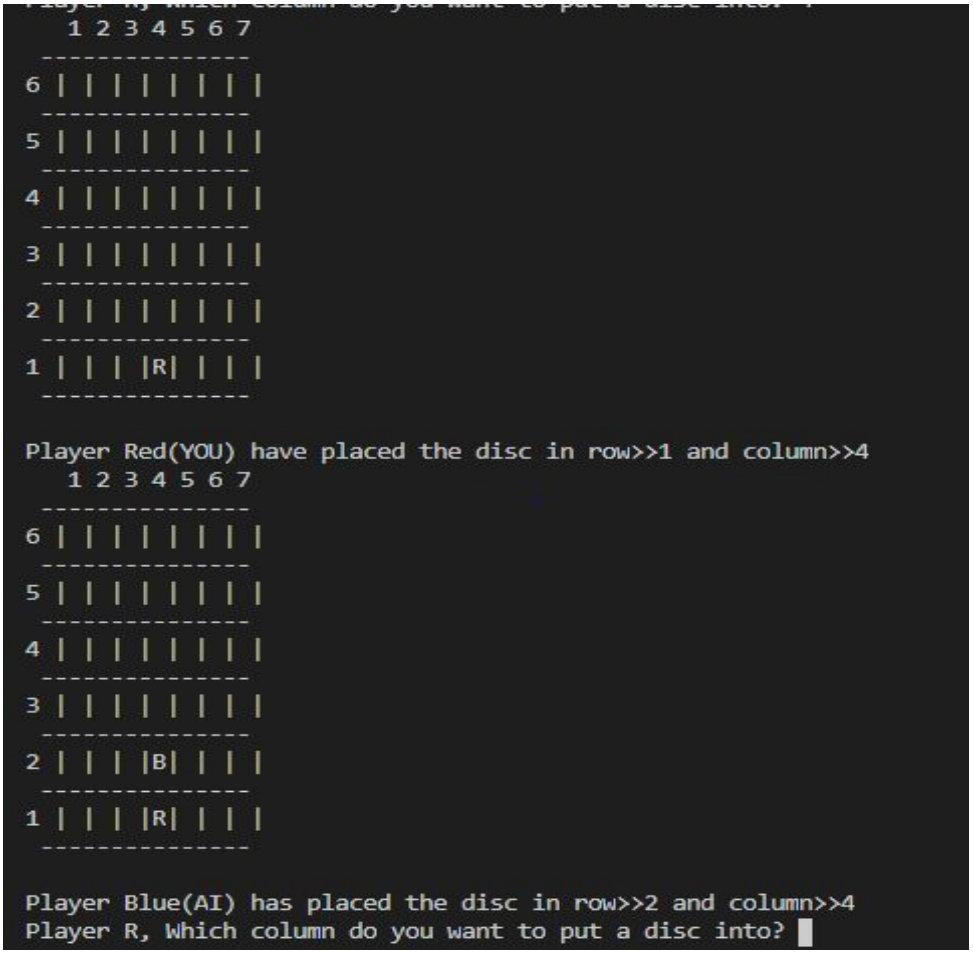


Fig 5.2 User Input

- Step 3: Upon getting four consecutive discs in a row, a player is declared winner.

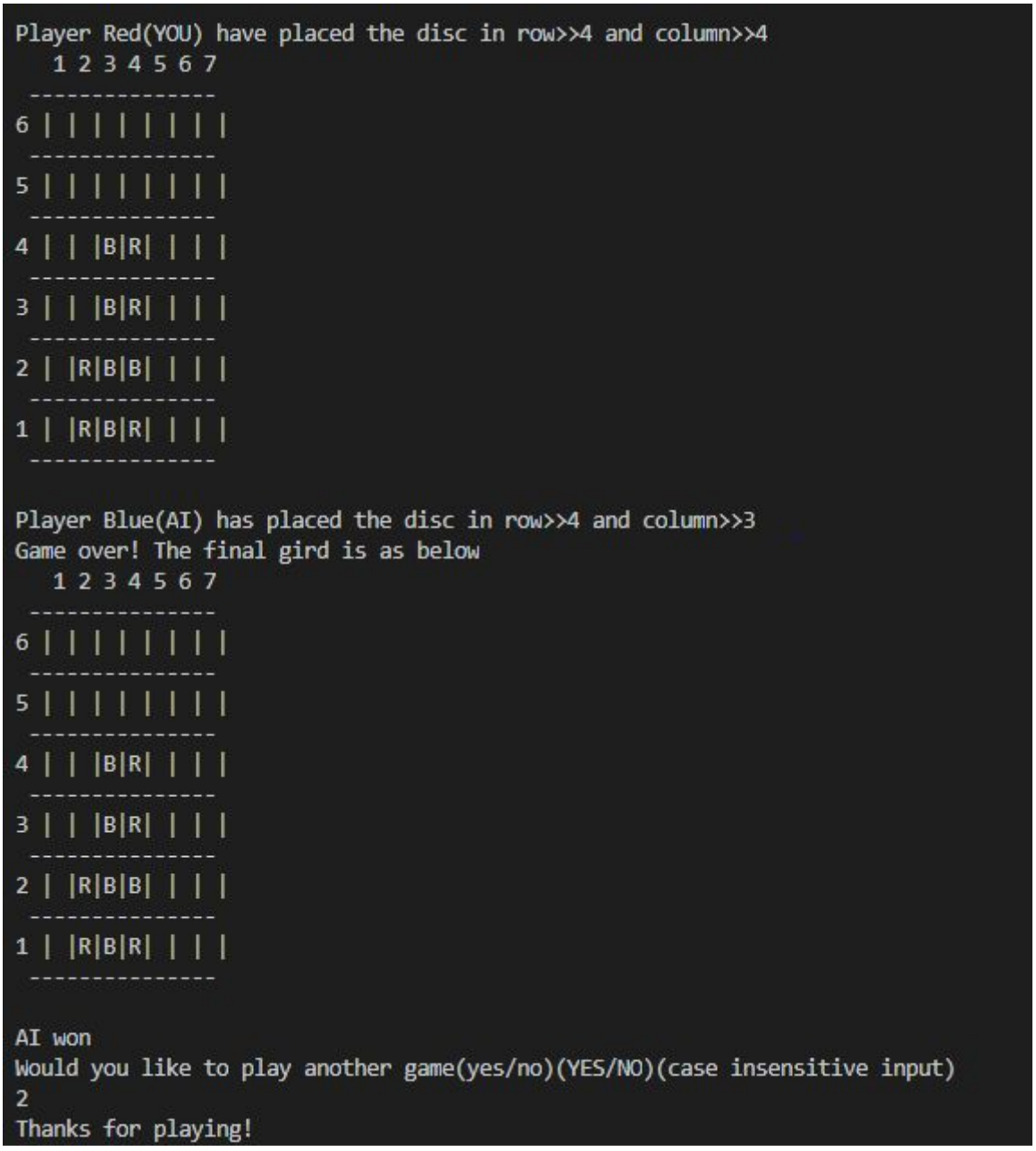


Fig 5.3 Win declaration

Section 6: Error Handling

- Column exceeds the number of columns on board: If the column value is greater than 8, a message to enter a valid input is displayed to the user

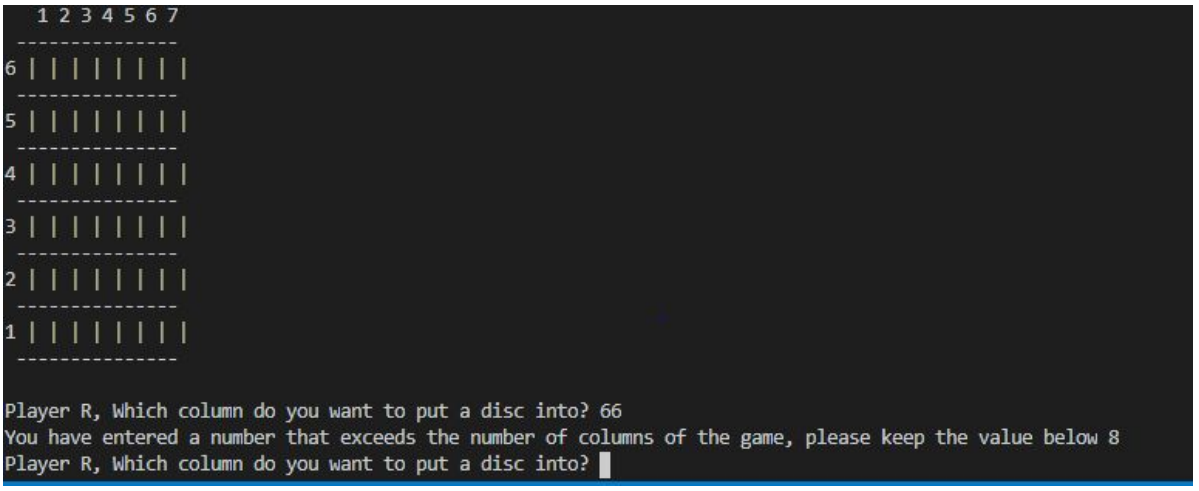


Fig 6.1 Number exceeds array bounds

- Column is negative: If a column is negative the validate method returns false and a message is displayed to a user to enter a new input.

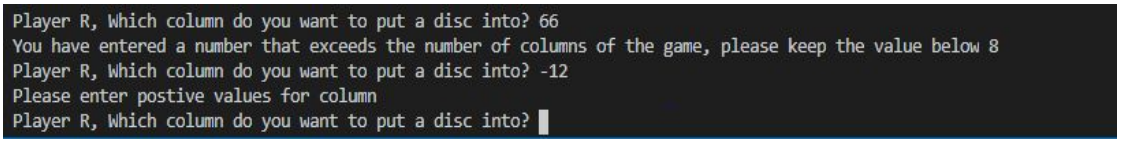


Fig 6.2 Negative number entered

- Column entered is string: if the user entered a string, an appropriate message is shown

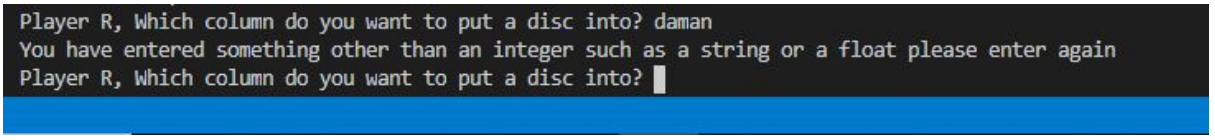


Fig 6.3 Invalid String input

- Column is already stacked:If a column is already full, a message is shown on screen depicting the same.

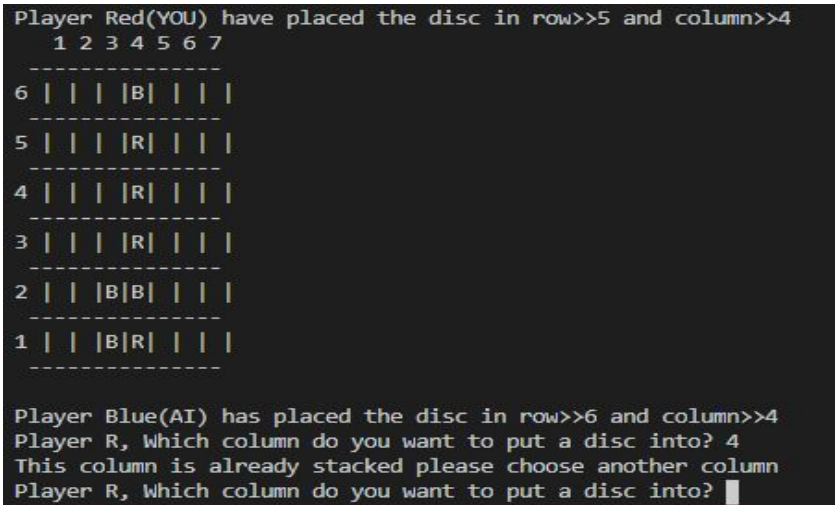


Fig 6.4 Column already stacked

Section 7: Future Improvements

Future improvements in this project could be among the following:

- Graphical UI: An improvement in this project would be to introduce a UI/UX based interface.
- Undo moves: another improvement would be the ability to go back in time and change some moves.
- Adjustable difficulty: an easy improvement could be adjusting the depth to which minimax traverses board positions, the depth could be a user changeable value.
- Storing past games is another improvement, game moves can be stored in a stack and that stack be stored in a database in text format and then parsed back to array and replayed.

Conclusion

Connect four is the perfect 2 player game for beginners who are exploring game theory and if algorithms such as negamax and minimax are implemented, a perfect beginners introduction to algorithms for backtracking and very basic artificial intelligence. Real world scenarios include games, and decision based scenarios including 2 opposing agents trying to bring each other's positional score down. This game helps understand game theory and basic Artificial Intelligence.