School of Computer and Information Sciences (SCIS)

Project Report For

Checkers

Submitted By:
Biplaba Samantaray(18MCMC05)
Pallavi Kumari(18MCMC33)
MCA IV

Submitted To
Dr. PSVS Sai Prasad
Assistant Professor
School of Computer and Information Sciences (SCIS)
University of Hyderabad

Acknowledgement

A better co-ordination and co-operation reflect the project success but a major role play behind in the background apart from the hard work is a proper guidance.

In performing our assignment, we had to take the help and guideline of some respected persons, who deserve our greatest gratitude. The completion of this assignment gives us much Satisfaction and Pleasure. We would like to show our gratitude **Prof. Sai Prasad, SCIS, University of Hyderabad** for giving us a good guideline for assignment throughout numerous consultations.

We would also like to expand our deepest gratitude to all those who have directly and indirectly guided us in writing this assignment.

In addition, a thank to all the References and Books we have studied and read throughout for completing this assignment. We have learnt a new concept for pygame development which are unaware of before this.

Implementation of Minimax algorithm and its integration with GUI was a tedious work, but finally leads to its successful completion because of all the contribution you have given to us. I would like to thank you all once again.

Table of content

GAME BASICS

1.a. Collected information about checkers

Checkers is played by two players. Each player begins the game with 12 coloured discs. (Typically, one set of pieces is black and the other red.) Each player places his or her pieces on the 12 dark squares closest to him or her. Black moves first. Players then alternate moves.

The board consists of 64 squares, alternating between 32 dark and 32 light squares. It is positioned so that each player has a light square on the right-side corner closest to him or her.

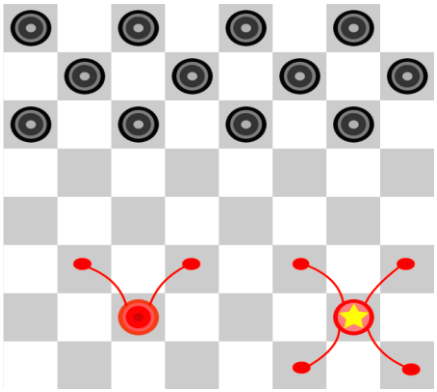
A player wins the game when the opponent cannot make a move. In most cases, this is because all of the opponent's pieces have been captured, but it could also be because all of his pieces are blocked in.

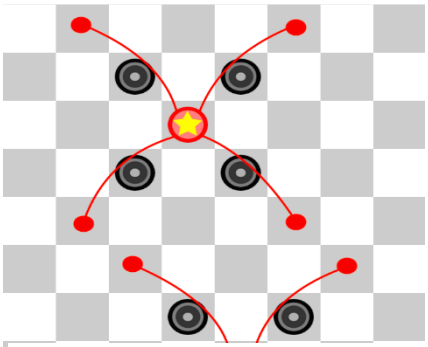
1.b. Rules of the Game

- Moves are allowed only on the dark squares, so pieces always move diagonally. Single pieces are always limited to forward moves (toward the opponent).
- A piece making a non-capturing move (not involving a jump) may move only one square.
- A piece making a capturing move (a jump) leaps over one of the opponent's pieces, landing in a straight diagonal line on the other side. Only one piece may be captured in a single jump; however, multiple jumps are allowed during a single turn.
- When a piece is captured, it is removed from the board.
- If a player is able to make a capture, there is no option; the jump must be made. If more than one capture is available, the player is free to choose whichever he or she prefers.
- When a piece reaches the furthest row from the player who controls that piece, it is crowned and becomes a king. One of the pieces which had been captured is placed on top of the king so that it is twice as high as a single piece.
- Kings are limited to moving diagonally but may move both forward and backward.

  (Remember that single pieces, i.e. non-kings, are always limited to forward moves.)

- Kings may combine jumps in several directions, forward and backward, on the same turn. Single pieces may shift direction diagonally during a multiple capture turn, but must always jump forward (toward the opponent).





Capturing move of pawn and king

1.c. Feature included in our project

- **Pawn could promote to king**
- **Single and Multiple jumps of both pawn and king**
- **Mandatory movement of both pawn and king if there any possibility to kill**

Yet to be implemented:

- **Some more work in GUI and Evaluation function**
- **A user should get a flexibility to choose whether he/she wants to kill the opponent or just skip for that case**

REQUIREMWNTT ANALYSIS

2.a. Software

Before the development itself, major decisions have to be made. Such as the programming language, integrated development environment, and the algorithms the developer needs for their project, in this case, algorithm represents the artificial intelligence player.

2.b. Language of Choice

After creating a simple implementation model and knowing the exact requirements for this project We needed to choose a programming language. Language with libraries that I use for storing all the possible moves for each round and easy implementation for Artificial intelligence algorithms. Also, libraries for easy to understand object code development and graphical user interface that looks good and is simple to create.

We were deciding between java and python. At this point, for the ease of implementation, we chose python for this project.

2.c. Integrated Development Environment

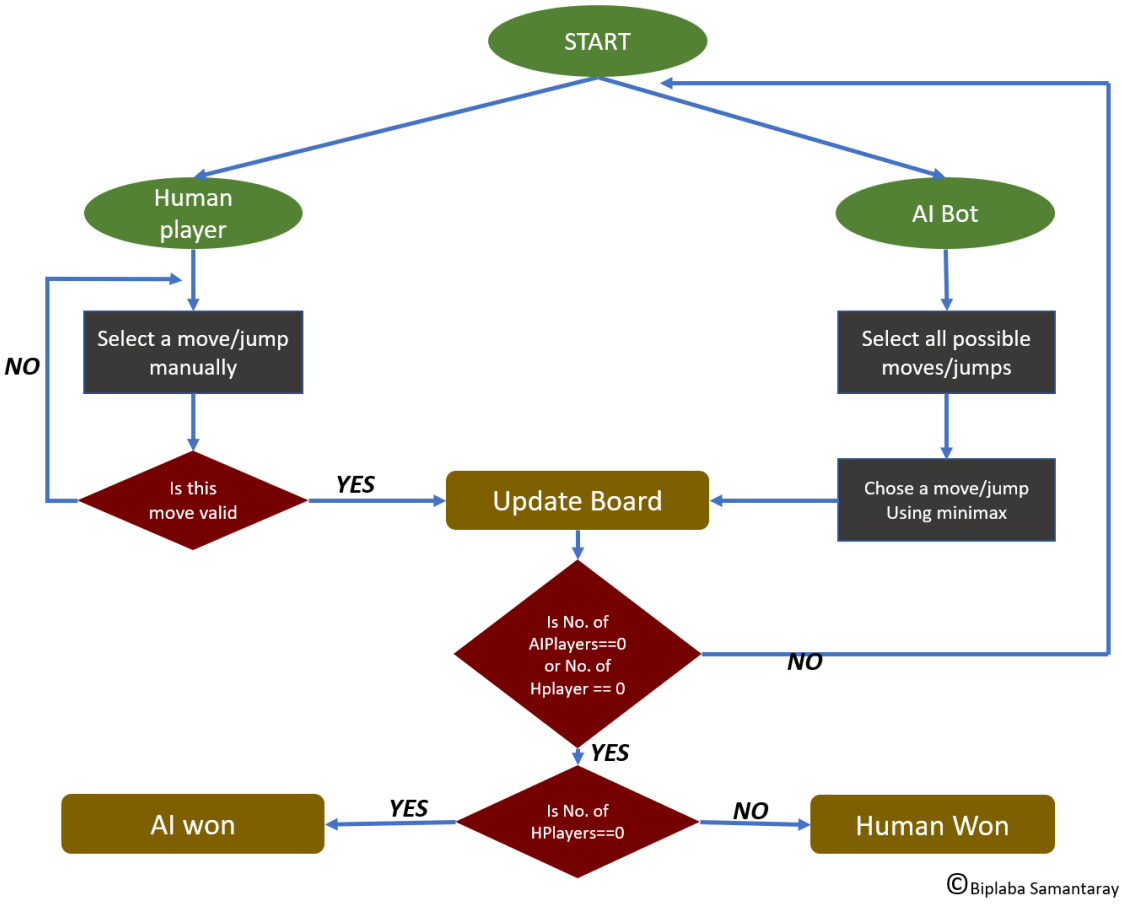
Before the development phase starts it is important to choose an IDE that is well designed and helpful. People usually start using some IDE and then they stick with it. We use **PyCharm**. PyCharm offers a better developing environment, easier debugging mode. Also, we are using some inbuilt library for GUI development and for mathematical calculations which is easily implementable in this IDE.

Language	Python 3.8
IDE	PyCharm
Libraries	All basic libraries of Python Pygame: for GUI
Other	We are using concept of OOPs for project development

ARCHETECTURE AND MODULES

3.a. Defining the architecture

After successfully analyzing the idea and selected the tools we proposed an architecture with which we both are comfortable to work with. The following figure represents our game loop and the strategy of our development.



3.b. Classes and functions

Initially we observed that we have to focus on three things that are:

- The GUI
- Defining the rules through functions
- Defining Minimax algorithm + Evaluation Function

A successful project reflects the better co-ordination and co-operation behind the scene. As instructed by our Instructor, Prof. Sai Prasad sir, we have decided to work in a co-operative

way for its successful completion, considering the time constraints into the mind as well.As of our ease of choice Pallavi decided to develop the whole GUI and I(Biplaba) decided to work on the minimax algorithm. As deciding the rules (What rules to be included or excluded) is one of the major tasks and could affect both of our development, we together work on that part.

As we have already mentioned we are using the concepts of object-oriented programming and python as our key tool so we split our whole project to several classes and for each class corresponding methods considering the function calls to be flexible enough and we could update our project further.

Classes and methods:

Classes used for GUI Implementation

- Checkerboard class
- Player class

Classes used for Minimax Algorithm

- AI class

All the description for the classes and their respective methods are briefly described in GUI implementation and AI implementation section.

We have also used some basic pygame function to display messages, draw shapes(polygon) in our game.

ENVIRONMENT SETUP

4.a. For windows:

- Install python3 from following link
  https://www.python.org/downloads/
  install and set your environment variable if needed
- To install the Pygame package open your Command prompt and run following command

```
pip install Pygame
```

- To run the game open command prompt on your same project directory and run following commands

```
python main.py
```

- From any of your python ide like PyCharm/Jupiter/visual studio you can directly run the project

4.b. For Linux:

- Install python3 from following link
  https://www.python.org/downloads/
- To install the Pygame package open your terminal and run following command

```
python3 -m pip install -U pygame --user
```

- To run the game open terminal on your same project directory and run following commands

```
python3 main.py
```

- From any of your python ide like PyCharm/Jupiter/visual studio you can directly run the project

If you are still finding installation problems kindly follow the instructions from the link or contact me on the details given in the last page

https://cs.hofstra.edu/docs/pages/guides/InstallingPygame.html

THE GUI IMPLEMENTATION

5.a. The board implementation:

With the ease of using pygame for Game Development, a board of size (8 x 8) is being created. For making it more interactive to the users, relevant information (player's turn, Player's score and Node explored by AI agent(bot)), I have shown it on the left side of the board.

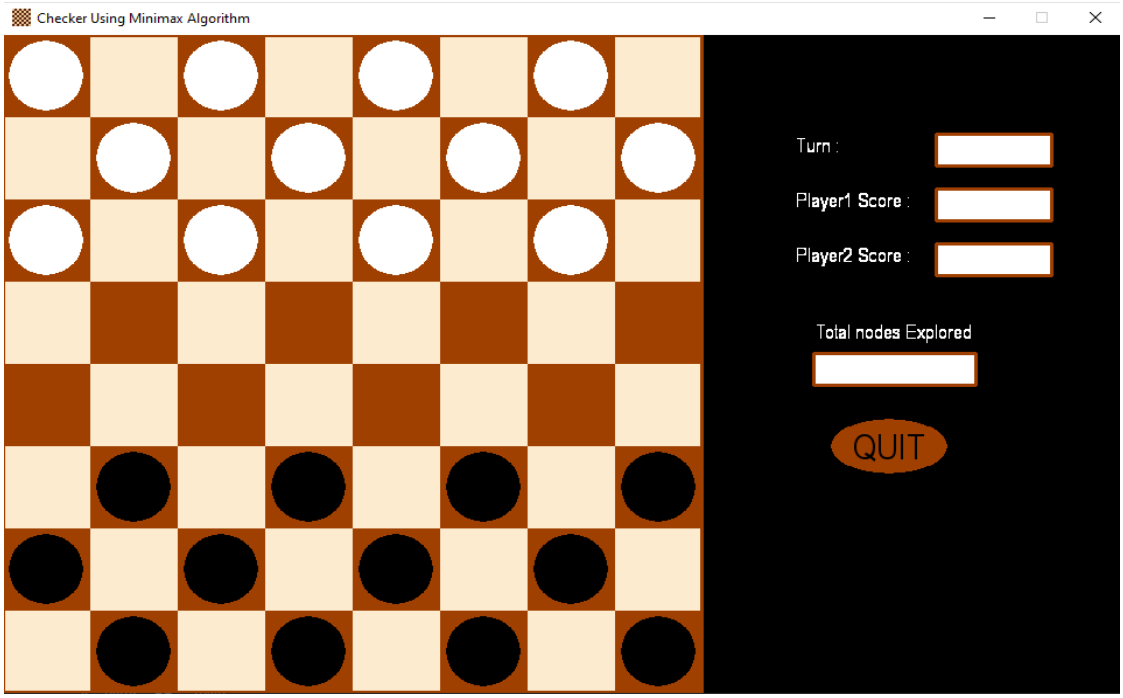


Fig: Initial Configuration of Checkers Board

Classes and Methods Used:

For better understanding and implementation, I have use python OOPs concept throughout game.

- Class Name: CheckerBoard

  Used for: Creating the board and also for updating the board

Module Used: `def __init__(self, surface, size, col1, col2):`

- All the board configuration initialization will be done using this constructor.
- Two colors are used to fill the alternative square of checker board.

Parameter used	Configuration
surface	Screen size (960 x 600)
size	Board size (600 x 600)
col1	WOODEN
col2	GREY

Module Used: `def draw (self, turn):`

- This method is used to draw the board based on initialization done by Checkerboard constructor.
- Based on the turn(player1/player2), it fills the alternative squares.
  - Player1 reside over the square filled with col1(WOODEN)
  - Player2 reside over the square filled with col2(Light GREY)

Module Used: `def update_board (self, ply1, ply2):`

- Based on the position of the player1(ply1) and player2(ply2), Using pyagame.display.flip() method, board is updated every time if there will be any movement over the board.

5.b. The pawn and king implementation:

This section is about how pawn and king are drawn on the board. For designing the pawn, I have chosen White to represent Bot player pawn and Black color to represent Human player pawn. The classes and methods used is described below in brief:

➢ Class name: Player
Used for: draw the corresponding player pieces over the board using pygame.draw.circle() and also rules for player are described in the class.

Method Used: `def __init__(self, surface, size, num, col):`

- All the initialization for player is done using this constructor.
- If num ==1 represents player1; if num ==2 represents player2

Parameter description:

Parameter used	Configuration
surface	Screen size (960 x 600)
size	Board size (600 x 600)
num	Player1/player2
col	Player color

Method Used: `def init_pos (self, n):`

- This method is used to define the initial position of the player.
- If n==1, player1 pawn will be placed on the square coloured with WOODEN
- If n==2, player2 pawn will be placed on the square color with

Method Used: `def draw(self):`

- Define the centre and radius of the pawn pieces and draw it over the square on board.

Method Used: `def promote_king(self, pos):`

- Pawn is promoted to king whenever the opponent player reaches to the last row of other side as already defined in the Defining Rule above.

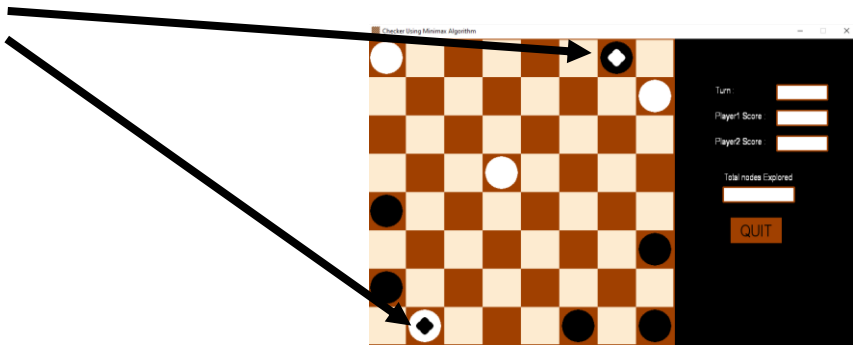


Fig: Arrow represents when the pawn promotes to kind

5.c. The Rule implementation:

A Game is considered working correctly if it follows the rules define. Here are the methods used for its efficient rule implementation:

Module used: `def select_piece(surface, player, selected, moveto, event):`

- This method is used to get the position of the player pawn selected.

Module used: `def move(self, selected, moveto, board):`

- This method is used to move the selected piece.
- Check for the valid move calling the methods described below.

Module used: `def check_forced_move(self, board):`

- Check for if there is any eating pawn an opponent can eat, it will be not selected to move further.

Module used: `def check_eating_move(self, selected, moveto, board):`

- As rule defines, if there will be possibility to eat opponent pawn, it will eat the pawn and then jump to valid position.

Module used: `def check_valid_move(self, selected, moveto, board):`

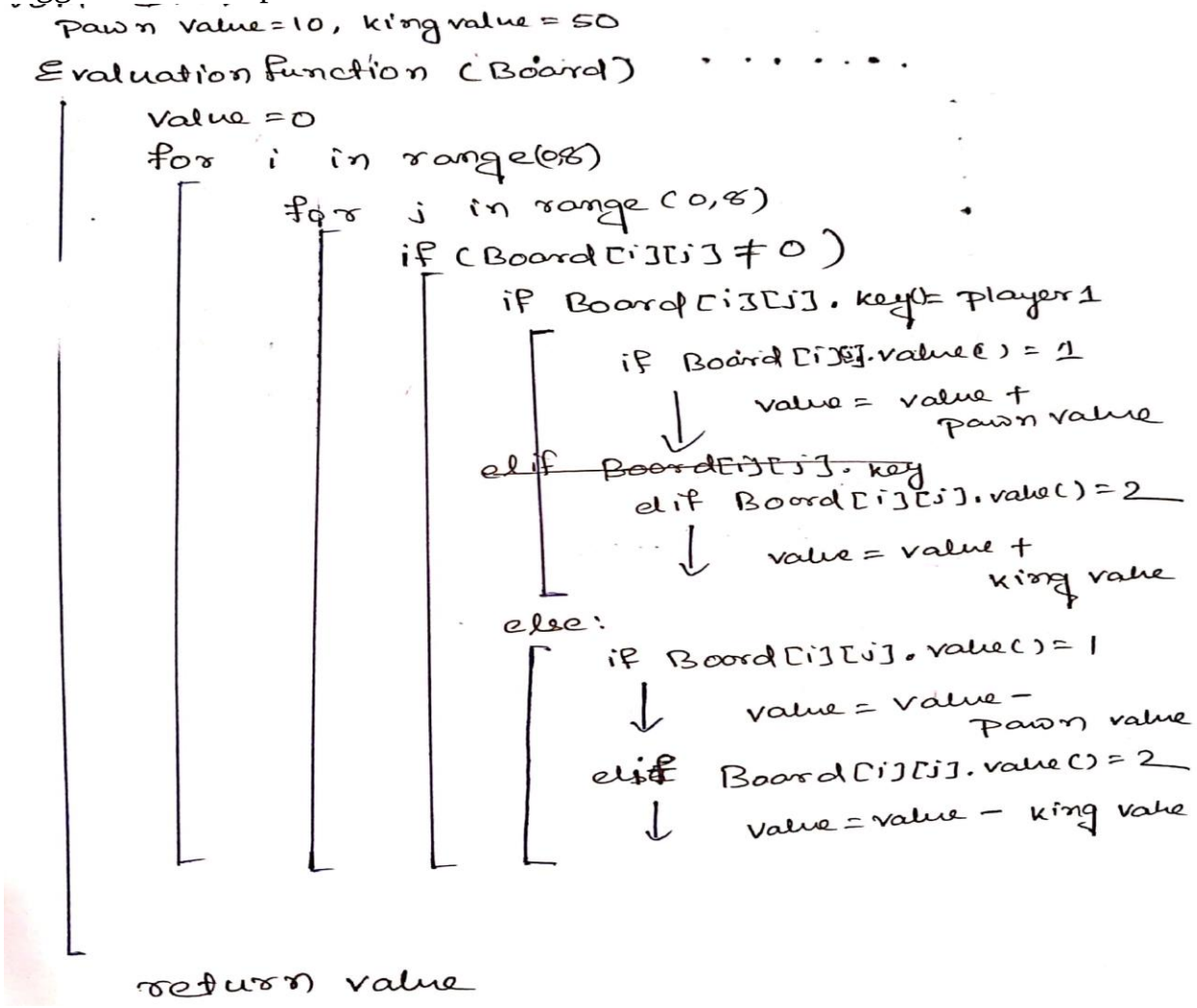
- Check for the boundaries of the board whether the move is in the coordinate of the board designed or not.

THE ARTIFICIAL INTELLIGENT BOT IMPLEMENTATION

5.a. Defining a suitable Evaluation function

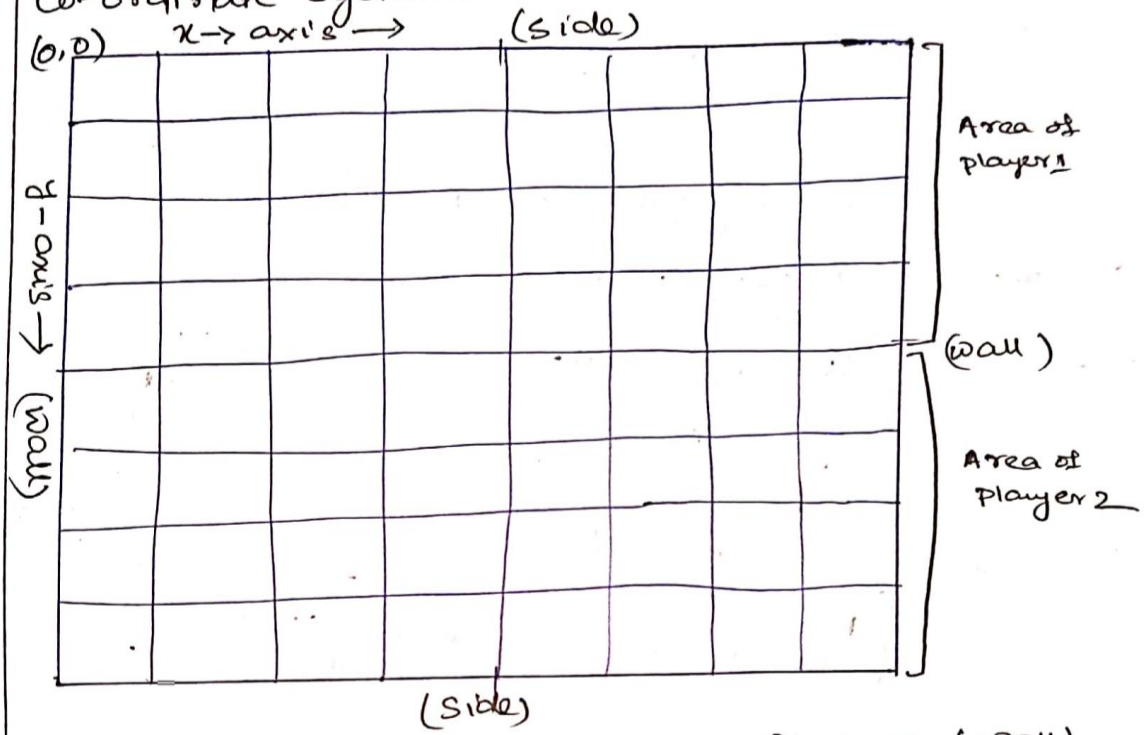
An **evaluation function**, also known as a **heuristic evaluation function** or **static evaluation function**, is a function used by game-playing computer programs to estimate the value or goodness of a position (usually at a leaf or terminal node) in a game tree (Definition from Wikipedia). The concept for evaluation function described in the class for the game Tic Tac Toe helps me to understand the concept. I go through different sources to understand the strategy used to develop this function for different board games like Tic Tac Toe, Gomoku, Connect4 etc.

Initially I created a very simple Evaluation function as given below from taking suggestions from prof. Sai Prasad sir.



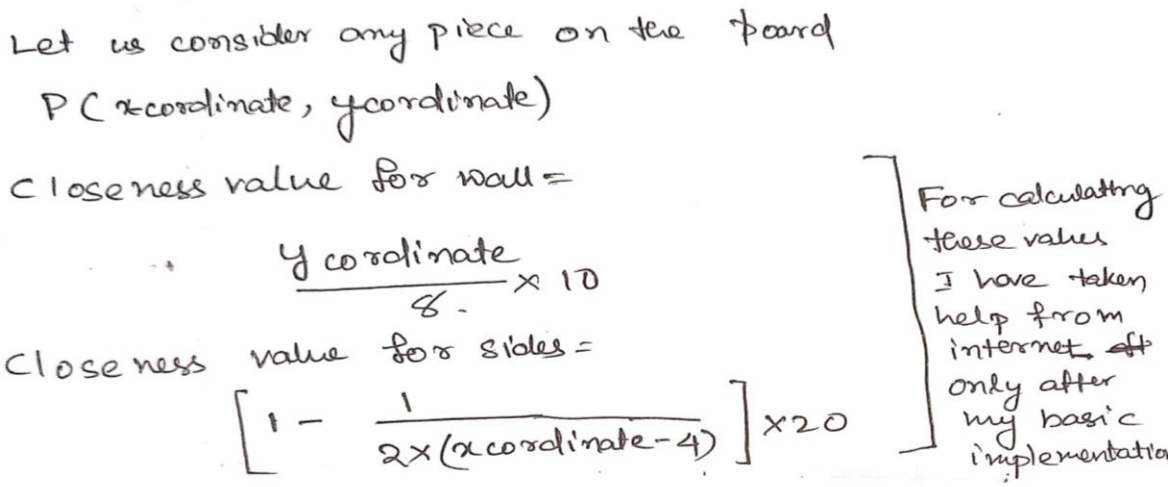
What constraints I had taken to consideration for further developments:
I found that the closeness towards the boundary plays a major role when it comes to the game checker. So, taking help from few sources I used following method to get my equations.

Calculations for development of ~~the~~ evaluation function which is more effective than the basic one. Let us consider the following board with co-ordinate system equivalent to monitor co-ordinate system.



The above board having → left area (wall)
right area (wall)
top side (player 1)
bottom side (player 2)

We can calculate a ~~positi~~ closeness value for a position of board that it is how close to the x-axis (side) / how close to the y-axis (wall)



I am using the above equations on my evaluation functions to increase efficiency
Here is my final evaluation function:

```
    self.pieceVal = 10
    self.kingVal = 50
    self.sideVal = 20
    self.wallVal = 10
    def evaluate_state(self, board):
        value = 0
        n1 = 0 #counter to check wining condition
        n2 = 0  #counter to check wining condition
        for i in range(10):
            for j in range(10):
                if not board[i][j] == 0:
                    if str(list(board[i][j].keys())[0]) == "ply" + str(self.ply):
                        n1 += 1
                        if i < 5:
                            value += int(1/(i+1))*self.sideVal
                        else:
                            value += int(1/(abs(i-10)))*self.sideVal
                        value += int((j+1)/10)*self.wallVal
                        if int(list(board[i][j].values())[0]) == 1:
                            value += self.pieceVal
                        elif int(list(board[i][j].values())[0]) == 2:
                            value += self.kingVal
                    else:
                        n2 += 1
                        if i < 5:
                            value -= int(1/(i+1))*self.sideVal
                        else:
                            value -= int(1/(abs(i-10)))*self.sideVal
                        value -= int(abs(j-10)/10)*self.wallVal
                        if int(list(board[i][j].values())[0]) == 1:
                            value -= self.pieceVal
                        elif int(list(board[i][j].values())[0]) == 2:
                            value -= self.kingVal
        if n2 == 0 and n1 > 0:
            value = 10000
        elif n1 == 0 and n2 > 0:
            value = -10000
        return value

```

Python file length : 21,397 lines : 519

5.b. The Minimax Algorithm:

Minimax (sometimes Minmax, MM[1] or saddle point[2]) is a decision rule used in artificial intelligence, decision theory, game theory, statistics, and philosophy for minimizing the possible loss for a worst case (maximum loss) scenario. When dealing with gains, it is referred to as "maximin"—to maximize the minimum gain. Originally formulated for two-player zero-sum game theory, covering both the cases where players take alternate moves and those where they make simultaneous moves, it has also been extended to more complex games and to general decision-making in the presence of uncertainty. (From Wikipedia)
The pseudocode for this algorithm is :

```
function minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, minimax(child, depth − 1, FALSE))
        return value
    else (* minimizing player *)
        value := +∞
        for each child of node do
            value := min(value, minimax(child, depth − 1, TRUE))
        return value
```

For checkers the algorithm I have used and verified by prof. Sai Prasad Sir is as follows.

```
Minimax (Board, depth, isMax)

    value = Evaluationfunction(Board)
    moves = find_moves(~~AMoves~~, Board)
    if (value = 10000)
        return value
```

```
if (depth > 0 )
    if  is max == True:
        Best value = - infinitye
        Best move = None
        for Move in moves
            Board = update (Board, move)
            value Move = minimax (Board, depth-1,
                                  false)
            t = Best value
            Best value = max ( Best value , value move)
            if t ≠ Best vale
                Best move = move
        return (Best value, Best move)

    else :
        Best value = + infinitye
        Best move = None
        for move in Moves
            Board = update (Board, move)
            value Move = minimax (Board, depth-1, True)
            t = Best value
            Best value = min ( Best vale , value move)
            if (t ≠ Best value)
                ↓ Best move = move
                return (Best value, Best move)
        return (Best value, Best move)

else:
    return (value, None)
```

CONCLUSION

Though our hard work and proper guidance, we have succeeded to complete this project. But there are some bugs in the program code while exiting the game(the wining condition) which will be modified in coming version.
Also, we'll try to overcome the limitation of minimax algorithm by using alpha beta pruning in the coming versions and also try to implement the various levels to the game by working with the depth and evaluation function.
For any type query including environment set up or better understanding you can contact any of us through following details

Biplaba Samantaray
Email : samantaraybiplaba@gmail.com
Phone No: 8637212213

Pallavi Kumari
Email : kpallavi9802@gmail.com

REFERENCES

https://en.wikipedia.org/wiki/Minimax
https://www.pygame.org/docs/
http://aima.eecs.berkeley.edu/books.html
https://www.thesprucecrafts.com/play-checkers-using-standard-rules-409287
https://www.youtube.com/watch?v=ScKIdStgAfU