**Team Members and contribution:**

**1.Revanth Siva Sai Ram Balineni – 1910110314 Percent – 37%**

**2.Raya Venkata Sathvik – 1910110437 Percent – 30%**

**3.Pulugu Vamshidhar Reddy – 1910110292 Percent – 33%**

**Requirements:**

1. **pygame package: can be installed by executing command “pip install pygame”**
2. **open the complete folder with your IDE so that you don’t have change paths for the images used in project**

**Main algo used is in minimax class**

**Strategy followed is evaluation of boards if a move is made. Board which has the highest score after depth of 4 moves was chosen to be final. We found depth of 4 to be most efficient considering time complexity. A depth of 4 strikes balance between intelligence and delay due to computation of scores for so many possible boards**

**CHECKERS**

We are Using Pygame in our assignment to implement checkers. It is a cross-platform set of Python modules which is used to create 2D games.

It consists of computer graphics and sound libraries designed to be used with the Python programming language.

**Drawing the Board :**

For displaying the game window we are using a method called as Pygame.display.setmode((width,height)) which displays the window with the given width and height

To display the title we use Pygame.display.set\_caption(name) method.

In our checkers game we are using two images creamtile and browntile for drawing the squares and black and white colors for drawing the pieces

For drawing the squares and pieces we are using Boardstruc and Piece class respectively.

For drawing the squares we are using the method draw \_squares which is in Boardstruc class and for drawing the pieces we are using print method which is in Piece class.

**Boardstruc Class :**

In the Boardstruc class we are using the following fields and methods.

**Board=[]** : It is used to store the values(whether the piece is present or not) of the squares in the board.

**Selected** : It is used to specify whether a square is selected or not.

**Posmoves={}** : It is a dictionary which stores the possible moves.

**Validmovecolor={}** : It is a dictionary which consists of all the pieces which have captures moves of a particular color

**Whitepieces,Blackpieces** : It stores the no of white or black pieces left

**Whitekings,blackkings** : It stores the number of white kings or black kings that are left.

**draw\_squares(window)** : In the draw\_squares function firstly we are attaching a background image to the window then we are giving a padding of 30 to top, left, right, bottom and then drawing squares to the window using two for loops. Here for squares we are using two images cream and brown.

**createboard()** : createboard functions initializes the two dimensional board list with pieces if it is present at that squares or initializes with 0 if any piece is not present at that squares.

**returnpiece(self, row, col)** : returnpiece function returns the piece at the given row and col if it is present else returns 0.

**evaluate(self) :.** It returns a score for particular permutation of the board. This function is used by minimax class for evaluating the board.

**getallpieces(self, color) :** This function returns a list of pieces with specified color.

**move(self, piece, row, col):** This function moves the piece to specified square(row,col) and if the specified square is present in the last row or first row then it makes that piece a king and increments the value of that color kings.

**draw(self, window):** This functiondraws the squares and draws the pieces at the position they are in according to board list.

**returncapturemovesforcolor(self, color, validchecker=[]):** loops through the entire board and returns the pieces which have capture moves for the specified color.

**forcecapture(self, piece) :** This function returns the possible moves for the piece if there are no capture moves for the color, If there are captures moves for the piece then it returns only those capture moves else it returns nothing**.**

**getposmoves(self, piece) :** This function returns a dictionary of possible moves of a piece.

**lefttraverse(self, start, stop, step, color, left, skipped=[])** : Traverses the left brach of the given piece.

**righttraverse(self, start, stop, step, color, right, skipped=[]) :** Traverses the right branch of the given piece

**animateMove(self,color,erow,ecol,srow,scol,window,skipped) :** Takes the ending position and starting position of the piece as input and animated the move from starting position to ending position

**removepiece(self, pieces) :** This function removes the piece and decrements the value of those color kings if the piece is a king else it decrements the values of those color pieces.

**winner(self) :** This function returns the string which is the name of the color of the winner. It also checks the condition in which all the moves of one player are blocked by other player so that the other player wins.

**Piece Class :**

**Padding :** It is used to set padding in between square and the circle

**Outline :** It is used to set border to the piece.

**row :** It is the row at which the current piece is present

**col :** It is the column at which the current piece is present

**color :** It is the color of the current piece

**isKing :** It specifies whether a piece is king or not

**x :** Initially it is set to zero

**y :** Initially it is set to zero

**selected :** Specifies whether a piece is selected or not.

**positioncal(self) :** This function sets x and y to centre of the square

**makeKing(self) :** This function sets true if a piece is king else sets it to false

**move(self, newrow, newcol) :** This function sets the piece’s row and column to given newrow and newcolum.

**print(self, window) :** This function draws the piece(i.e circle) with the piece’s color and also it attaches a crown image to the piece if the piece is a king .

**Class Random:**

**validpieces(self, boardst):** Returns a list of white pieces which can make valid moves corresponding to the state of the board

**validmovemaker(self, board, game):** Makes a random valid move following the random algorithm. The rand function in the method takes time as seed so that the choice made by the rand function is truly random and it changes every time the user plays against it

**Class minimax:**

**minimaxalgo(self,boardstate, depth, shouldmaximize, game):** takes the current board as input and returns the board after making the best possible choice on behalf of the computer. The strategy followed here is that the computer sends the list of all possible boards to be evaluated and selects the board which has highest possible score

**getallmoves(self,board, color, game):** returns all the moves that can be made in the given board state, deep copies the original board and makes those moves so that it could return these boards for minimax algorithm so that the algorithm can evaluate these boards and choose the best possible one

**replicatemove(self, piece, move, board, game, skip):** called by the getallmoves function so that it could make function could make a move on deep copies

**Gameplay Class:**

**current :** It is used to specify whether a piece is selected or not

**board :** It is used to assign instance of Boardstruc class

**Turn :** black color is assigned

**Possiblemoves** : stores the possible moves of a given piece.

**window** : It is the game window which is displayed on screen

**update(self) :** Updates the Pygame display by drawing the squares and drawing the pieces and drawing the possible moves.

**reset(self) :** It resets the gameplay by assigning all variables to initial values**.**

**move(self,row,col) :** It moves current piece to the specified row and column it also animates the move if its legal

**randommove(self, piece, row, col)** : It moves the piece selected by the random function to the specified row and column it also animates that move

**swapturns(self) :** This function swaps turns between white and black players

**selectedtile(self,row,col) :** moves the selected piece to the specified position if it is valid . if it is not valid then it calls the same function again until the user picks a valid row and column to move the piece

**drawpossiblemoves(self, moves) :** Iterates through the move dictionary and draws the possible moves

**win(self) :** It checks if there is a winner.

**returnboard(self) :** This function returns the current board

**minimaxmove(self, board) :** It changes the current board to the board returned by the minimax class**.**

**drawcase(self, currwhite, currblack) :** Checks the 50 move draw condition specified in the assignment.

**returnmouse(pos):** called for returning the position of the mouse when the user clicks the mouse

**main() :**  All the execution, method calling and choice making happens in this function. This function constantly checks if one of the users has won and updates the board constantly when moves are being made