**KARNATAK LAW SOCIETY’S**

**GOGTE INSTITUTE OF TECHNOLOGY**

**UDYAMBAG, BELAGAVI – 590008**

**(An Autonomous Institution under Visvesvaraya Technological University, Belagavi)**

**(Approved By AICTE, New Delhi)**

**DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING**

 

**COURSE PROJECT: ARTIFICIAL INTELLIGENCE**

**REPORT ON THREE MEN’S MORRIS GAME USING EASYAI**

Guided by: Prof. Shubha Sanu

Hemanth Tubachi 2GI18IS015

Laxmi Nyamagoud 2GI18IS020

Rachana Kampli 2GI18IS032

Rohan Kokatanur 2GI18IS066

**2020-2021**

**CERTIFICATE**



This is to certify that **Mr. Hemanth I T, Ms. Laxmi Nyamagoud, Ms. Rachana Kampli, Mr. Rohan Kokatanur** of **Sixth Semester** bearing **USN: 2GI18IS015, 2GI18IS020, 2GI18IS032, 2GI18IS066** has satisfactorily completed the course in Course activity of Artificial Intelligence. It can be considered as a bonafide work carried out for partial fulfillment of the academic requirement of 6th Semester B.E. (Information Science & Engineering) prescribed by KLS Gogte Institute of Technology, Belagavi during the academic year 2020-21.

The report has been approved as it satisfies the academic requirements prescribed for the said degree.

**Signature of The Faculty Member Signature of The HOD.**

Date: 21/05/2021

**TITLE:**

THREE MEN’S MORRIS GAME USING EASYAI

**ABSTRACT:**

Three Men's Morris is an abstract strategy game played on a three-by-three board (counting lines or nodes) that is similar to tic-tac-toe. It is a zero-sum game played on a 9-point grid(node) between two opponents each having three pieces; black and white. Here each player has 3 similar pieces and alternatively each player puts his piece in a node with a goal to make all of his/her pieces in a line as in tic-tac-toe. After all the pieces are placed in some nodes and similar pieces are not in a line, the pieces are moved to adjacent distance alternatively by each player with an aim to make his/her pieces in a row. In this game, the computer applies logic and competes against the human moves.

**OBJECTIVES:**

1) To learn about practical implementation of Artificial intelligence.

2) To create a simple AI game.

**INTRODUCTION:**

Three Men's Morris is a strategic symmetric two-player board game played on a three-by-three-point grid(node). Each player has three pieces. All the pieces have same movement i.e., each piece can move straight to adjacent node only. A player can move one piece at a time, and next player moves his/her piece. Each player has to block next player from making his/her dice in a line and aim to make his/her piece in a line. The line can be vertical, or horizontal. The winner is the first player to align their three pieces on a line drawn on the Board. The board is empty to begin the game and players take turns placing their pieces on empty intersections. Once all pieces are placed (assuming there is no winner by then), play proceeds with each player moving one of their pieces per turn. A piece may move only to any adjacent empty position. A player wins if thereby they get three pieces in a line. AFS is designed to perform well with larger numbers of active users than other distributed file systems.

**METHODOLOGY:**

The AI is a Negamax algorithm with alpha-beta pruning and transposition tables.

* **Negamax algorithm**:

Negamax search is a variant form of minimax search that relies on the zero-sum property of a two-player game.

This algorithm relies on the fact that {max (a, b) =-min (-a, -b)} to simplify the implementation of the minimax algorithm. More precisely, the value of a position to player A in such a game is the negation of the value to player B. Thus, the player on move looks for a move that maximizes the negation of the value resulting from the move: this successor position must by definition have been valued by the opponent. The reasoning of the previous sentence works regardless of whether A or B is on move. This means that a single procedure can be used to value both positions. This is a coding simplification over minimax, which requires that A selects the move with the maximum-valued successor while B selects the move with the minimum-valued successor.

* **Negamax with alpha beta pruning:**

Algorithm optimizations for minimax are also equally applicable for Negamax. Alpha-beta pruning can decrease the number of nodes the Negamax algorithm evaluates in a search tree in a manner similar with its use with the minimax algorithm. Alpha (α) and beta (β) represent lower and upper bounds for child node values at a given tree depth. Negamax sets the arguments α and β for the root node to the lowest and highest values possible.

When Negamax encounters a child node value outside an alpha/beta range, the Negamax search cuts off thereby pruning portions of the game tree from exploration. Cut offs are implicit based on the node return value. A node value found within the range of its initial α and β is the node's exact (or true) value. This value is identical to the result the Negamax base algorithm would return, without cut offs and without any α and β bounds. If a node return value is out of range, then the value represents an upper (if value ≤ α) or lower (if value ≥ β) bound for the node's exact value. Alpha-beta pruning eventually discards any value bound results. Such values do not contribute nor affect the Negamax value at its root node.

This pseudocode shows the fail-soft variation of alpha-beta pruning. Fail-soft never returns α or β directly as a node value. Thus, a node value may be outside the initial α and β range bounds set with a Negamax function call. In contrast, fail-hard alpha-beta pruning always limits a node value in the range of α and β.

This implementation also shows optional move ordering prior to the foreach loop that evaluates child nodes. Move ordering is an optimization for alpha beta pruning that attempts to guess the most probable child nodes that yield the node's score. The algorithm searches those child nodes first. The result of good guesses is earlier and more frequent alpha/beta cut offs occur, thereby pruning additional game tree branches and remaining child nodes from the search tree.

* **Negamax with alpha beta pruning and transposition tables:**

Transposition tables selectively memorize the values of nodes in the game tree. Transposition is a term reference that a given game board position can be reached in more than one way with differing game move sequences.

When Negamax searches the game tree, and encounters the same node multiple times, a transposition table can return a previously computed value of the node, skipping potentially lengthy and duplicate re-computation of the node's value. Negamax performance improves particularly for game trees with many paths that lead to a given node in common.

**ABOUT EASYAI:**

EasyAI is an open source software originally written by Zulko and released under the MIT license. It is hosted on GitHub, where you can submit improvements, get support, etc. EasyAI is an artificial intelligence framework for two-players abstract games such as Tic Tac Toe, Connect 4, Reversi, etc.

* **Human and AI players:**

The players can be either a Human\_Player() (which will be asked interactively which moves it wants to play) or a AI\_Player(algo).

If you are a human player, you will be asked to enter a move when it is your turn. You can also enter show moves to have a list of all moves allowed, or quit to quit. The variable algo is any function f(game)->move. It can be an algorithm that determines the best move by thinking N turns in advance.

The Negamax algorithm will always look for the shortest path to victory, or the longest path to defeat. It is possible to go faster by not optimizing this (the disadvantage being that the AI can then make suicidal moves if it has found that it will eventually lose against a perfect opponent). To do so, you must provide the argument win\_score to Negamax which indicates above which score a score is considered a win.

* **Interactive Play:**

If you are needing to be more interactive with the game play, such as when integrating with other frameworks, you can use the get\_move and play\_move methods instead. get\_move gets an AI player’s decision. play\_move executes a move (for either player).

**IMPLEMENTATION:**

# importing easyAI library components

from easyAI import TwoPlayersGame

from easyAI.Player import Human\_Player

# inheriting TwoPlayersGame class

class MMM(TwoPlayersGame):

    """

        The board positions are numbered as follows :

            1 2 3

            4 5 6

            7 8 9

    """

    def \_\_init\_\_(self, p):

        self.players = p

        self.board = [0 for i in range(9)]

        # first move player(1 starts)

        self.nplayer = 1

        self.winner=0

    def possible\_moves(self):

        # all possible moves on board

        lss=["12","14","21","23","25","32","36","41","45","47","52","54","56","58","65","63","69","74","78","85","87","89","96","98"]

        # now possible moves

        if(self.nmove>6):

            ls=[]

            x=self.nplayer

            for i in lss:

                a=list(i)

                if(self.board[int(a[1])-1]==0 and self.board[int(a[0])-1]==(x)):

                    ls.append(i)

            return(ls)

        else:

            return [i+1 for i, e in enumerate(self.board) if e == 0]

    def make\_move(self, move):

        if(self.nmove>6):

            # how to move after first 6 moves

            mov=list(str(move)) #[1,2]

            mov[0]=int(mov[0])

            mov[1]=int(mov[1])

            # board update

            self.board[int(mov[0])-1]=0

            self.board[int(mov[1])-1]=self.nplayer

        else:

            # how to move first 6 moves

            self.board[int(move)-1] = self.nplayer

    def lose(self):

        """ DID I LOSE ? LOSING CONDITIONS """

        ret = any([all([(self.board[c-1] == self.nopponent)for c in line])for line in [

                            [1, 2, 3], [4, 5, 6], [7, 8, 9], #horizontal wincases

                            [1, 4, 7], [2, 5, 8], [3, 6, 9]  #vertical wincases

                ]])

        return ret

    def is\_over(self):

        if((self.possible\_moves() == []) or self.lose()):

            self.winner=self.nopponent

        # limit to 20 moves and declare draw

        if(self.nmove>20):

            print("")

            print("\t\tMATCH DRAW\t\t")

            return 1

        return (self.possible\_moves() == []) or self.lose()

    def show(self):

        # things to show for each move

        print(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

        print("|\t\t\t\t\t\t\t |")

        print("|\tScores:\tPlayer1=",end=" ")

        print(p1,end="")

        print("\tPlayer2=",end=" ")

        print(p2,end="")

        print("\t\t |")

        print("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|")

        entries=['-','O','x']

        print("|\t\t\t\t\t\t\t |")

        for i in range(3):

            print("|\t\t",end="\t")          # 0 1 2

            for j in range(3):      # 0 1 2

                str=(entries[self.board[3\*i+j]])

                print(str,end=" ")

            print("\t\t\t\t |")

        print("|\t\t\t\t\t\t\t |")

        print("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|")

    def scoring(self):

        return -100 if self.lose() else 0

# Selects the difficulty level of game.

def mode\_selection():

    """

        Selects the difficulty level of game.

    """

    print("Difficulty Level ?? (Easy=E Medium=M Hard=H)  :",end="")

    diff=input()

    if(diff=="E" or diff=="e"):

        print(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

        print("|\t\t\t\t\t\t\t |")

        print("|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   EASY MODE SELECTED  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|")

        print("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|")

        print("")

        return 2

    if(diff=="M" or diff=="m"):

        print(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

        print("|\t\t\t\t\t\t\t |")

        print("|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  MEDIUM MODE SELECTED  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|")

        print("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|")

        print("")

        return 3

    if(diff=="H" or diff=="h"):

        print(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

        print("|\t\t\t\t\t\t\t |")

        print("|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   HARD MODE SELECTED  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|")

        print("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|")

        print("")

        return 4

    else:

        print("Enter a valid input!.")

        diff=askntell()

        return diff

# shows the rules initially

def show\_rules():

    print(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ")

    print("|\t\t\t\t\t\t\t |")

    print("|\t\t\tRULES\t\t\t\t |")

    print("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|")

    print("|\t\t\t\t\t\t\t |")

    print("|\t1. To begin: \t\t\t\t\t |")

    print("|\t   the board is empty, and each player has three |")

    print("|\t   pieces in hand. Players decide at random who  |")

    print("|\t   goes first.\t\t\t\t\t |")

    print("|\t\t\t\t\t\t\t |")

    print("|\t2. Placement & movement: \t\t\t |")

    print("|\t   Each player takes it in turn to place a piece |")

    print("|\t   on any intersection on the board. When all of |")

    print("|\t   the pieces are entered, players instead move  |")

    print("|\t   a piece on the board along a marked line to   |")

    print("|\t   the adjacent point a horizontal or vertical   |")

    print("|\t   line to win.\t\t\t\t\t |")

    print("|\t\t\t\t\t\t\t |")

    print("|\t3. To win: \t\t\t\t\t |")

    print("|\t   When one player has a straight line of three  |")

    print("|\t   of their own pieces, horizontally, vertically |")

    print("|\t   or diagonally along a marked line, that \t |")

    print("|\t   player wins the game.\t\t\t |")

    print("|\t\t\t\t\t\t\t |")

    print("|\t4. Board Layout (Numbering of Places): \t\t |")

    print("|\t   \t\t 1  2  3 \t\t\t |")

    print("|\t   \t\t 4  5  6 \t\t\t |")

    print("|\t   \t\t 7  8  9 \t\t\t |")

    print("|\t\t\t\t\t\t\t |")

    print("|\t5. To move: \t\t\t\t\t |")

    print("|\t   A peice from fourth place to fifth place, \t |")

    print("|\t   you can enter '45'. \t\t\t\t |")

    print("|\t\t\t\t\t\t\t |")

    print("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|")

    print("")

# shows options after each game

def game\_over\_scene():

    print("")

    print("\t\tGAME OVER\t\t")

    print("\t\tPlayer ",end="")

    print(a.winner,end=" ")

    print(" WON")

    print("")

    print("Do you want to play again ?     Y/N  :",end="")

# shows exit screen contents

def game\_exit\_scene():

            print(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

            print("|\t\t\t\t\t\t\t |")

            print("|\tScores:\tPlayer1=",end=" ")

            print(p1,end="")

            print("\tPlayer2=",end=" ")

            print(p2,end="")

            print("\t\t |")

            print("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|")

            print("")

            print("\t\t WINNER: PLAYER ",end="")

            if(p1>p2):

                print("1")

            else:

                print("2")

            print(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

            print("|\t\t\t\t\t\t\t |")

            print("|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   GAME EXIT  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|")

            print("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|")

            print("")

if \_\_name\_\_ == "\_\_main\_\_":

    from easyAI import AI\_Player, Negamax

    show\_rules()

    diff=mode\_selection()

    ai\_algo = Negamax(diff)

    rest=1

    p1,p2=0,0

    while(rest):

        #starting a game

        a=MMM([Human\_Player(), AI\_Player(ai\_algo)])

        a.play()

        # updating scores

        if(a.winner==1):

            p1+=1

        else:

            p2+=1

        # end UI

        game\_over\_scene()

        restart=(input())

        print("")

        if(restart=="N" or restart=="n"):

            game\_exit\_scene()

            break

**CONCLUSION:**

Using EasyAI, we can create intelligent two player games.

**REFERENCES:**

* [Three Men's Morris | Cyningstan](http://www.cyningstan.com/game/106/three-mens-morris)

( http://www.cyningstan.com/game/106/three-mens-morris )

* EasyAI Documentation

<http://zulko.github.io/easyAI/>

<http://zulko.github.io/easyAI/get_started.html>

* <https://en.wikipedia.org/wiki/Negamax>