A.I. Final Report

Tic-Tac-Toe A.I.

13016243: Artificial intelligence

Faculty of Engineering, KMITL

By

620011295 Worada Rangsriseaneepitak

62011277 Thawanrat Atthawiwatkul

62011286 Unn Jertjamjarat

Project title:

- Tic-Tac-Toe A.I.

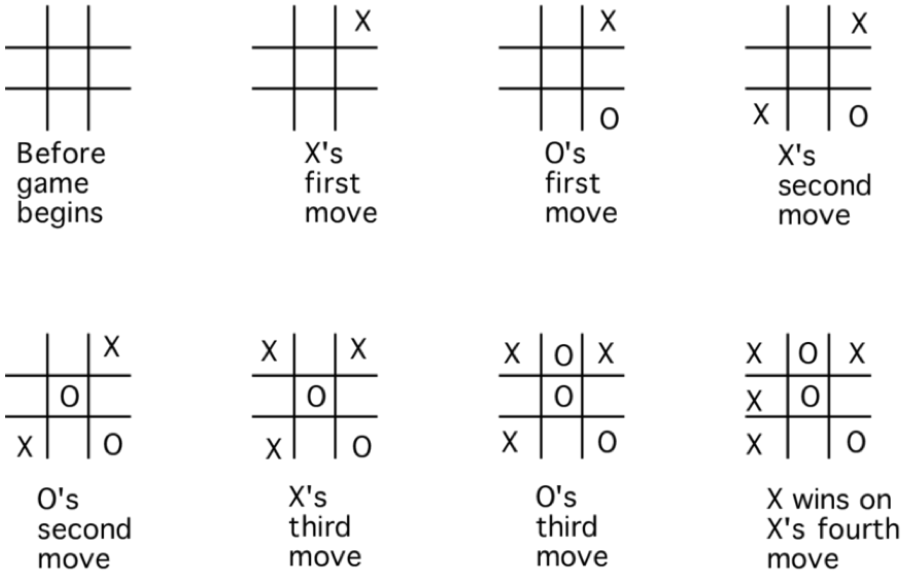
Technical requirement:

- Prolog -> SWI prolog
- Python -> VS code

Project Description:

A Tic-Tac-Toe Artificial Intelligence program that can play Tic Tac Toe with you(computer will be the player). You can select X(first) and O(second). The computer will play their best to win.

Tic-tac-toe is a very popular game for two players, X and O, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a vertical, horizontal or diagonal row wins the game.



AI concepts applied in the project

The algorithm implemented in this project is akin to a Utility-based agent, in which it tries to accomplish the goal which is predicting the best moves of the game by observing the placement of the symbol.

Property 1: The game admits the player that uses this **optimal strategy** will win or draw but it will not lose.

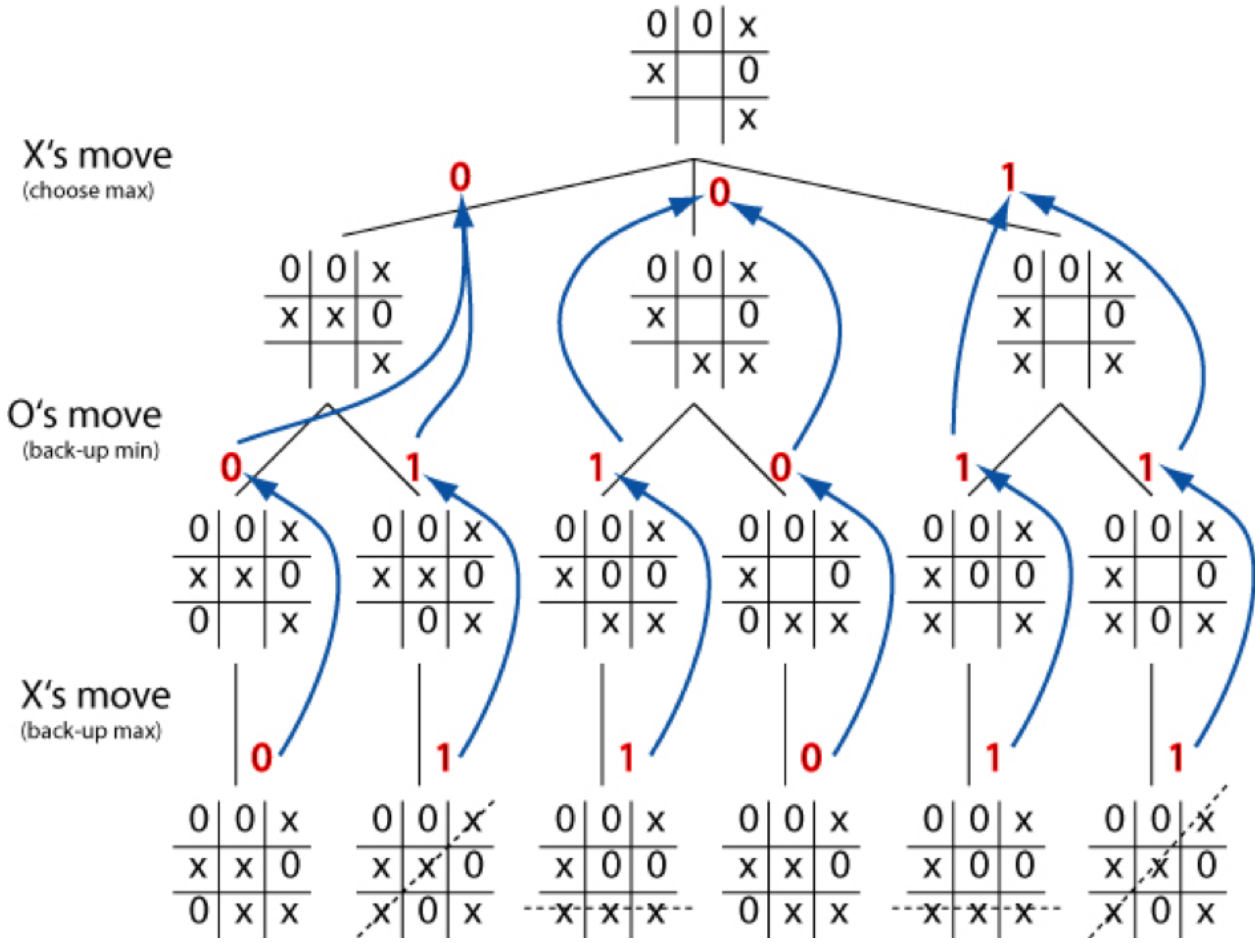
Property 2: The number of possible different matches is relatively small.

From properties 1 and 2 it follows that a practical, and general, algorithm to win/draw the game is to use the **MiniMax.**

At each turn the algorithm evaluates all the possible consequences of each move (possible due to property 2) and chooses the one that will ensure a victory or a draw (possible due to property 1).

An AI player that chooses each move with the alpha beta search algorithm will never lose. To make the game more realistic it is nice to introduce a stochastic factor so that each time with a predefined probability the AI player moves randomly rather than following the alpha beta algorithm. This

will make the game more realistic as it will make the AI player more human and sometimes it will lose.



PEAS

Agent Type	**Performance Measure**	**Environment**	**Actuator**	**Sensor**
Utility-based agent	Verify duration, Symbol pattern variety	The Table, The symbol	Put symbol in the table	Board Sensor

Description:

- Performance Measure:

- Verify duration

The time that agent to verify the environment and generate the information and suggestion.

-Symbol pattern variety

When pressing the generate button, the agent will generate the random pattern of symbols in the table. The pattern should be different but not fill all the table grid.

- Environment:

-The Table

The table in which the symbol is placed. If the table is full but no side wins, the game would be tied.al

- **The symbol**

  Each symbol in the table is used to decide the next suggested move and other information. If the same symbol is aligned, that symbol side wins.

- Actuator:

- Put symbol in the table

The agent will display the symbol(x or o) in the table according to the user or program generated.

- Sensor:

- Board Sensor

The current state of the board is scanned by the sensor in order to plan out how to place the Tetrominos.

Task Environment and their Characteristics

Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete	Known
Tic tac toe	Fully Observable	Single	Deterministic	Sequential	Static	Discrete	Known

Description:

- Fully Observable

- The AI has full access to the state of the environment at any point in time, that is it can always know the state of the.

- Single Agent

- There is only one agent playing the game, so it is clearly a single agent environment.

- Deterministic

- The Tic-tac-toe game is totally dependent on the player placement.The next state of the board is completely determined by the agent.

- Sequential

- The board is always affected by each decision of the agent, and any placement of the symbol lasts until the end of the session.

- Static

- The board technically does not change while the agent is thinking.

- Discrete

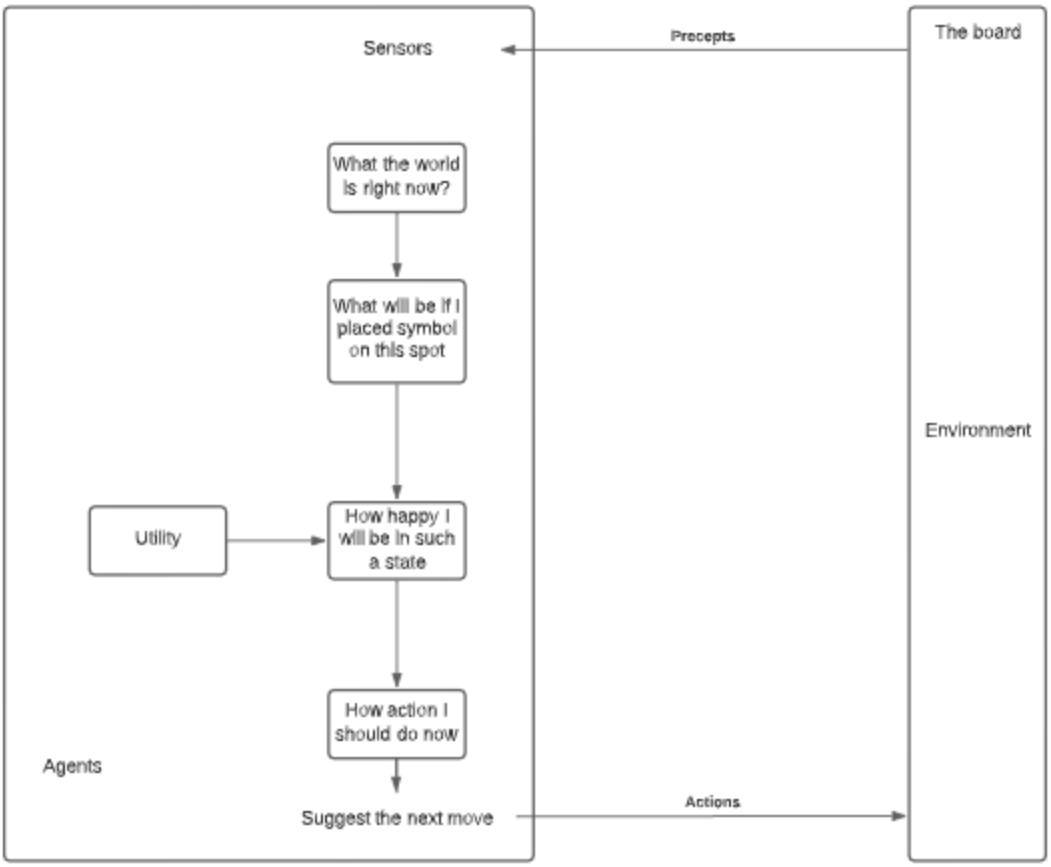
- There are a finite number of states in which the placement can be in, this again is dependent on the placement of the symbol.

- Known

- The agent has knowledge about how tic-tac-toe works.

Architecture of the Agent

https://lucid.app/lucidchart/14f4326d-71e0-4057-962d-1e38f2e36f42/edit?viewport_loc=120%2C72%2C2219%2C1021%2C0_0&invitationId=inv_66a099c6-9705-4d7f-be61-d08e40508ce7



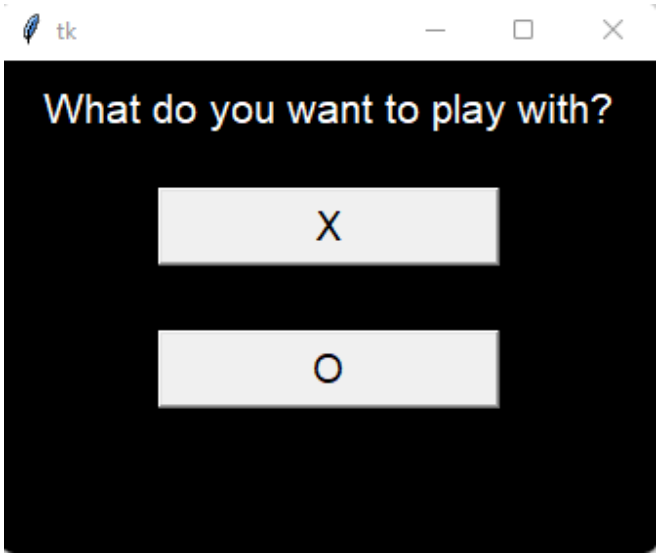
What functionalities do the Prolog Parts do?

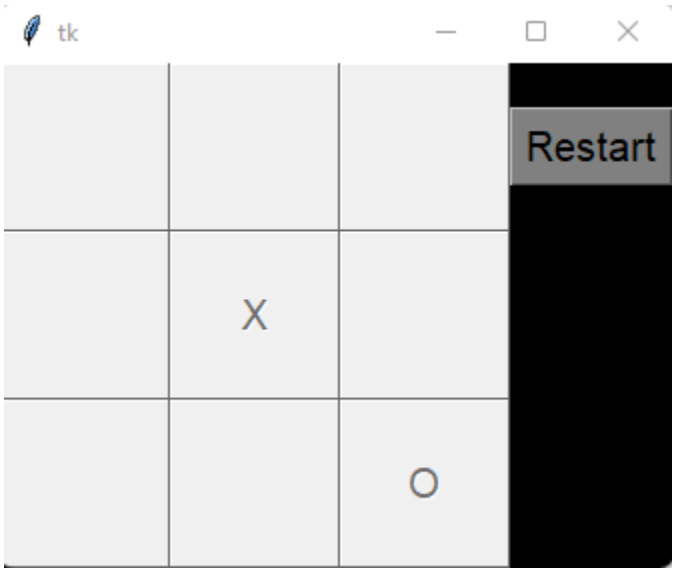
- The prolog part is used to do the logical part of the game.
- It is used to do the logic of movement.
- The algorithm used in the prolog is Minimax.

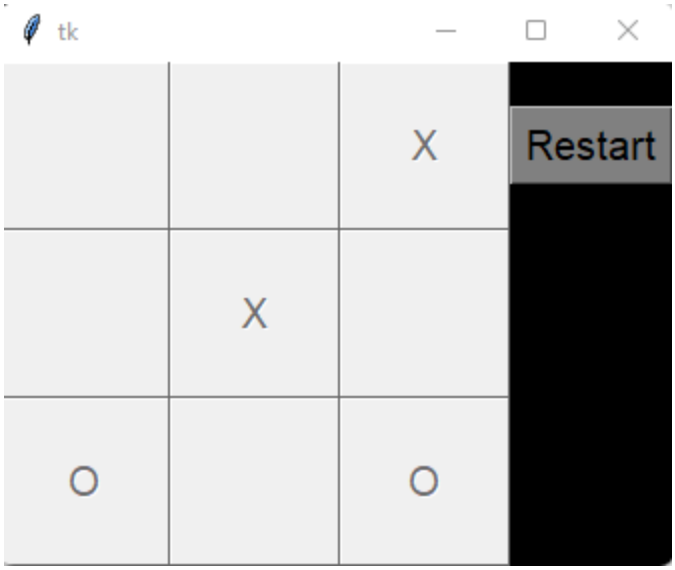
What functionalities do the Python Parts do?

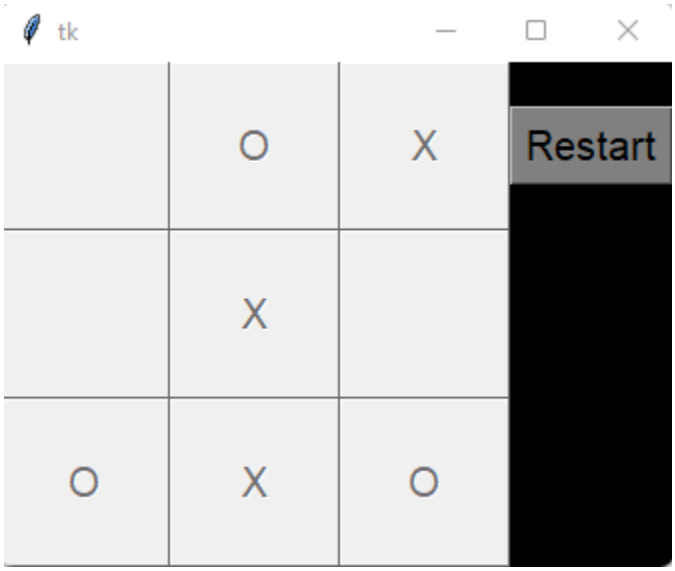
- It prints the table of Tic Tac Toe using TKinter.
- It is used to display the UI part of the program.

Screenshot

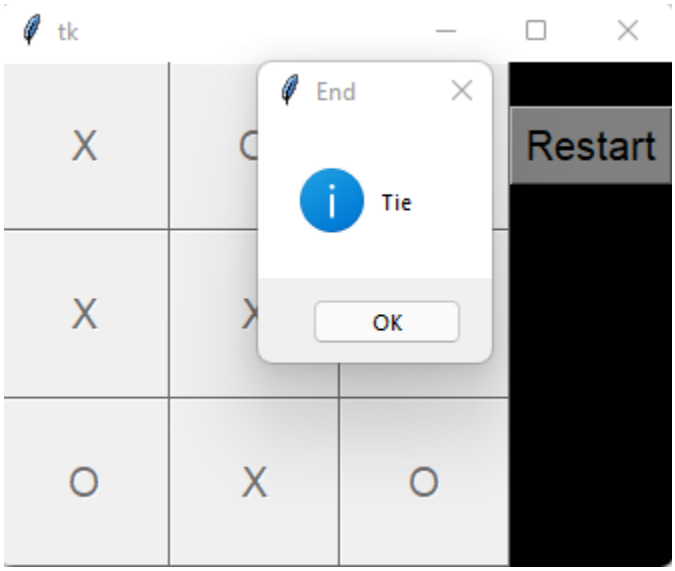





tk
X
Restart
X
O
O


tk
O
X
Restart
X
O
X
O


tk
O
X
Restart
X
X
O
O
X
O


tk
X
X
O
X
O
End
Tie
OK
Restart

Source Code

connectionProlog.py

```
from pyswip.prolog import Prolog



prolog = Prolog()



prolog.consult('tictactoegame.pl')



def moveHuman(player, state, board, pos):
    for move in
prolog.query(f'humanMove([{player},{state},{board}],NewB,{pos})'):
        newPlayer = move['NewB'][0]
        newState = str(move['NewB'][1])
        newBoard = move['NewB'][2]

    return newPlayer, newState, newBoard



def moveMachine(player, state, board):
    for move in prolog.query(f'bestMove([{player}, {state},
{board}],NewB)'):
        newPlayer = move['NewB'][0]
        newState = str(move['NewB'][1])
        newBoard = move['NewB'][2]
```

```
    return newPlayer, newState, newBoard
```

Main.py

```
import play


if __name__ == '__main__':
    app = play.TicTacToe()
    app.mainloop()
```

Play.py

```
import tkinter as tk
import tkinter.messagebox as mssg
import connectionProlog as pl


LARGE_FONT = ('Arial', 16)
MEDIUM_FONT = ('Arial', 15)
SMALL_FONT = ('Arial', 12)



class TicTacToe(tk.Tk):


    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)


        container = tk.Frame(self)
```

```
        container.pack(side='top', fill='both', expand=True)


        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)


        self.order = None


        self.frames = {}
        for F in (StartPage, BoardPage):
            frame = F(container, self)
            self.frames[F] = frame
            frame.grid(row=0, column=0, sticky='nsew')


        self.show_frame(StartPage)


    def get_page(self, page_class):
        return self.frames[page_class]


    def show_frame(self, container):
        frame = self.frames[container]
        frame.tkraise()


    def startGame(self, desicion):
        self.order = ['human', 'machine'] if desicion == 1 else
['machine', 'human']


        if self.order[0] == 'machine':
```

```
            page = self.get_page(BoardPage)
            page.machineTurn()

        self.show_frame(BoardPage)


class StartPage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent, bg='black')

        self.controller = controller

        label = tk.Label(self, text='What do you want to play with?',
font=LARGE_FONT,
                         bg='black', fg='white')
        label.pack(pady=10, padx=10)

        option1 = tk.Button(self, text="X", font=MEDIUM_FONT, width=15,
                            command=lambda: self.controller.startGame(1))
        option2 = tk.Button(self, text="O", font=MEDIUM_FONT, width=15,
                            command=lambda: self.controller.startGame(2))


        option1.pack(pady=15, padx=10)
        option2.pack(pady=18, padx=10)
```

```
class BoardPage(tk.Frame):

    def __init__(self, parent, controller):

        tk.Frame.__init__(self, parent, bg='black')


        self.controller = controller

        bw = 3

        bh = 7



        self.boardObjects = []


        self.board = [0, 0, 0, 0, 0, 0, 0, 0, 0]

        self.player = 1

        self.state = 'play'



        pos1 = tk.Button(self, text=' ', height=bw, width=bh,

                         font=MEDIUM_FONT, borderwidth=1,

                         command=lambda: self.humanTurn(1))

        pos1.grid(row=2, column=1)

        self.boardObjects.append(pos1)


        pos2 = tk.Button(self, text=' ', height=bw, width=bh,

                         font=MEDIUM_FONT, borderwidth=1,

                         command=lambda: self.humanTurn(2))

        pos2.grid(row=2, column=2)
```

```
        self.boardObjects.append(pos2)


        pos3 = tk.Button(self, text=' ', height=bw, width=bh,
                         font=MEDIUM_FONT, borderwidth=1,
                         command=lambda: self.humanTurn(3))
        pos3.grid(row=2, column=3)
        self.boardObjects.append(pos3)


        pos4 = tk.Button(self, text=' ', height=bw, width=bh,
                         font=MEDIUM_FONT, borderwidth=1,
                         command=lambda: self.humanTurn(4))
        pos4.grid(row=3, column=1)
        self.boardObjects.append(pos4)


        pos5 = tk.Button(self, text=' ', height=bw, width=bh,
                         font=MEDIUM_FONT, borderwidth=1,
                         command=lambda: self.humanTurn(5))
        pos5.grid(row=3, column=2)
        self.boardObjects.append(pos5)


        pos6 = tk.Button(self, text=' ', height=bw, width=bh,
                         font=MEDIUM_FONT, borderwidth=1,
                         command=lambda: self.humanTurn(6))
        pos6.grid(row=3, column=3)
        self.boardObjects.append(pos6)


        pos7 = tk.Button(self, text=' ', height=bw, width=bh,
```

```
                        font=MEDIUM_FONT, borderwidth=1,
                        command=lambda: self.humanTurn(7))
        pos7.grid(row=4, column=1)
        self.boardObjects.append(pos7)

        pos8 = tk.Button(self, text=' ', height=bw, width=bh,
                        font=MEDIUM_FONT, borderwidth=1,
                        command=lambda: self.humanTurn(8))
        pos8.grid(row=4, column=2)
        self.boardObjects.append(pos8)

        pos9 = tk.Button(self, text=' ', height=bw, width=bh,
                        font=MEDIUM_FONT, borderwidth=1,
                        command=lambda: self.humanTurn(9))
        pos9.grid(row=4, column=3)
        self.boardObjects.append(pos9)

        rs = tk.Button(self, text='Restart', bg='grey',
                       font=MEDIUM_FONT, borderwidth=2,
                       command=lambda: self.restart())
        rs.grid(row=2, column=5)

    def disAll(self):
        for obj in self.boardObjects:
            obj.config(state=tk.DISABLED)

    def setBoard(self):
```

```
        for value, obj in zip(self.board, self.boardObjects):
            if value != 0:
                obj['text'] = 'X' if value == 1 else 'O'
                obj.config(state=tk.DISABLED)
            else:
                obj['text'] = ' '
                obj.config(state=tk.ACTIVE)

    def restart(self):
        self.board = [0, 0, 0, 0, 0, 0, 0, 0, 0]
        self.state = 'play'
        self.player = 1

        self.setBoard()
        self.controller.show_frame(StartPage)

    def humanTurn(self, position):

        if self.state == 'play':
            self.player, self.state, self.board =
pl.moveHuman(self.player, self.state, self.board, position)
            self.setBoard()
            self.checkWinner()
            self.machineTurn()

    def machineTurn(self):
```

```
        if self.state == 'play':
            self.player, self.state, self.board =
pl.moveMachine(self.player, self.state, self.board)
            self.setBoard()
            self.checkWinner()


    def checkWinner(self):


        if self.state == 'win':


            winner = self.controller.order[0] if self.player == 2 \
                else self.controller.order[1]


            message = f'The winner is {winner}'
            mssg.showinfo('Fin', message)




            self.disAll()


        elif self.state == 'draw':
            mssg.showinfo('End', 'Tie')
            self.disAll()
```

Minimax.py

```
:- module(minimax, [minimax/3]).


% minimax(Pos, BestNextPos, Val)

% Pos is a position, Val is its minimax value.

% Best move from Pos leads to position BestNextPos.

minimax(Pos, BestNextPos, Val) :-                    % Pos has successors

    bagof(NextPos, move(Pos, NextPos), NextPosList),

    best(NextPosList, BestNextPos, Val), !.


minimax(Pos, _, Val) :-                      % Pos has no successors

    utility(Pos, Val).



best([Pos], Pos, Val) :-

    minimax(Pos, _, Val), !.


best([Pos1 | PosList], BestPos, BestVal) :-

    minimax(Pos1, _, Val1),

    best(PosList, Pos2, Val2),

    betterOf(Pos1, Val1, Pos2, Val2, BestPos, BestVal).




betterOf(Pos0, Val0, _, Val1, Pos0, Val0) :-    % Pos0 better than Pos1

    min_to_move(Pos0),                          % MIN to move in Pos0
```

```
    Val0 > Val1, !                          % MAX prefers the greater
value

    ;

    max_to_move(Pos0),                      % MAX to move in Pos0

    Val0 < Val1, !.                         % MIN prefers the lesser
value


betterOf(_, _, Pos1, Val1, Pos1, Val1).     % Otherwise Pos1 better
than Pos0
```

Tictactoe.pl

```
:- module(tictactoe,
[move/2,min_to_move/1,max_to_move/1,utility/2,winPos/2,drawPos/2]).


% move(+Pos, -NextPos)

% True if there is a legal (according to rules) move from Pos to NextPos.

move([X1, play, Board], [X2, win, NextBoard]) :-

    nextPlayer(X1, X2),

    move_aux(X1, Board, NextBoard),

    winPos(X1, NextBoard), !.


move([X1, play, Board], [X2, draw, NextBoard]) :-

    nextPlayer(X1, X2),

    move_aux(X1, Board, NextBoard),

    drawPos(X1,NextBoard), !.


move([X1, play, Board], [X2, play, NextBoard]) :-
```

```
    nextPlayer(X1, X2),
    move_aux(X1, Board, NextBoard).


% move_aux(+Player, +Board, -NextBoard)
% True if NextBoard is Board whith an empty case replaced by Player mark.
move_aux(P, [0|Bs], [P|Bs]).


move_aux(P, [B|Bs], [B|B2s]) :-
    move_aux(P, Bs, B2s).



% min_to_move(+Pos)
% True if the next player to play is the MIN player.
% Was changed or -> 2 for better handling in Python
min_to_move([2, _, _]).


% max_to_move(+Pos)
% True if the next player to play is the MAX player.
% Changed x -> 2 for better handling in Python
max_to_move([1, _, _]).


% utility(+Pos, -Val) :-
% True if Val the the result of the evaluation function at Pos.
% We will only evaluate for final position.
% So we will only have MAX win, MIN win or draw.
% We will use  1 when MAX win
%              -1 when MIN win
```

```
%              0 otherwise.
% Changed x -> 1 for better handling in Python
% Was changed or -> 2 for better handling in Python
utility([2, win, _], 1).       % Previous player (MAX) has win.
utility([1, win, _], -1).      % Previous player (MIN) has win.
utility([_, draw, _], 0).


% winPos(+Player, +Board)
% True if Player win in Board.
winPos(P, [X1, X2, X3, X4, X5, X6, X7, X8, X9]) :-
    equal(X1, X2, X3, P) ;    % 1st line
    equal(X4, X5, X6, P) ;    % 2nd line
    equal(X7, X8, X9, P) ;    % 3rd line
    equal(X1, X4, X7, P) ;    % 1st col
    equal(X2, X5, X8, P) ;    % 2nd col
    equal(X3, X6, X9, P) ;    % 3rd col
    equal(X1, X5, X9, P) ;    % 1st diag
    equal(X3, X5, X7, P).     % 2nd diag


% drawPos(+Player, +Board)
% True if the game is a draw.
drawPos(_,Board) :-
    \+ member(0, Board).


% equal(+W, +X, +Y, +Z).
% True if W = X = Y = Z.
equal(X, X, X, X).
```

Tictactoegame.pl

```
:- use_module(minimax).

:- use_module(tictactoe).


% bestMove(+Pos, -NextPos)

% Compute the best Next Position from Position Pos

% with minimax or alpha-beta algorithm.

bestMove(Pos, NextPos) :-

    minimax(Pos, NextPos, _).




% play

% Start the game.

play :-

    nl,

    write('====================='), nl,

    write('= Prolog TicTacToe ='), nl,

    write('====================='), nl, nl,

    write(' x starts the game'), nl,

    playAskColor.




% playAskColor

% Ask the color for the human player and start the game with it.

playAskColor :-

    nl, write('Color for human player ? (x or o)'), nl,
```

```
    read(Player), nl,
    (
    Player \= o, Player \= x, !,     % If not x or o -> not a valid color
      write('Error : not a valid color !'), nl,
      playAskColor                    % Ask again
      ;
      EmptyBoard = [0, 0, 0, 0, 0, 0, 0, 0, 0],
      show(EmptyBoard), nl,

      % Start the game with color and emptyBoard
      % Is modified to x by 1 for better handling in Python
      play([1, play, EmptyBoard], Player)
    ).



% play(+Position, +HumanPlayer)
% If next player to play in position is equal to HumanPlayer -> Human must play
% Ask to human what to do.
play([Player, play, Board], Player) :- !,
    nl, write('Next move ?'), nl,
    read(Pos), nl,                                % Ask human where to play
    (
      humanMove([Player, play, Board], [NextPlayer, State, NextBoard], Pos), !,
      show(NextBoard),
      (
```

```
        State = win, !,                                 % If Player win ->
stop

        nl, write('End of game : '),

        write(Player), write(' win !'), nl, nl

        ;

        State = draw, !,                                % If draw -> stop

        nl, write('End of game : '),

        write(' draw !'), nl, nl

        ;

        play([NextPlayer, play, NextBoard], Player) % Else -> continue the
game

      )

      ;

      write('-> Bad Move !'), nl,                 % If humanMove fail ->
bad move

      play([Player, play, Board], Player)         % Ask again

    ).




% play(+Position, +HumanPlayer)

% If it is not human who must play -> Computer must play

% Compute the best move for computer with minimax or alpha-beta.

play([Player, play, Board], HumanPlayer) :-

    nl, write('Computer play : '), nl, nl,

    % Compute the best move

    bestMove([Player, play, Board], [NextPlayer, State, BestSuccBoard]),

    show(BestSuccBoard),
```

```
    (
      State = win, !,                                  % If Player win ->
stop
      nl, write('End of game : '),
      write(Player), write(' win !'), nl, nl
      ;
      State = draw, !,                                 % If draw -> stop
      nl, write('End of game : '), write(' draw !'), nl, nl
      ;
      % Else -> continue the game
      play([NextPlayer, play, BestSuccBoard], HumanPlayer)
    ).




% nextPlayer(X1, X2)
% True if X2 is the next player to play after X1.
nextPlayer(2, 1).
nextPlayer(1, 2).


% When human play
humanMove([X1, play, Board], [X2, State, NextBoard], Pos) :-
    nextPlayer(X1, X2),
    set1(Pos, X1, Board, NextBoard),
    (
      winPos(X1, NextBoard), !, State = win ;
      drawPos(X1,NextBoard), !, State = draw ;
```

```
      State = play
    ).




% set1(+Elem, +Pos, +List, -ResList).
% Set Elem at Position Pos in List => Result in ResList.
% Rem : counting starts at 1.
set1(1, E, [X|Ls], [E|Ls]) :- !, X = 0.


set1(P, E, [X|Ls], [X|L2s]) :-
    number(P),
    P1 is P - 1,
    set1(P1, E, Ls, L2s).



% show(+Board)
% Show the board to current output.
show([X1, X2, X3, X4, X5, X6, X7, X8, X9]) :-
    write('   '), show2(X1),
    write(' | '), show2(X2),
    write(' | '), show2(X3), nl,
    write('  -----------'), nl,
    write('   '), show2(X4),
    write(' | '), show2(X5),
    write(' | '), show2(X6), nl,
    write('  -----------'), nl,
```

```
    write('   '), show2(X7),
    write(' | '), show2(X8),
    write(' | '), show2(X9), nl.




% show2(+Term)
% Write the term to current outupt
% Replace 0 by ' '.
show2(X) :-
    X = 0, !,
    write(' ').

show2(X) :-
    write(X).
```