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.

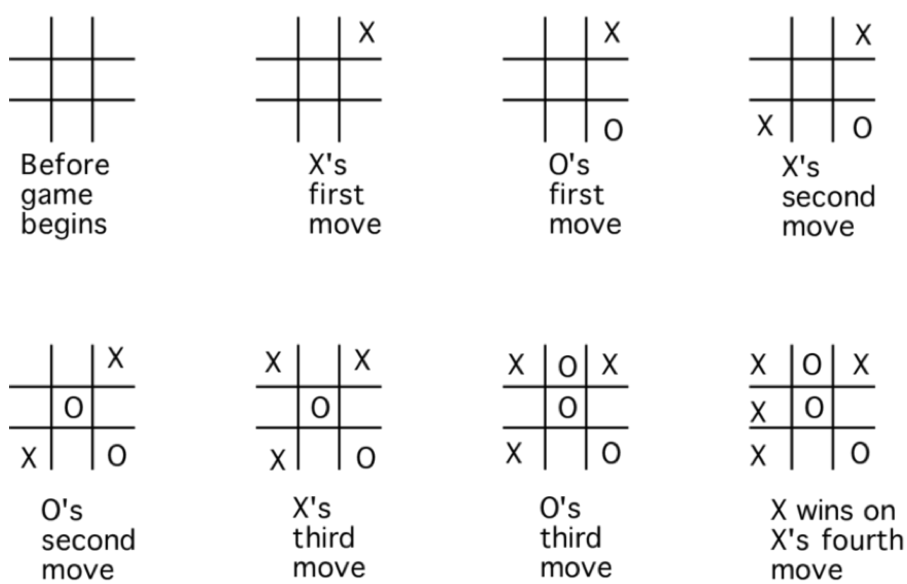
Technical requirement:

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

Project Description:

A Tic-Tac-Toe Artificial Intelligence program that tells what is the best move to win the game with an explanation of the A.I. generated outcome. This is the program where you can practice how to play Tic-Tac-Toe with the smartest move and to win within the shortest time. This game implements the Prolog programming language and Python.

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.

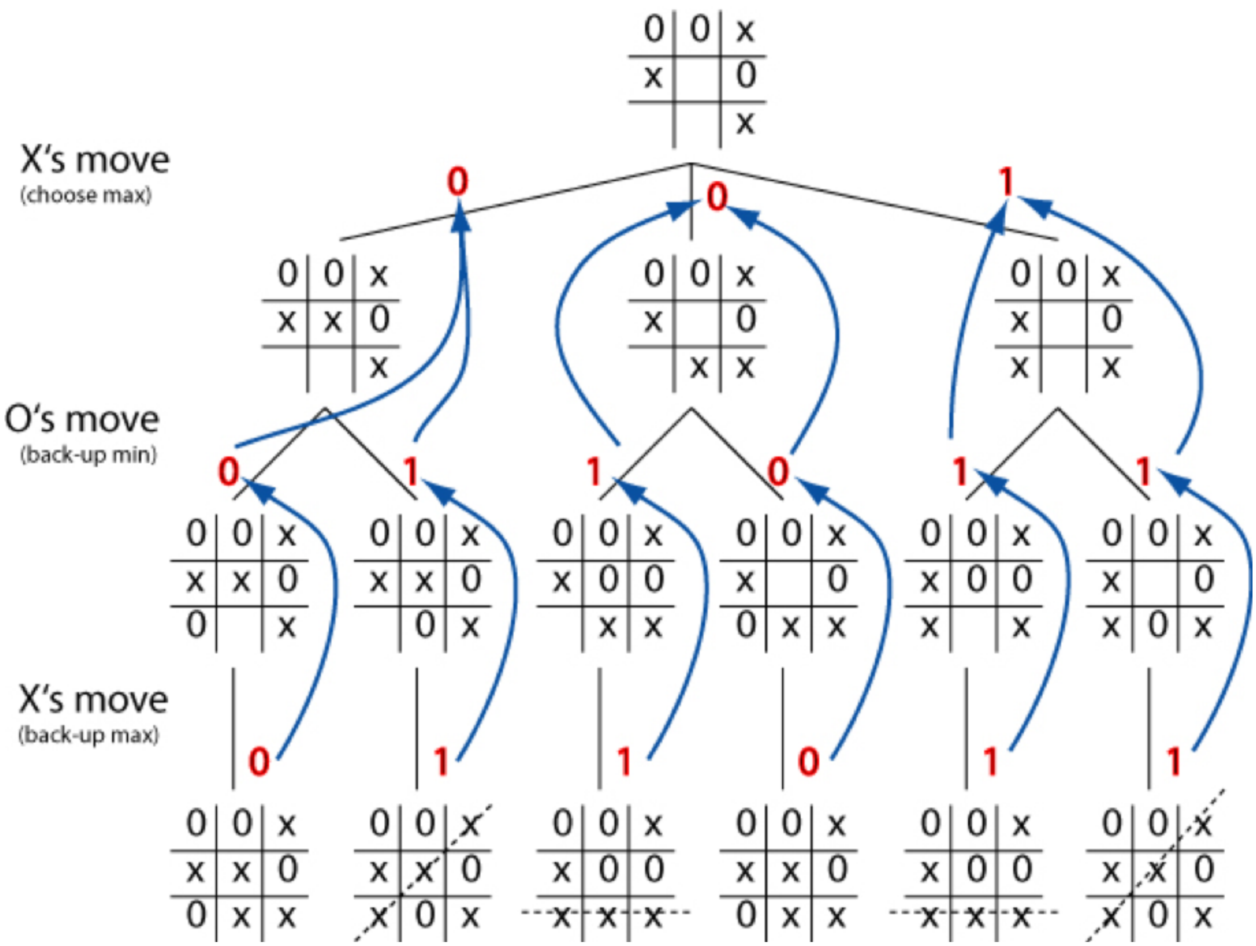
Property 1: The game admits the player that uses this **optimal strategy** will win or draw but it will not lose.

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 **Alpha Beta search**.

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

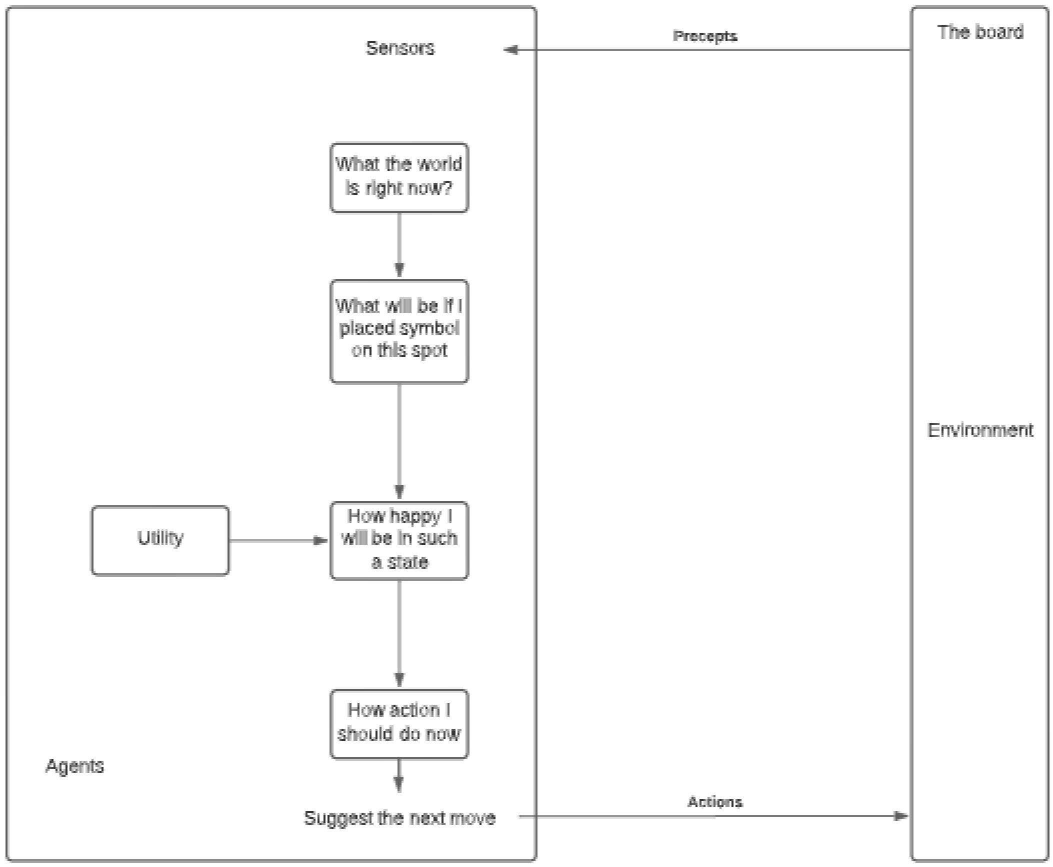
- There are a finite number of states in which the placement can be in, this again is dependent on the placement of the symbol.

- 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



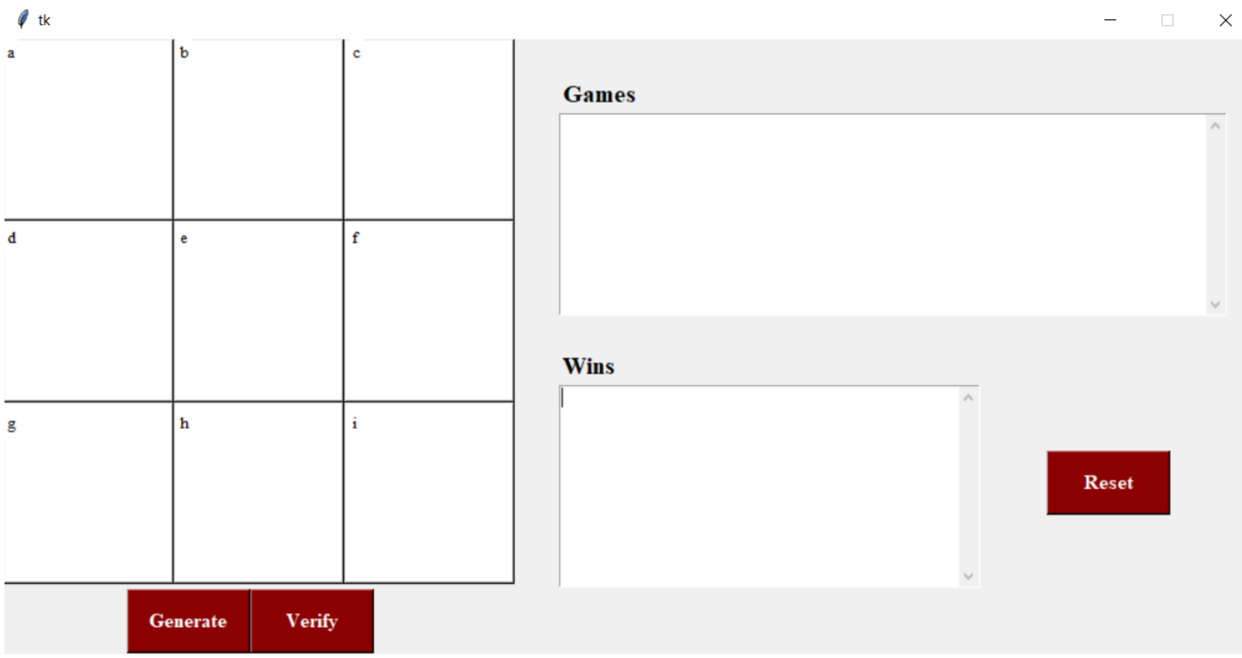
What functionalities do the Prolog Parts do?

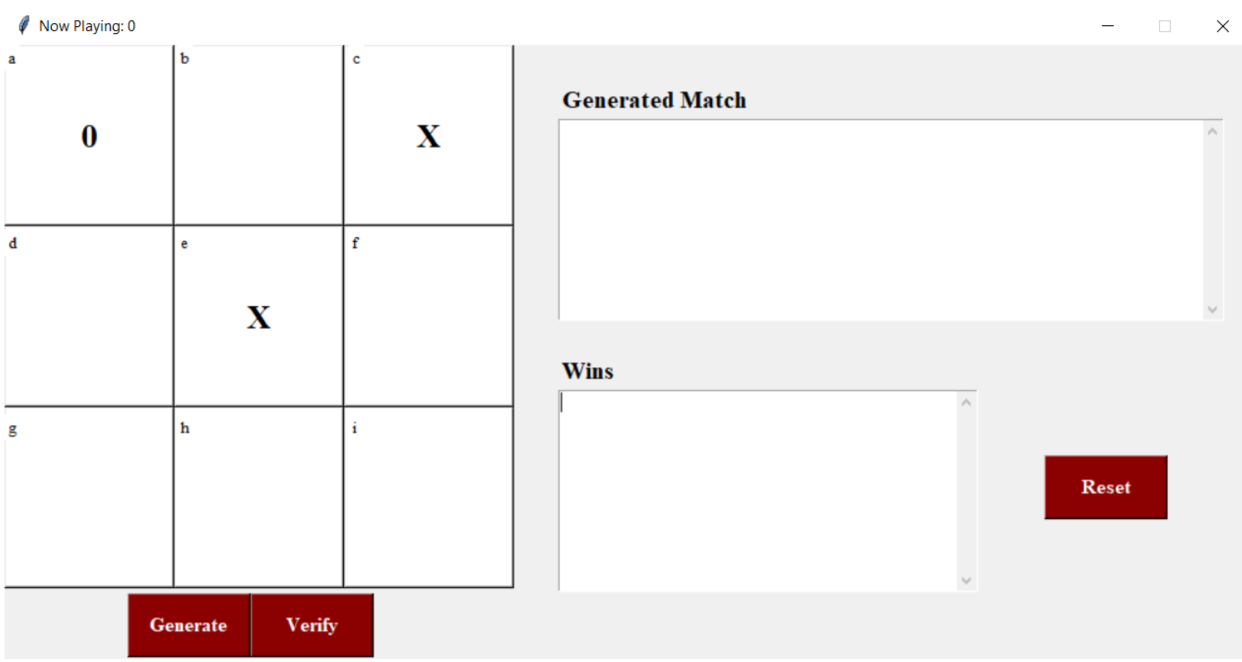
- The prolog part is used to do the logical part of the game.
- It is used to do the logic of movement.

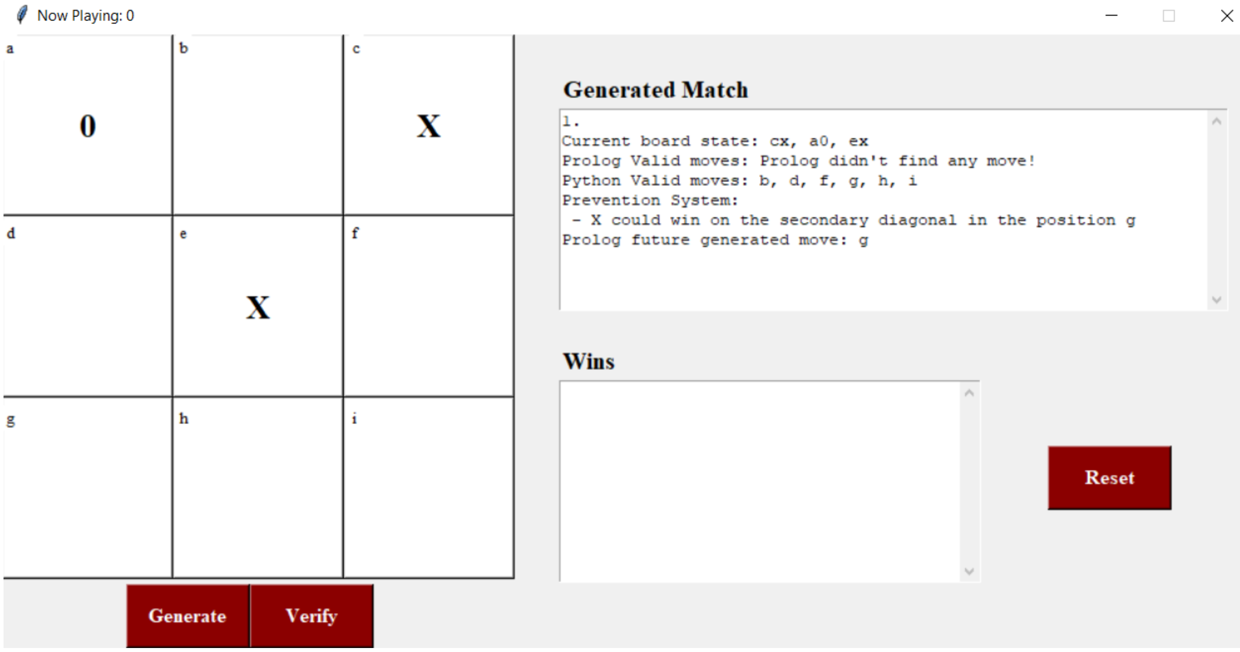
What functionalities do the Python Parts do?

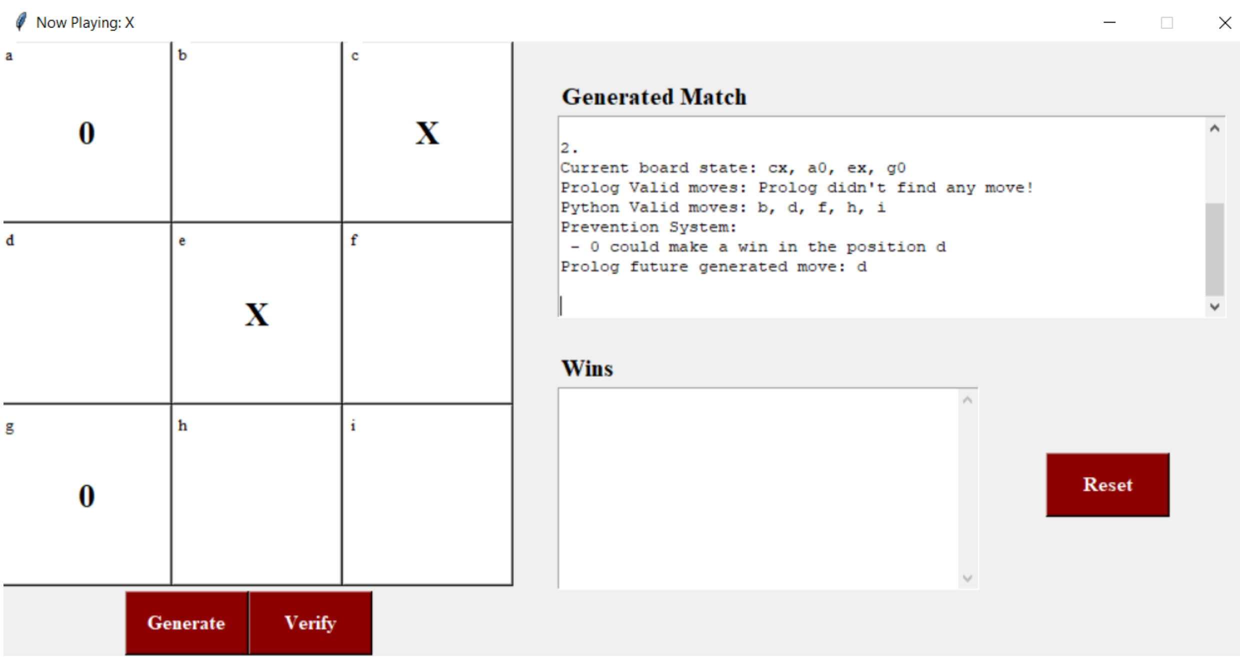
- It prints the table of Tic Tac Toe using TKinter.
- It is used to display the UI part of the program.

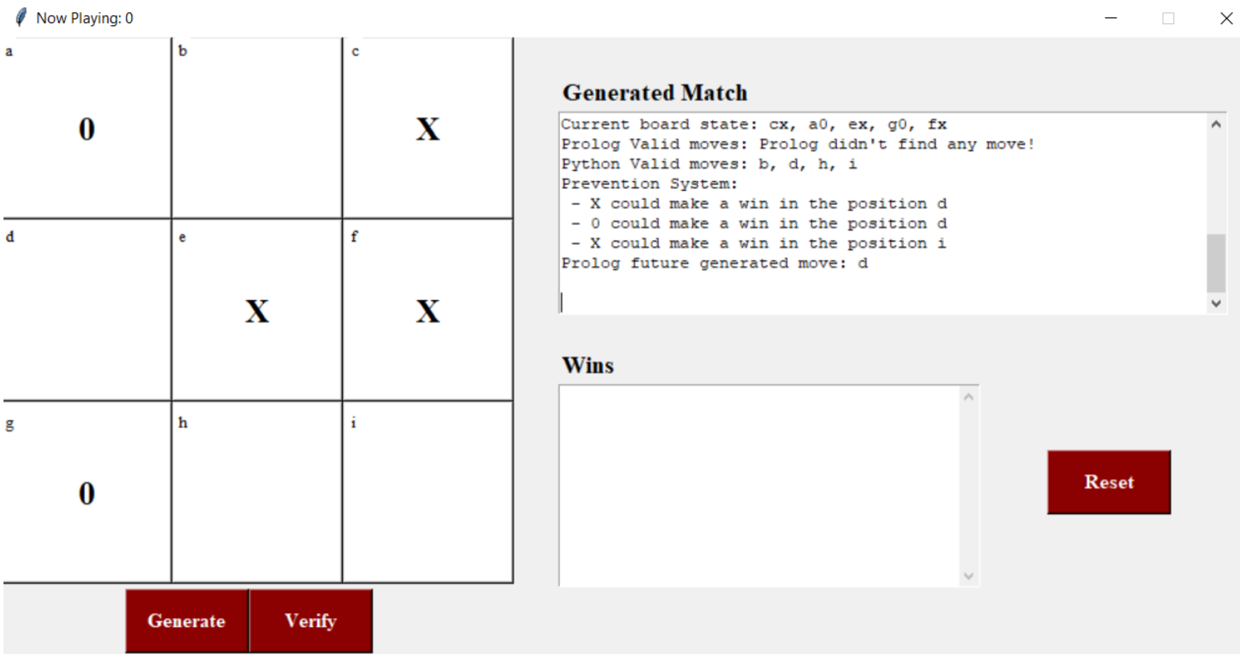
Screenshot

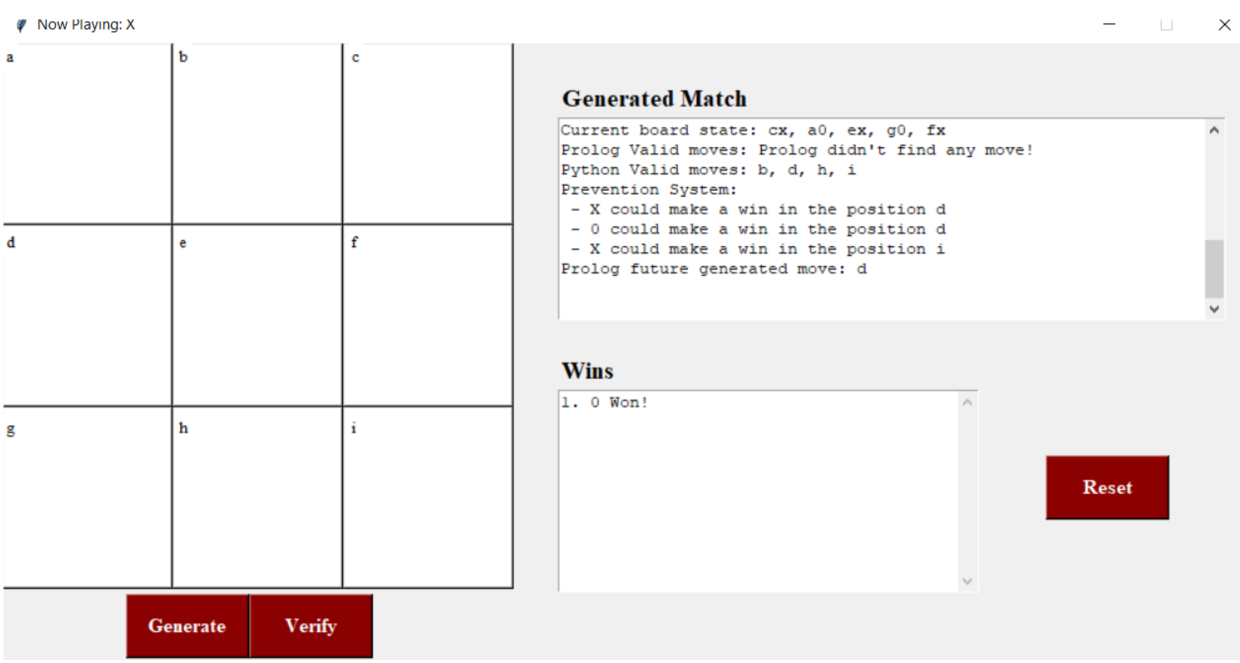

tk
a
b
c
d
e
f
g
h
i
Games
Wins
Reset
Generate
Verify


Now Playing: 0
a
0
b
c
X
d
e
X
f
g
h
i
Generated Match
Wins
Reset
Generate
Verify


Now Playing: 0
a
b
c
0
X
d
e
f
X
g
h
i
Generate
Verify
Generated Match
1.
Current board state: cx, a0, ex
Prolog Valid moves: Prolog didn't find any move!
Python Valid moves: b, d, f, g, h, i
Prevention System:
 - X could win on the secondary diagonal in the position g
Prolog future generated move: g
Wins
Reset


Now Playing: X
a
b
c
0
X
d
e
f
X
g
h
i
0
Generate
Verify
Generated Match
2.
Current board state: cx, a0, ex, g0
Prolog Valid moves: Prolog didn't find any move!
Python Valid moves: b, d, f, h, i
Prevention System:
 - 0 could make a win in the position d
Prolog future generated move: d
Wins
Reset


Now Playing: 0
a
0
b
c
X
d
e
X
f
X
g
0
h
i
Generate
Verify
Generated Match
Current board state: cx, a0, ex, g0, fx
Prolog Valid moves: Prolog didn't find any move!
Python Valid moves: b, d, h, i
Prevention System:
- X could make a win in the position d
- 0 could make a win in the position d
- X could make a win in the position i
Prolog future generated move: d
Wins
Reset


Now Playing: X
a
b
c
d
e
f
g
h
i
Generate
Verify
Generated Match
Current board state: cx, a0, ex, g0, fx
Prolog Valid moves: Prolog didn't find any move!
Python Valid moves: b, d, h, i
Prevention System:
- X could make a win in the position d
- 0 could make a win in the position d
- X could make a win in the position i
Prolog future generated move: d
Wins
1. 0 Won!
Reset

Source Code

ref.py

```
from pathlib import Path

from threading import Thread

import itertools


import random

from time import sleep

from pyswip import Prolog

from tkinter import *

import tkinter.messagebox


prolog = Prolog()

prolog2 = Prolog()

prolog.consult("../Prolog/test.pl")

prolog2.consult("../Prolog/program.pl")

list(prolog.query("init."))


tk = Tk()

tk.geometry("1014x500+450+152")

tk.resizable(0, 0)


prev = 0

debug = True

game, game1, game2 = 1, 1, 1

bclick = True
```

```python
flag = 0

win = False

board_matrix = [["a", "b", "c"], ["d", "e", "f"], ["g", "h", "i"]]

moves_list = ["a", "b", "c", "d", "e", "f", "g", "h", "i"]

learned = False


def generate():

    global game, game2, win, prev

    game = 1

    prev = 0

    reset()

    txtOutput.delete('0.0', END)

    txtOutput1.delete('0.0', END)

    main_label.config(text='Generated Match')

    lista = []

    index = random.randint(2, 4)

    for _ in range(index):

        poz = random.choice(button_list)

        if poz not in lista:

            lista.append(poz)

            button_click(poz)



def combine(lista):

    x = list(itertools.permutations(lista))

    ll = map(list, x)

    return list(ll)
```

```python
def remaining_spots():
    remaining = []
    for index, button in enumerate(button_list):
        if button["text"] != "X" and button["text"] != "0":
            remaining.append(moves_list[index])
    return remaining


def find_pos(lista, player):
    for x in lista:
        if player in x:
            return lista.index(x)


def dontlethimwin():
    global board_matrix, prev

    moves_lista = []
    for index, button in enumerate(button_list):
        if button["text"] == "X":
            moves_lista.append("{}{}".format(moves_list[index], "X"))
        elif button["text"] == "0":
            moves_lista.append("{}{}".format(moves_list[index], "0"))
        else:
            moves_lista.append("GG")
```

```python
    table = [moves_lista[x:x + 3] for x in range(0, len(moves_lista), 3)]
    # print("TABLE")
    # for x in table:
    #     print(x)
    txtOutput.insert(END, "Prevention System:")
    dontwin = []

    for x in table:
        contorx = 0
        contory = 0
        for y in x:
            if "0" in y:
                contory += 1
            elif "X" in y:
                contorx += 1
        if contory == 2:
            pozitie = find_pos(x, "GG")
            if pozitie is not None:
                dontwin.append(f" - 0 could make a win in the position
{board_matrix[table.index(x)][pozitie]}")
                prev += 1
        elif contorx == 2:
            pozitie = find_pos(x, "GG")
            if pozitie is not None:
                dontwin.append(f" - X could make a win in the position
{board_matrix[table.index(x)][pozitie]}")
                prev += 1
```

```
    for x in range(3):

        contorx = 0

        contory = 0

        coloana = []

        for y in range(3):

            coloana.append(table[y][x])

            if "0" in table[y][x][1]:

                contory += 1

            elif "X" in table[y][x][1]:

                contorx += 1

        if contory == 2:

            pozitie = find_pos(coloana, "GG")

            if pozitie is not None:

                dontwin.append(f" - 0 could make a win in the position 
{board_matrix[pozitie][x]}")

                prev += 1

        elif contorx == 2:

            pozitie = find_pos(coloana, "GG")

            if pozitie is not None:

                dontwin.append(f" - X could make a win in the position 
{board_matrix[pozitie][x]}")

                prev += 1


    diag1 = [table[i][i] for i in range(3)]

    diag2 = ([table[3 - 1 - i][i] for i in range(3 - 1, -1, -1)])


    contorx = 0
```

```
    contory = 0
    for x in diag1:
        if "0" in x[1]:
            contory += 1
        elif "X" in x[1]:
            contorx += 1
    if contory == 2:
        pozitie = find_pos(diag1, "GG")
        if pozitie is not None:
            dontwin.append(
                f" - 0 could make a win on the main diagonal in the
position {board_matrix[pozitie][pozitie]}")
            prev += 1
    elif contorx == 2:
        pozitie = find_pos(diag1, "GG")
        if pozitie is not None:
            dontwin.append(
                f" - X could win on the main diagonal in the position
{board_matrix[pozitie][pozitie]}")
            prev += 1

    contorx = 0
    contory = 0
    for x in diag2:
        if "0" in x[1]:
            contory += 1
        elif "X" in x[1]:
            contorx += 1
```

```
    if contory == 2:

        pozitie = find_pos(diag2, "GG")

        if pozitie is not None:

            dontwin.append(

                f" - 0 could win on the secondary diagonal in the position
{board_matrix[pozitie][abs(2 - pozitie)]}")

            prev += 1

    elif contorx == 2:

        pozitie = find_pos(diag2, "GG")

        if pozitie is not None:

            dontwin.append(

                f" - X could win on the secondary diagonal in the position
{board_matrix[pozitie][abs(2 - pozitie)]}")

            prev += 1


    if prev == 0:

        txtOutput.insert(END, "\tNo possible move at this state to win")

    else:

        for x in dontwin:

            txtOutput.insert(END, "\n" + str(x))

    txtOutput.insert(END, "\n" + "Prolog future generated move:\t")

    with open("../Prolog/input.txt", "r") as fd:

        ceva = fd.read()

    if ceva != "None":

        input_tmp =
str(Path("../Prolog/input.txt").resolve()).replace("\\", "/")

        list(prolog.query("read_from_file('{}').".format(input_tmp)))
```

```
        with open("output.txt", "r") as fd:
            command = fd.read()
            txtOutput.insert(END, moves_list[int(command[1]) - 1])
    txtOutput.insert(END, "\n\n")



def verify():
    global game2
    if main_label['text'] == "Games":
        txtOutput.delete('0.0', END)
        txtOutput1.delete('0.0', END)
    main_label.config(text='Generated Match')
    listax = []
    lista0 = []
    lista2x = []
    lista20 = []
    for index, button in enumerate(button_list):
        if button["text"] == "X":
            listax.append("{}{}".format(moves_list[index], "x"))
            lista20.append("{}{}".format(moves_list[index], "0"))

        if button["text"] == "0":
            lista0.append("{}{}".format(moves_list[index], "0"))
            lista2x.append("{}{}".format(moves_list[index], "x"))

    lista_mare = []
    for x in range(min(len(listax), len(lista0))):
```

```
        lista_mare.append(listax[x])

        lista_mare.append(lista0[x])


    if len(listax) > len(lista0):

        lista_mare.append(listax[-1])

    elif len(listax) < len(lista0):

        lista_mare.append(lista0[-1])


    lista_mare2 = []

    for x in range(min(len(lista2x), len(lista20))):

        lista_mare2.append(lista2x[x])

        lista_mare2.append(lista20[x])


    if len(lista2x) > len(lista20):

        lista_mare2.append(lista2x[-1])

    elif len(lista2x) < len(lista20):

        lista_mare2.append(lista20[-1])


    mutari = []

    txtOutput.insert(END, str(game2) + ".\n" + "Current board state:\t")

    game2 += 1

    string = ""

    for x in lista_mare:

        string += x + ", "

    string = string[:-2]

    txtOutput.insert(END, string + "\n")
```

```
string_tmp = ""
for x in lista_mare:
    string_tmp += str(x) + " "
if not string_tmp:
    string_tmp = "None"
with open("../Prolog/input.txt", "w+") as fd:
    fd.write(string_tmp)

txtOutput.insert(END, "Prolog Valid moves:\t")

lista_mare = combine(lista_mare)
for x in lista_mare:
    query = "verify({},V).".format(x)
    val = list(prolog.query(query))
    if val:
        mutari.append(val[0]['V'].decode('ascii'))

lista_mare2 = combine(lista_mare2)
for x in lista_mare2:
    query = "verify({},V).".format(x)
    val = list(prolog.query(query))
    if val:
        mutari.append(val[0]['V'].decode('ascii'))

mutari = list(set(mutari))
while '\n' in mutari:
    mutari.remove('\n')
```

```
for index, button in enumerate(button_list):
    if button["text"] == "X" or button["text"] == "0":
        if moves_list[index] in mutari:
            mutari.remove(moves_list[index])

if not mutari:
    txtOutput.insert(END, "Prolog didn't find any move!\n")
else:
    string2 = ""
    for x in mutari:
        string2 += x + ", "
    string2 = string2[:-2]
    txtOutput.insert(END, string2 + "\n")

remaining = remaining_spots()
string = ""
for x in remaining:
    string += x + ", "
string = string[:-2]
txtOutput.insert(END, "Python Valid moves:\t")
txtOutput.insert(END, str(string) + "\n")

dontlethimwin()

txtOutput.see(END)
```

```
def button_click(buttons, player=None):
    global bclick, flag
    if buttons["text"] == " " and bclick:
        if player is None:
            player = "X"
        buttons["text"] = player
        bclick = False
        tk.title("Now Playing: 0")
        win_check()
        flag += 1

    elif buttons["text"] == " " and not bclick:
        if player is None:
            player = "0"
        buttons["text"] = player
        bclick = True
        tk.title("Now Playing: X")
        win_check()
        flag += 1
    else:
        tkinter.messagebox.showinfo("Tic-Tac-Toe", "Button already
Clicked!")



def reset():
    global flag, win, bclick, game2
    button1["text"] = button2["text"] = button3["text"] = button4["text"]
= button5["text"] = button6["text"] = \
```

```
        button7["text"] = button8["text"] = button9["text"] = " "
    flag = 0
    win = True
    bclick = True
    game2 = 1


def fullreset():
    global game, game1
    reset()
    game, game1 = 1, 1
    txtOutput.delete('0.0', END)
    txtOutput1.delete('0.0', END)


def win_check():
    global game
    if (button1["text"] == "X" and button2["text"] == "X" and
button3["text"] == "X" or
            button4["text"] == "X" and button5["text"] == "X" and
button6["text"] == "X" or
            button7["text"] == "X" and button8["text"] == "X" and
button9["text"] == "X" or
            button1["text"] == "X" and button5["text"] == "X" and
button9["text"] == "X" or
            button3["text"] == "X" and button5["text"] == "X" and
button7["text"] == "X" or
            button1["text"] == "X" and button4["text"] == "X" and
button7["text"] == "X" or
```

```
            button2["text"] == "X" and button5["text"] == "X" and
button8["text"] == "X" or

            button3["text"] == "X" and button6["text"] == "X" and
button9["text"] == "X"):

        # tkinter.messagebox.showinfo("Tic-Tac-Toe", "X Won!")

        txtOutput1.insert(END, str(game) + "." + " " + "X Won!" + "\n")

        txtOutput1.see(END)

        game += 1

        reset()

        return True


    elif flag == 8:

        # tkinter.messagebox.showinfo("Tic-Tac-Toe", "It is a Tie")

        txtOutput1.insert(END, str(game) + "." + " " + "Tie!\n")

        txtOutput1.see(END)

        game += 1

        reset()

        return True


    elif (button1["text"] == "0" and button2["text"] == "0" and
button3["text"] == "0" or

          button4["text"] == "0" and button5["text"] == "0" and
button6["text"] == "0" or

          button7["text"] == "0" and button8["text"] == "0" and
button9["text"] == "0" or

          button1["text"] == "0" and button5["text"] == "0" and
button9["text"] == "0" or

          button3["text"] == "0" and button5["text"] == "0" and
button7["text"] == "0" or
```

```
            button1["text"] == "O" and button4["text"] == "O" and
button7["text"] == "O" or
            button2["text"] == "O" and button5["text"] == "O" and
button8["text"] == "O" or
            button3["text"] == "O" and button6["text"] == "O" and
button9["text"] == "O"):
        # tkinter.messagebox.showinfo("Tic-Tac-Toe", "O Won!")
        txtOutput1.insert(END, str(game) + "." + " " + "O Won!" + "\n")
        txtOutput1.see(END)
        game += 1
        reset()
        return True
    return False


button1 = Button(tk, text=" ", font="Times 20 bold", bg="white",
fg="black", height=4, width=8,
                 command=lambda: button_click(button1))
button1.grid(row=3, column=0)

button2 = Button(tk, text=" ", font="Times 20 bold", bg="white",
fg="black", height=4, width=8,
                 command=lambda: button_click(button2))
button2.grid(row=3, column=1)

button3 = Button(tk, text=" ", font="Times 20 bold", bg="white",
fg="black", height=4, width=8,
                 command=lambda: button_click(button3))
button3.grid(row=3, column=2)
```

```
button4 = Button(tk, text=" ", font="Times 20 bold", bg="white",
fg="black", height=4, width=8,
                 command=lambda: button_click(button4))
button4.grid(row=4, column=0)

button5 = Button(tk, text=" ", font="Times 20 bold", bg="white",
fg="black", height=4, width=8,
                 command=lambda: button_click(button5))
button5.grid(row=4, column=1)

button6 = Button(tk, text=" ", font="Times 20 bold", bg="white",
fg="black", height=4, width=8,
                 command=lambda: button_click(button6))
button6.grid(row=4, column=2)

button7 = Button(tk, text=" ", font="Times 20 bold", bg="white",
fg="black", height=4, width=8,
                 command=lambda: button_click(button7))
button7.grid(row=5, column=0)

button8 = Button(tk, text=" ", font="Times 20 bold", bg="white",
fg="black", height=4, width=8,
                 command=lambda: button_click(button8))
button8.grid(row=5, column=1)

button9 = Button(tk, text=" ", font="Times 20 bold", bg="white",
fg="black", height=4, width=8,
                 command=lambda: button_click(button9))
```

```
button9.grid(row=5, column=2)


button_list = [button1, button2, button3, button4, button5, button6,
button7, button8, button9]


buttongenerate = Button(tk, text="Generate", font="Times 12 bold",
bg="red4", fg="white", height=2, width=10,
                        command=generate).place(x=100, y=445)


buttongo = Button(tk, text="Verify", font="Times 12 bold", bg="red4",
fg="white", height=2, width=10,
                  command=verify).place(x=200, y=445)


buttonreset = Button(tk, text="Reset", font="Times 12 bold", bg="red4",
fg="white", height=2, width=10,
                     command=fullreset).place(x=845, y=333)


if debug:
    Label(tk, text="a", font="Times 10", bg="white").place(x=0, y=0)
    Label(tk, text="b", font="Times 10", bg="white").place(x=140, y=0)
    Label(tk, text="c", font="Times 10", bg="white").place(x=280, y=0)
    Label(tk, text="d", font="Times 10", bg="white").place(x=0, y=150)
    Label(tk, text="e", font="Times 10", bg="white").place(x=140, y=150)
    Label(tk, text="f", font="Times 10", bg="white").place(x=280, y=150)
    Label(tk, text="g", font="Times 10", bg="white").place(x=0, y=300)
    Label(tk, text="h", font="Times 10", bg="white").place(x=140, y=300)
    Label(tk, text="i", font="Times 10", bg="white").place(x=280, y=300)
```

```
main_label = Label(tk, text="Games", font="Times 15 bold", justify=RIGHT)
main_label.place(x=450, y=30)
Label(tk, text="Wins", font="Times 15 bold").place(x=450, y=250)
txtFrame = Frame(tk, borderwidth=1, relief="sunken")
txtOutput = Text(txtFrame, wrap=NONE, height=10, width=65, borderwidth=0)
vscroll = Scrollbar(txtFrame, orient=VERTICAL, command=txtOutput.yview)
txtOutput["yscroll"] = vscroll.set

vscroll.pack(side="right", fill="y")
txtOutput.pack(side="left", fill="both", expand=True)

txtFrame.place(x=450, y=60)

txtFrame1 = Frame(tk, borderwidth=1, relief="sunken")
txtOutput1 = Text(txtFrame1, wrap=NONE, height=10, width=40,
borderwidth=0)
vscroll1 = Scrollbar(txtFrame1, orient=VERTICAL, command=txtOutput1.yview)
txtOutput1["yscroll"] = vscroll1.set

vscroll1.pack(side="right", fill="y")
txtOutput1.pack(side="left", fill="both", expand=True)

txtFrame1.place(x=450, y=280)

tk.mainloop()
```

Program.pl

```
win(Board, Who) :- row_case_win(Board, Who).

win(Board, Who) :- col_case_win(Board, Who).

win(Board, Who) :- diag_case_win(Board, Who).


row_case_win(Board, Who) :- Board = [Who,Who,Who,_,_,_,_,_,_].

row_case_win(Board, Who) :- Board = [_,_,_,Who,Who,Who,_,_,_].

row_case_win(Board, Who) :- Board = [_,_,_,_,_,_,Who,Who,Who].


col_case_win(Board, Who) :- Board = [Who,_,_,Who,_,_,Who,_,_].

col_case_win(Board, Who) :- Board = [_,Who,_,_,Who,_,_,Who,_].

col_case_win(Board, Who) :- Board = [_,_,Who,_,_,Who,_,_,Who].


diag_case_win(Board, Who) :- Board = [Who,_,_,_,Who,_,_,_,Who].

diag_case_win(Board, Who) :- Board = [_,_,Who,_,Who,_,Who,_,_].


move([b,B,C,D,E,F,G,H,I], Who, [Who,B,C,D,E,F,G,H,I]).

move([A,b,C,D,E,F,G,H,I], Who, [A,Who,C,D,E,F,G,H,I]).

move([A,B,b,D,E,F,G,H,I], Who, [A,B,Who,D,E,F,G,H,I]).

move([A,B,C,b,E,F,G,H,I], Who, [A,B,C,Who,E,F,G,H,I]).

move([A,B,C,D,b,F,G,H,I], Who, [A,B,C,D,Who,F,G,H,I]).

move([A,B,C,D,E,b,G,H,I], Who, [A,B,C,D,E,Who,G,H,I]).

move([A,B,C,D,E,F,b,H,I], Who, [A,B,C,D,E,F,Who,H,I]).

move([A,B,C,D,E,F,G,b,I], Who, [A,B,C,D,E,F,G,Who,I]).

move([A,B,C,D,E,F,G,H,b], Who, [A,B,C,D,E,F,G,H,Who]).


x_can_win_in_one(Board) :- move(Board, x, Newboard), win(Newboard, x).
```

```
o_can_win_in_one(Board) :- move(Board, 'O', Newboard), win(Newboard, 'O').


x_response(Board,Newboard) :-
  move(Board, x, Newboard),
  win(Newboard, x),
  !.
x_response(Board,Newboard) :-
  move(Board, x, Newboard),
  not(o_can_win_in_one(Newboard)).
x_response(Board,Newboard) :-
  move(Board, x, Newboard).
x_response(Board,Newboard) :-
  not(member(b,Board)),
  !,
  open('output.txt',append,OS),
  write(OS,'Cats game!'),
  close(OS),
  Newboard = Board.


o_response(Board,Newboard) :-
  move(Board, 'O', Newboard),
  win(Newboard, 'O'),
  !.
o_response(Board,Newboard) :-
  move(Board, 'O', Newboard),
  not(x_can_win_in_one(Newboard)).
o_response(Board,Newboard) :-
```

```
  move(Board, '0', Newboard).
o_response(Board,Newboard) :-
  not(member(b,Board)),
  !,
  open('output.txt',append,OS),
  write(OS,'Cats Game!'),
  close(OS),
  Newboard = Board.

respond('0', Board, Newboard) :-
  o_response(Board, Newboard).
respond(x, Board, Newboard) :-
  x_response(Board, Newboard).

read_from_file(File) :-
    open('output.txt',write,OS),
    write(OS,''),
    close(OS),

    open(File, read, Stream),
   % nb_setval(last,'0'),
    nb_setval(mylist, [a,b,c,d,e,f,g,h,i]),
    nb_getval(mylist, Board),
    get_char(Stream, Char1),
    process_the_stream(Char1, Stream, Board),
    close(Stream).
```

```
process_the_stream(end_of_file, _, Board) :-
    replace(a,b, Board, R),
    replace(c,b, R, R1),
    replace(d,b, R1, R2),
    replace(e,b, R2, R3),
    replace(f,b, R3, R4),
    replace(g,b, R4, R5),
    replace(h,b, R5, R6),
    replace(i,b, R6, R7),

   %write(R7),
   %Last = 'O',
    nb_getval(last,Last),
    other(Last,Next),
    respond(Next, R7, Newboard),

   %write(Newboard),

    open('output.txt',write,OS),
    write(OS,Next),
    close(OS),
    get_first(R7, Newboard, [1,2,3,4,5,6,7,8,9]),
    !.

process_the_stream(Char, Stream, Board) :-
    get_char(Stream, Char1),
    nb_setval(last, Char1),
```

```
    get_char(Stream, _),

    get_char(Stream, Char3),

    replace(Char, Char1, Board, Rez),

    process_the_stream(Char3, Stream, Rez).


replace(_, _, [], []).

replace(O, R, [O|T], [R|T2]) :- replace(O, R, T, T2).

replace(O, R, [H|T], [H|T2]) :- H \= O, replace(O, R, T, T2).


other(x,'0').

other('0',x).


get_first([],[],[]) :- nb_setval(first,false), nb_getval(first,R),

                       open('output.txt',append,OS),

                       write(OS,R),

                       close(OS).

get_first([H|T],[H|T1],[_|It]) :- get_first(T,T1,It).

get_first([_|_],[_|_],[Ih|_]) :- nb_setval(first,Ih), nb_getval(first,R),

                       open('output.txt',append,OS),

                       write(OS,R),

                       close(OS).
```

```
iterate_list([]).

iterate_list([Head|Tail]):-
  write(Head),
  write(' '),
  iterate_list(Tail).

list_append(A,Tail,[A|Tail]).

myfunc() :-    list_append(a,[b,c,d,e,f,g,h,i],L),
    write(L),
    iterate_list(L).

func([]) :- nb_setval(mylist, [a,b,c,d,e,f,g,h,i]),
            nb_getval(mylist, CounterValue),write(CounterValue),
            replace(c,x,CounterValue,Rez),
            write(Rez),
            nb_setval(mylist, Rez),
            nb_getval(mylist, CounterValue),write(CounterValue),
            replace(a,x,CounterValue,Rez),
            write(Rez).
```