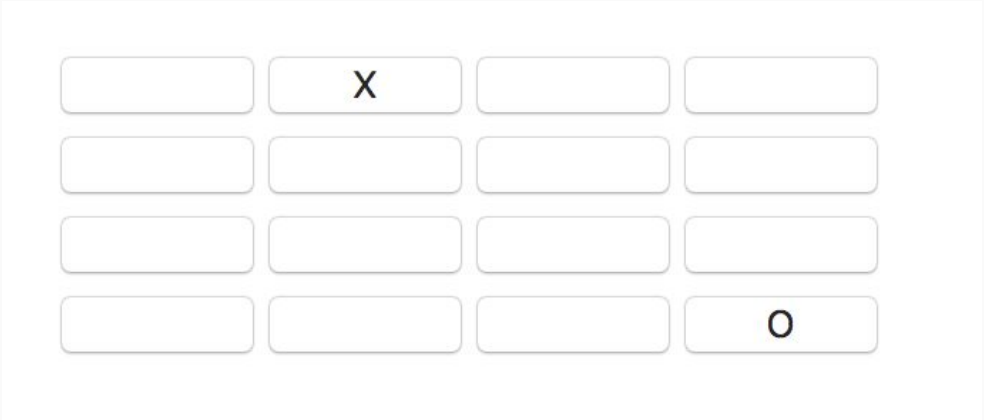
4 × 4 TIC TAC TOE AI

ARTIFICIAL INTELLIGENCE (CS 6613) PROJECT

AJAY THORVE
NYU ID: N13008057
Net ID: aat414

Introduction

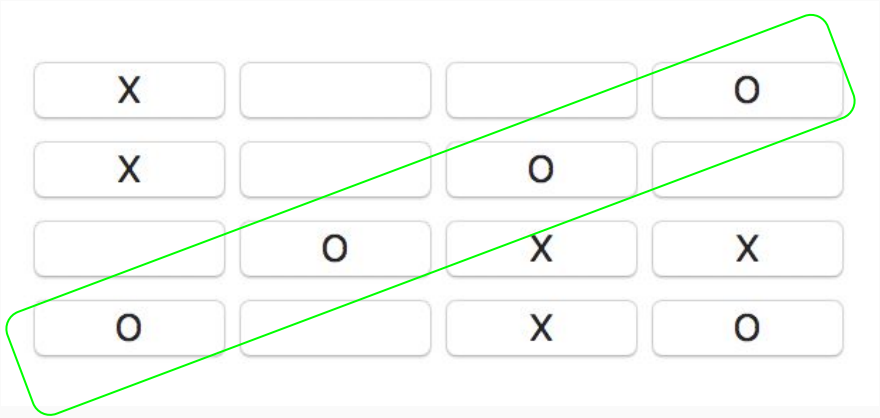
An interactive 4×4 Tic-Tac-Toe game for a person to play against a computer. The game consists of a 4×4 grid. To win, a player must place 4 of his/her symbols on 4 squares that line up vertically, horizontally or diagonally (45 or 135 degrees.)



About the game

The player and the AI get one turn each alternately.

Whoever gets four X’s or O’s in a horizontal, vertical or diagonal line, first, wins.



One of the possible winning conditions

Implementation

The source code of the game consists of 3 files.

Eval.py

index.py

tictactoeai.py

1. Index.py

The core file, which has the homepage GUI and instantiates the class written in Tictactoeai.py

2. Tictactoeai.py

Contains class “Game”, of which numerous instants can be created simultaneously. Contains the Alpha-Beta Algorithm Code.

3. Eval.py

All Evaluation functions: Easy, Medium, Difficult and a custom one

Game Flow - 1

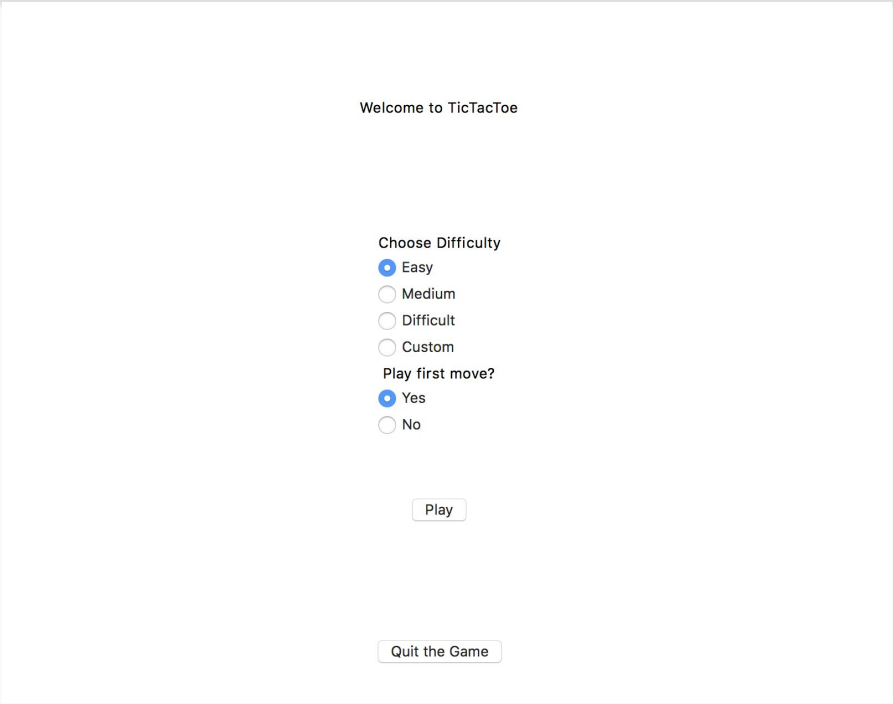
Home Screen:

User gets to select difficulty:

Easy/ Medium/ Difficult

And First move:

User/AI



Game Flow - 2

When Clicked on Play:

Click on any of the 16 boxes when it is your move.

By default, user player has been assigned “O” symbol, and AI has been assigned “X” symbol

Algorithm Used - Minimax

```
def AlphaBetaSearch(self,board,t0,depth):
    v=self.MaxAction(board,-1000,1000,CurrentLevel,depth,t0,bestmove)
    for position,value in bestmove.items():
      if value==v:
        print (position,value)
        return v,str(position)
```

```
def MaxAction(self,board,alpha,beta,i,depth,t0,bestmove):
    if self.CheckGameStatus(board)==False or t>self.max_time or i>=(depth+self.CutOffDepth):
      return self.CutOffSearch(board[:])
    v=-10000
    indexi,indexj=self.Actions(board)
    for x in range(len(indexi)):
            v_min=self.MinAction(self.ResultX(board[:],indexi[x],indexj[x]),alpha,beta,i+1,depth,t0,b
estmove)
            v=max(v,v_min)
             if v>=beta:
                self.MaxPruning+=1
                return v
             alpha=max(alpha,v)
    return v
```

```
def MinAction(self,board,alpha,beta,i,depth,t0,bestmove):
    if self.CheckGameStatus(board)==False or t>self.max_time or i>=(depth+self.CutOffDepth):
      return self.CutOffSearch(board[:])
    v=10000
    indexi,indexj=self.Actions(board)
    for x in range(len(indexi)):
            v_max=self.MaxAction(self.ResultX(board[:],indexi[x],indexj[x]),alpha,beta,i+1,depth,t0,
bestmove)
            v=min(v,v_max)
             if v<=alpha:
                self.MinPruning+=1
                return v
             beta=min(beta,v)
    return v
```

Levels - 1

1. Easy

Random Function used which accumulates the possible actions for the AI currently, and randomly selects a position to write “X”.

2. Medium

Evaluation function used : *o[2]-o[3]* ,

o[i] : “i” number of number of O’s vertically, horizontally and diagonally, without any X’s

Max Depth Cut off limit : 7, Max time for a move allowed: 1 second

Levels - 2

3. Difficult

Evaluation function used : *6*x[3]+3*x[2]+x[1]-6*o[3]-3*o[2]-o[1]* ,

o[i] : “i” number of number of O’s vertically, horizontally and diagonally, without any X’s
x[i] : “i” number of number of X’s vertically, horizontally and diagonally, without any O’s

Max Depth Cut off limit : 6, Max time for a move allowed: 10 seconds

Statistics of AI’s moves

The required statistics of each of the AI’s moves is displayed dynamically on the game window.

Total number of nodes generated: 22170
Cutoff occured
Maximum Depth reached: 7
Number of times Pruning occured in MAX Function: 2896
Number of times Pruning occured in MIN Function: 539