Solving Tic Tac Toe with AI: Minimax Algorithm

CSCI6649 – GAME AI PROJECT

PARISHAD AJALLOOEIAN & JIGNESHKUMAR MAKAWANA

DATE: 2025

Introduction

- Goal: Build a Tic Tac Toe game with AI using Minimax.
- Tools Used: Python 3, Console interface.
- AI Concept: Minimax Algorithm.

How Tic Tac Toe Works

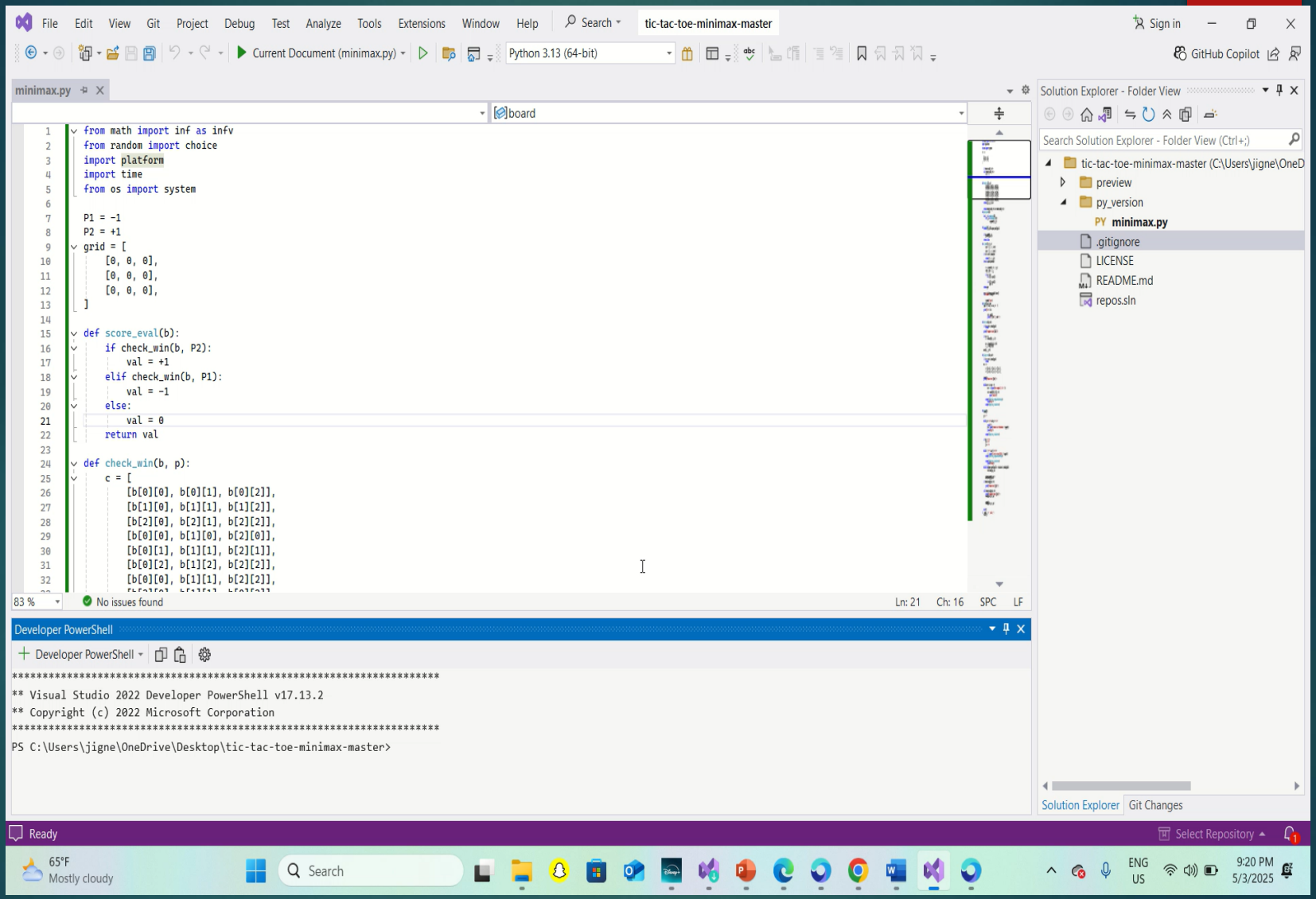
- 3x3 grid, two players (X and O).
- Three in a row wins: horizontally, vertically, or diagonally.
- Game ends in win or draw.

What is the Minimax Algorithm?

- Decision-making algorithm for two-player games.
- Simulates all possible moves and outcomes.
- AI assumes opponent will play optimally.
- Returns the move with best guaranteed result.

Demo Walkthrough

- Parishad: Game rules, main() function, human input.
- Jigneshkumar: Minimax logic, score explanation.
- Live demo of human and AI turns.


File Edit View Git Project Debug Test Analyze Tools Extensions Window Help
Search
tic-tac-toe-minimax-master
Sign in
Current Document (minimax.py)
Python 3.13 (64-bit)
GitHub Copilot
minimax.py
board
from math import inf as infv
from random import choice
import platform
import time
from os import system
P1 = -1
P2 = +1
grid = [
[0, 0, 0],
[0, 0, 0],
[0, 0, 0],
]
def score_eval(b):
if check_win(b, P2):
val = +1
elif check_win(b, P1):
val = -1
else:
val = 0
return val
def check_win(b, p):
c = [
[b[0][0], b[0][1], b[0][2]],
[b[1][0], b[1][1], b[1][2]],
[b[2][0], b[2][1], b[2][2]],
[b[0][0], b[1][0], b[2][0]],
[b[0][1], b[1][1], b[2][1]],
[b[0][2], b[1][2], b[2][2]],
[b[0][0], b[1][1], b[2][2]],
83 %
No issues found
Ln: 21
Ch: 16
SPC
LF
Solution Explorer - Folder View
Search Solution Explorer - Folder View (Ctrl+;)
tic-tac-toe-minimax-master (C:\Users\jigne\OneD
preview
py_version
minimax.py
.gitignore
LICENSE
README.md
repos.sln
Solution Explorer
Git Changes
Developer PowerShell
Developer PowerShell
****
** Visual Studio 2022 Developer PowerShell v17.13.2
** Copyright (c) 2022 Microsoft Corporation
****
PS C:\Users\jigne\OneDrive\Desktop\tic-tac-toe-minimax-master>
Ready
Select Repository
65°F
Mostly cloudy
Search
ENG
US
9:20 PM
5/3/2025

Implementation Structure

- ai_move() : Implements the Minimax algorithm recursively.
- score_eval() : Evaluates the board for win/loss/draw (+1, 0, -1).
- check_win(), open_spots(), mark(): Supporting game logic functions.
- start(): Main game loop – manages player input and bot turns.
- bot_turn() & player_turn(): Control computer and human moves.

- ai_move()

```
57  def ai_move(b, d, p):
58      if p == P2:
59          opt = [-1, -1, -infv]
60      else:
61          opt = [-1, -1, +infv]
62
63      if d == 0 or is_done(b):
64          sc = score_eval(b)
65          return [-1, -1, sc]
66
67      for m in open_spots(b):
68          i, j = m[0], m[1]
69          b[i][j] = p
70          s = ai_move(b, d - 1, -p)
71          b[i][j] = 0
72          s[0], s[1] = i, j
73
74          if p == P2:
75              if s[2] > opt[2]:
76                  opt = s
77          else:
78              if s[2] < opt[2]:
79                  opt = s
80
81      return opt
```

```
14
15  def score_eval(b):
16      if check_win(b, P2):
17          val = +1
18      elif check_win(b, P1):
19          val = -1
20      else:
21          val = 0
22      return val
23
24  def check_win(b, p):
25      c = [
26          [b[0][0], b[0][1], b[0][2]],
27          [b[1][0], b[1][1], b[1][2]],
28          [b[2][0], b[2][1], b[2][2]],
29          [b[0][0], b[1][0], b[2][0]],
30          [b[0][1], b[1][1], b[2][1]],
31          [b[0][2], b[1][2], b[2][2]],
32          [b[0][0], b[1][1], b[2][2]],
33          [b[2][0], b[1][1], b[0][2]],
34      ]
35      return [p, p, p] in c
36
```

Scoring System

Supporting Functions

```
35          return [p, p, p] in c
36
37      def is_done(b):
38          return check_win(b, P1) or check_win(b, P2)
39
40      def open_spots(b):
41          s = []
42          for i, r in enumerate(b):
43              for j, e in enumerate(r):
44                  if e == 0:
45                      s.append([i, j])
46          return s
47
48      def is_valid(i, j):
49          return [i, j] in open_spots(grid)
50
51      def mark(i, j, p):
52          if is_valid(i, j):
53              grid[i][j] = p
54              return True
55          return False
56
```

```
100  def bot_turn(m1, m2):
101      d = len(open_spots(grid))
102      if d == 0 or is_done(grid):
103          return
104      reset()
105      print(f'Computer turn [{m1}]')
106      show(grid, m1, m2)
107
108      if d == 9:
109          i = choice([0, 1, 2])
110          j = choice([0, 1, 2])
111      else:
112          m = ai_move(grid, d, P2)
113          i, j = m[0], m[1]
114
115      mark(i, j, P2)
116      time.sleep(1)
```

```
def player_turn(m1, m2):
    d = len(open_spots(grid))
    if d == 0 or is_done(grid):
        return

    mv = -1
    mvs = {
        1: [0, 0], 2: [0, 1], 3: [0, 2],
        4: [1, 0], 5: [1, 1], 6: [1, 2],
        7: [2, 0], 8: [2, 1], 9: [2, 2],
    }

    reset()
    print(f'Human turn [{m2}]')
    show(grid, m1, m2)

    while mv < 1 or mv > 9:
        try:
            mv = int(input('Use numpad (1..9): '))
            c = mvs[mv]
            ok = mark(c[0], c[1], P1)
            if not ok:
                print('Invalid')
                mv = -1
        except (EOFError, KeyboardInterrupt):
            print('Bye')
            exit()
        except (KeyError, ValueError):
            print('Wrong')
```

```
148 def start():
149     reset()
150     p2 = ''
151     p1 = ''
152     f = ''
153 
154     while p1 != 'O' and p1 != 'X':
155         try:
156             print('')
157             p1 = input('Choose X or O\nChosen: ').upper()
158         except (EOFError, KeyboardInterrupt):
159             print('Bye')
160             exit()
161         except (KeyError, ValueError):
162             print('Wrong')
163 
164     if p1 == 'X':
165         p2 = 'O'
166     else:
167         p2 = 'X'
168 
169     reset()
170     while f != 'Y' and f != 'N':
171         try:
172             f = input('First to start?[y/n]: ').upper()
173         except (EOFError, KeyboardInterrupt):
174             print('Bye')
175             exit()
176         except (KeyError, ValueError):
177             print('Wrong')
178 
179     while len(open_spots(grid)) > 0 and not is_done(grid):
180         if f == 'N':
181             bot_turn(p2, p1)
182             f = ''
183 
184         player_turn(p2, p1)
185         bot_turn(p2, p1)
186 
```

```
185         bot_turn(p2, p1)
186 
187     if check_win(grid, P1):
188         reset()
189         print(f'Human turn [{p1}]')
190         show(grid, p2, p1)
191         print('YOU WIN!')
192     elif check_win(grid, P2):
193         reset()
194         print(f'Computer turn [{p2}]')
195         show(grid, p2, p1)
196         print('YOU LOSE!')
197     else:
198         reset()
199         show(grid, p2, p1)
200         print('DRAW!')
201 
202     exit()
203 
204 if __name__ == '__main__':
205     start()
206 
```

Challenges Faced

- Input validation and exception handling.
- Balancing AI difficulty.
- Ensuring clean game restart and flow.

Final Thoughts

- Learned game trees and recursion.
- Improved Python and algorithmic skills.
- Future: Add GUI (Tkinter or Pygame), difficulty levels.

Thank You / Q&A

- Thanks for watching!
- Questions or feedback welcome.
- Team: Parishad Ajallooeian & Jigneshkumar Makawana