RV College
of Engineering®
Autonomous Institution Affiliated to Visvesvaraya Technological University, Belagavi
Approved by AICTE, New Delhi, Accredited by NAAC, Bengaluru

Department of Electronics and Telecommunication Engineering
Ph: 080-67178040, 8041
e-mail: hod.tc@rvce.edu.in

18TE7F3-Deep Learning and Artificial Intelligence

Experiential Learning Report

On

"AI-Based Tic-Tac-Toe Game"

Submitted by

Name of the Student	**USN of the student**
Ch Mani Pujith	*1RV20ET015*

Under the guidance of

DR Ranjani G
Assistant Professor
Dept. of Electronics & Telecommunication Engineering
RV College of Engineering
Bengaluru

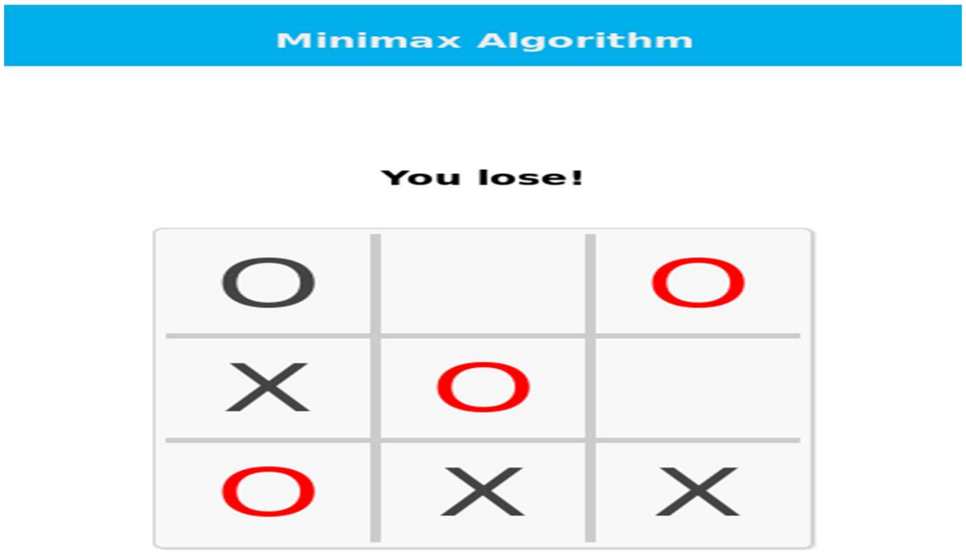
Contents

Introduction

- AI is a transformative force in various fields, including Playing Games.
- Artificial intelligence (AI) has been used in video games since the 1950s.
- No more predictable foes! AI creates cunning opponents that adapt to your strategies, making battles more thrilling.
- AI tailors the game to you, adjusting difficulty, generating unique quests, and crafting stories that resonate.
- AI-powered anti-cheat systems keep online competition fair and enjoyable for everyone.
- AI is constantly evolving, promising even more mind-blowing experiences in games to come!
- AI tailors the game difficulty, narrative elements, and even content based on individual player preferences and skill levels, making the experience truly personal.
- AI allows for branching narratives, dynamic dialogue, and adaptive responses, weaving immersive and personalized stories that react to player choices.
- AI continues to push boundaries, paving the way for truly intelligent companions, emotionally engaging characters, and even games that learn and adapt like living entities.

Understanding the Algorithm

- The algorithm was studied by the book Algorithms in a Nutshell (George Heineman; Gary Pollice; Stanley Selkow, 2009). Pseudocode (adapted):



- The minimax algorithm is a decision-making algorithm used in artificial intelligence (AI) for two-player zero-sum games. It is used to determine the best possible move for a player, assuming that the opponent is also playing optimally.
- The minimax algorithm works by recursively exploring all possible game states from the current state, and assigning a score to each state. The score represents the expected outcome for the player who is to move in that state. For the player who wants to maximize their score (the "max" player), the algorithm chooses the move that leads to the state with the highest score. For the player who wants to minimize their opponent's score (the "min" player), the algorithm chooses the move that leads to the state with the lowest score.
- The minimax algorithm can be a very powerful tool for AI game playing, but it can also be computationally expensive, especially for games with large state spaces. In practice, it is often necessary

Objectives :

- The primary goal is to create an AI that consistently defeats human opponents. This requires mastery of basic Tic Tac Toe strategy and efficient move evaluation.
- **Learn and adapt:** Go beyond static algorithms and develop AIs that learn from experience. They should adapt their strategies based on past games and opponent behavior.
- **Explore beyond single games:** Consider more complex Tic Tac Toe variations or introduce new mechanics. The AI should handle diverse scenarios and maintain its winning edge.
- **Optimize play speed:** In real-time games, fast decision-making is crucial. Develop efficient algorithms that analyze potential moves quickly without sacrificing quality.
- **Generalizable AI principles**: Use Tic Tac Toe as a training ground for AI principles applicable to more complex games. Develop AIs that learn, plan, and make strategic decisions effectively.
- **Procedural Content Generation:** Automatically create new levels, items, quests, and storylines, offering endless replayability and freshness.
- **Improved Game Design and Development:** Streamline workflows by automating repetitive tasks, allowing developers to focus on creative aspects.
- **Enhanced Learning and Education:** Utilize game-based learning powered by AI to create more engaging and effective educational experiences.

```
minimax(state, depth, player)
    if (player = max) then
        best = [null, -infinity]
    else
        best = [null, +infinity]
    if (depth = 0 or gameover) then
        score = evaluate this state for player
        return [null, score]
    for each valid move m for player in state s do
        execute move m on s
        [move, score] = minimax(s, depth - 1, -player)
        undo move m on s
        if (player = max) then
            if score > best.score then best = [move, score]
        else
            if score < best.score then best = [move, score]
    return best
end
```

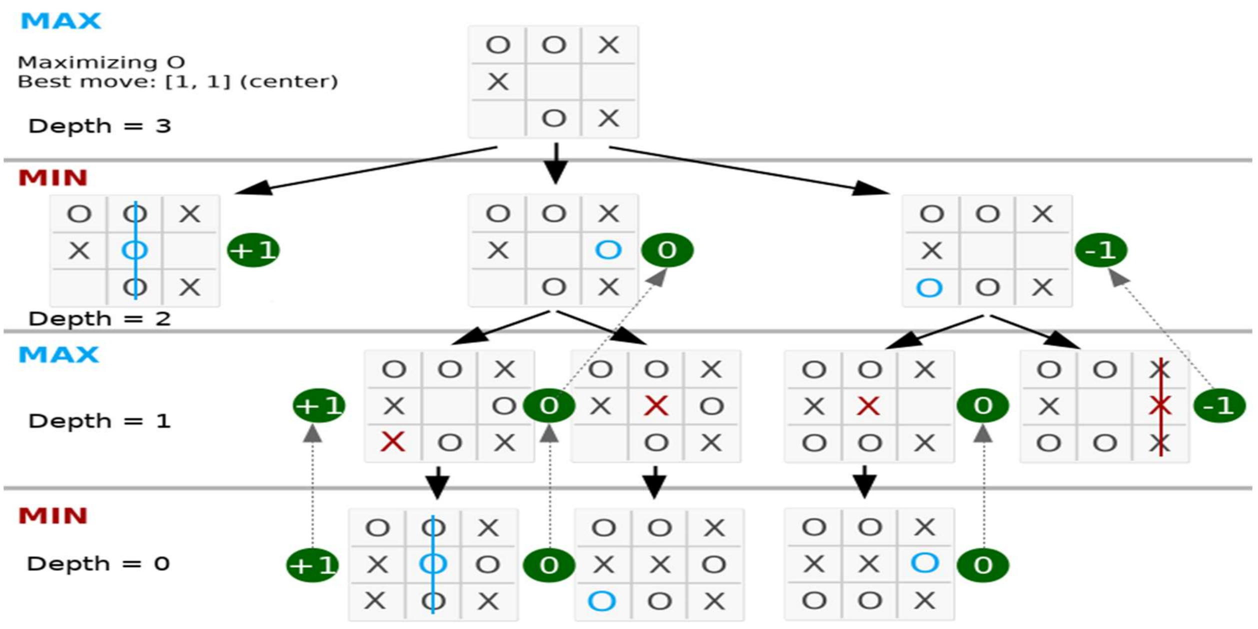
The code defines a function minimax that takes three arguments:

- state: The current state of the game.
- depth: The maximum depth of the search tree.
- player: The player whose turn it is to move.

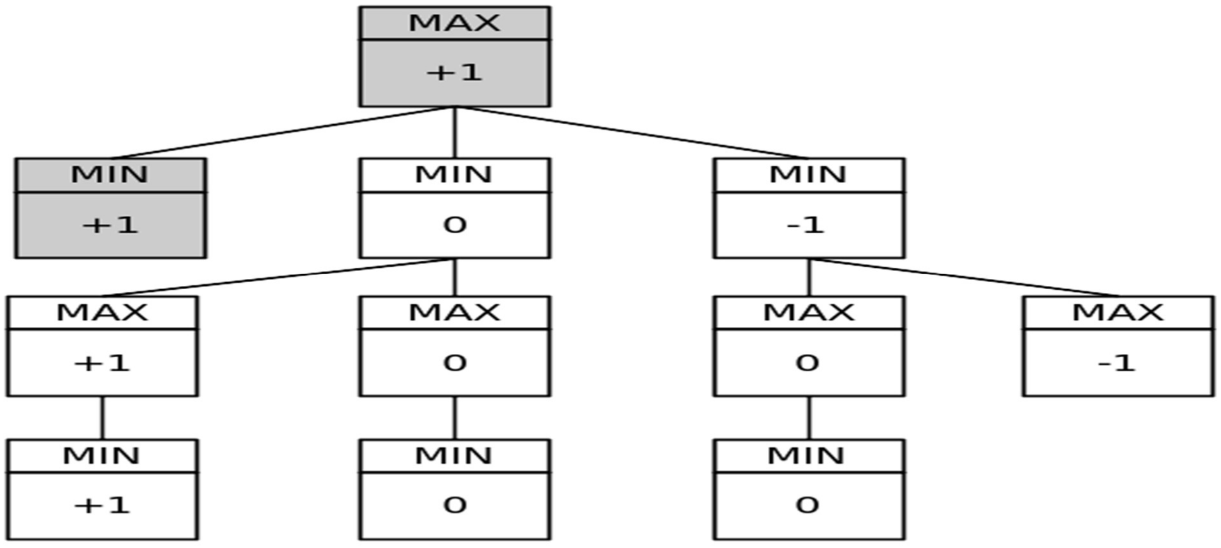
The minimax function works as follows:

- If the depth is 0 or the game is over, the function returns the score of the current state for the player whose turn it is to move.
- Otherwise, the function generates all possible moves for the player whose turn it is to move.
- For each move, the function recursively calls itself on the resulting state, with the depth decremented by 1 and the player switched.
- If the player whose turn it is to move is the "max" player, the function keeps track of the move with the highest score.
- If the player whose turn it is to move is the "min" player, the function keeps track of the move with the lowest score.
- Finally, the function returns the move with the best score for the player whose turn it is to move.

Methodology



Take a look that the depth is equal the valid moves on the board.



That tree has 11 nodes. The full game tree has 549.946 nodes! You can test it putting a static global variable in your program and incrementing it for each minimax function call per turn.

In a more complex game, such as chess, it's hard to search whole game tree. However, Alpha–beta Pruning is an optimization method to the minimax algorithm that allows us to disregard some branches in the search tree, because he cuts irrelevant nodes (subtrees) in search.

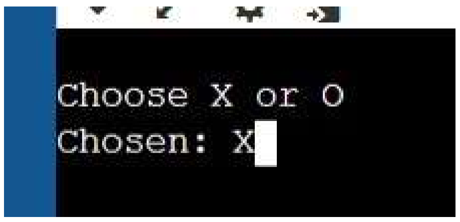
CODE

```
- from math import inf as infinity

from random import choice

import platform

import time

from os import system

"""

An implementation of Minimax AI Algorithm in Tic Tac Toe,

using Python.

This software is available under GPL license.

Author: Clederson Cruz

Year: 2017

License: GNU GENERAL PUBLIC LICENSE (GPL)

"""
```

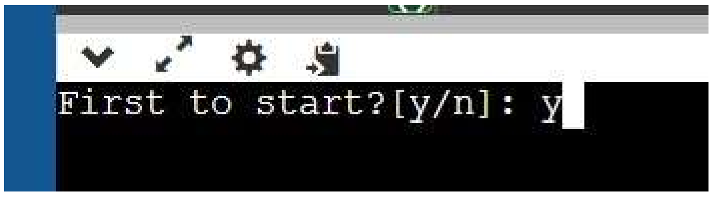
```
HUMAN = -1

COMP = +1

board = [

    [0, 0, 0],

    [0, 0, 0],

    [0, 0, 0], } ]

def evaluate(state):

    """

    Function to heuristic evaluation of state.

    :param state: the state of the current board

    :return: +1 if the computer wins; -1 if the human wins; 0 draw

    """

     - if wins(state, COMP):

        score = +1

    elif wins(state, HUMAN):

        score = -1

    else:

        score = 0

    return score

def wins(state, player):

    This function tests if a specific player wins. Possibilities:

    * Three rows    [X X X] or [O O O]

    * Three cols    [X X X] or [O O O]
    * Two diagonals [X X X] or [O O O]
    :param state: the state of the current board
    :param player: a human or a computer
    :return: True if the player wins
from math import inf as infinity
from random import choice
import platform
import time
from os import system
"""
An implementation of Minimax AI Algorithm in Tic Tac Toe,
using Python.
This software is available under GPL license.
Author: Clederson Cruz
Year: 2017
License: GNU GENERAL PUBLIC LICENSE (GPL)
"""
HUMAN = -1
COMP = +1
board = [
    [0, 0, 0],
    [0, 0, 0],
    [0, 0, 0], } ]
```

```
def evaluate(state):
    """
    Function to heuristic evaluation of state.
    :param state: the state of the current board
    :return: +1 if the computer wins; -1 if the human wins; 0 draw
    """
if wins(state, COMP):
        score = +1
    elif wins(state, HUMAN):
        score = -1
    else:
        score = 0
    return score
def wins(state, player):
    This function tests if a specific player wins. Possibilities:
    * Three rows    [X X X] or [O O O]
    * Three cols    [X X X] or [O O O]
    * Two diagonals [X X X] or [O O O]
    :param state: the state of the current board
    :param player: a human or a computer
    :return: True if the player wins
    if h_choice == 'X':
        c_choice = 'O'
    else:
        c_choice = 'X'
    # Human may starts first
    clean()
    while first != 'Y' and first != 'N':
        try:
            first = input('First to start?[y/n]: ').upper()
        except (EOFError, KeyboardInterrupt):
            print('Bye')
            exit()
        except (KeyError, ValueError):
            print('Bad choice')
    # Main loop of this game
    while len(empty_cells(board)) > 0 and not game_over(board):
        if first == 'N':
            ai_turn(c_choice, h_choice)
            first = ''
  human_turn(c_choice, h_choice)
        ai_turn(c_choice, h_choice)
    # Game over message
    if wins(board, HUMAN):
        clean()
        print(f'Human turn [{h_choice}]')
        render(board, c_choice, h_choice)
        print('YOU WIN!')
    elif wins(board, COMP):
        clean()
        print(f'Computer turn [{c_choice}]')
        render(board, c_choice, h_choice)
        print('YOU LOSE!')
    else:
        clean()
        render(board, c_choice, h_choice)
        print('DRAW!')
    exit()
```

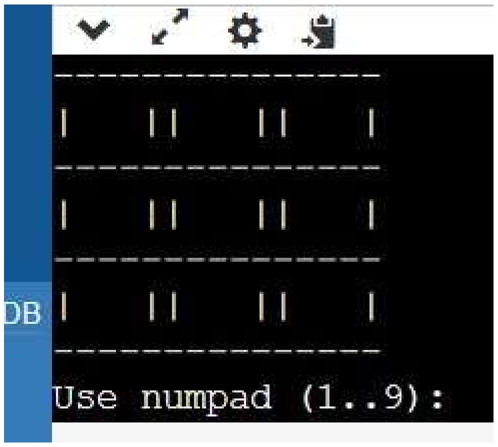
Stimulation Results :



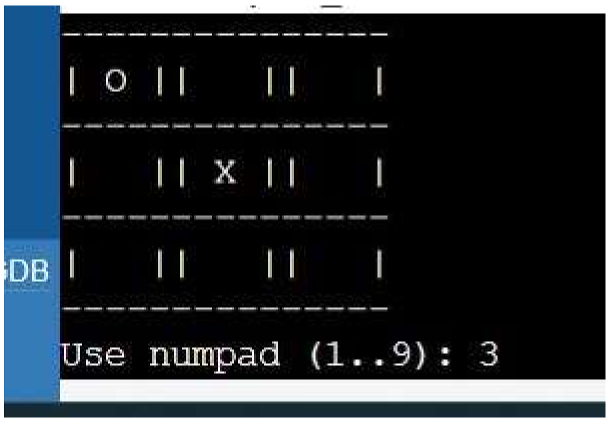
```
Choose X or O
Chosen: X
```



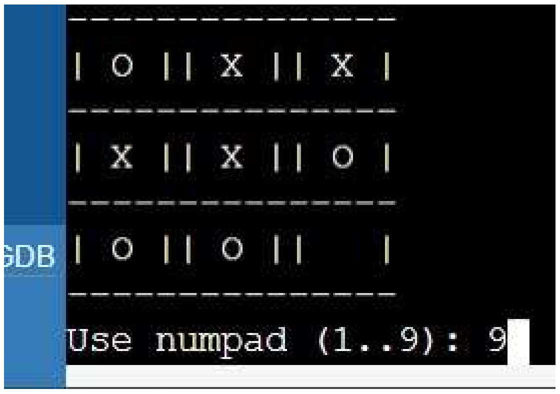
```
First to start?[y/n]: y
```



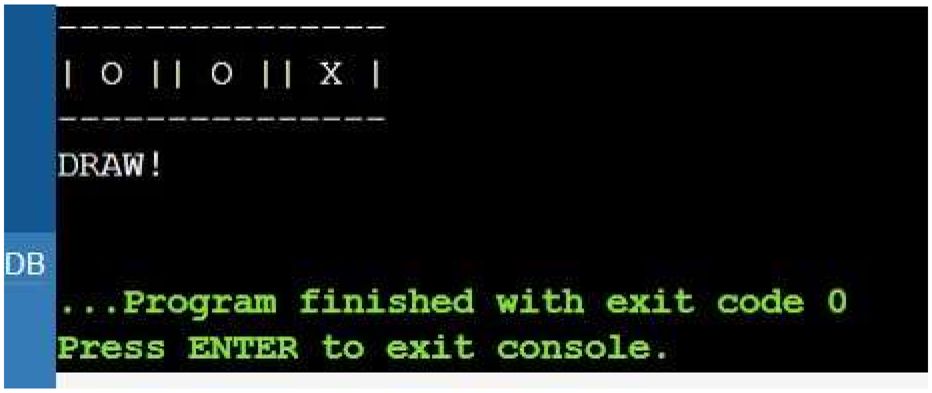
```
---------------
|   ||   ||   |
---------------
|   ||   ||   |
---------------
|   ||   ||   |
---------------
Use numpad (1..9):
```



```
---------------
| O ||   ||   |
---------------
|   || X ||   |
---------------
|   ||   ||   |
---------------
Use numpad (1..9): 3
```



```
---------------
| O || X || X |
---------------
| X || X || O |
---------------
| O || O ||   |
---------------
Use numpad (1..9): 9
```



```
---------------
| O || O || X |
---------------
DRAW!


...Program finished with exit code 0
Press ENTER to exit console.
```

Result :

Tik- tak -toe implemented perfectly, Minimax guarantees an optimal move for Max, making it unbeatable against non-optimal opponents.

In Tic-Tac-Toe, with a limited number of states, perfect play leads to draws unless one player makes a mistake.

Minimax performance depends on the depth of exploration (number of future moves considered). Increasing depth improves accuracy but increases computation time.