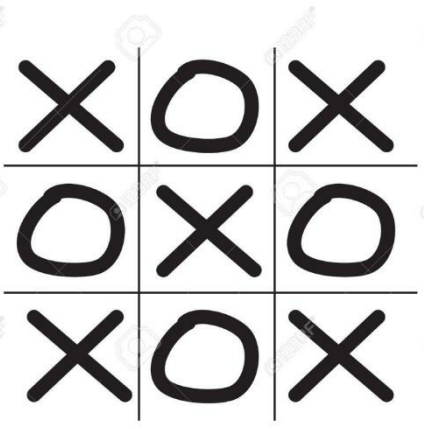
औद्योगिक प्रशिक्षण के लिए राष्ट्रीय संस्थान
National Institute for Industrial Training
One Premier Organization with Non Profit Status | Registered Under Govt. of WB
Empanelled Under Planning Commission Govt. of India
Inspired By: National Task Force on IT & SD Government of India

National Institute for Industrial Training- One Premier Organization with Non Profit Status Registered Under Govt.of West Bengal,Empanelled Under Planning Commission Govt.of India , Empanelled Under Central Social Welfare Board Govt. of India , Registered with National Career Services , Registered with National Employment Services.

THE UNBEATABLE TIC-TAC-TOE GAME USING PYTHON AND ITS APPLICATIONS IN AI

Name - Arpan Goswami
Subject - Python with Artificial Intelligence



The unbeatable

A minor project presented ny
1) **Arpan Goswami**
2) **Rounak Saha**
3) **Animesh Arya**
4) **Shailesh Das**
5) **Ayush Kumar**
6) **Ritik Verma**

Under the guidance of our guide
Mr.Arka Patra

Preface

This project is based on the application of artificial intelligence to develop an unbeatable AI which either draws or wins and always opts for the most optimum move in a game of tic tac toe.

The source code is reliant on the principles of minimax and negamax algorithms or the zero-sum principle in which other games such as connecting dots or our own domain tic tac toe.

Minimax algorithm is one such strategy used by combinatorial search. When two players are playing against each other, they are basically working towards opposite goals. So each side needs to predict what the opposing player is going to do in order to win the game. Keeping this in mind, Minimax tries to achieve this through strategy. It will try to minimize the function that the opponent is trying to maximize.

Introduction to Python

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Van Rossum led the language community until stepping down as leader in July 2018.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the Lisp tradition. It has `filter()`, `map()`, and `reduce()` functions; list comprehensions, dictionaries, and sets; and generatorexpressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

Why use Python?

Python is a high-level, interpreted and general-purpose dynamic programming language that focuses on code readability. The syntax in Python helps the programmers to do coding in fewer steps as compared to Java or C++. The language founded in the year 1991 by the developer Guido Van Rossum has the programming easy and fun to do. The Python is widely used in bigger organizations because of its multiple programming paradigms. They usually involve imperative and object-oriented functional programming. It has a comprehensive and large standard library that has automatic memory management and dynamic features.

Why Companies Prefer Python?

Python has top the charts in the recent years over other programming languages like C, C++ and Java and is widely used by the programmers. The language has undergone a drastic change since its release 25 years ago as many add-on features are introduced. The Python 1.0 had the module system of Modula-3 and interacted with Amoeba Operating System with varied functioning tools. Python 2.0 introduced in the year 2000 had features of garbage collector and Unicode Support. Python 3.0 introduced in the year 2008 had a constructive design that avoids duplicate modules and constructs. With the added features, now the companies are using Python 3.5.

The software development companies prefer Python language because of its versatile features and fewer programming codes. Nearly 14% of the programmers use it on the operating systems like UNIX, Linux, Windows and Mac OS. The programmers of big companies use Python as it has created a mark for itself in the software development with characteristic features like-

- Interactive
- Interpreted
- Modular
- Dynamic
- Object-oriented
- Portable

- High level
- Extensible in C++ & C

Advantages or Benefits of Python

The Python language has diversified application in the software development companies such as in gaming, web frameworks and applications, language development, prototyping, graphic design applications, etc. This provides the language a higher plethora over other programming languages used in the industry. Some of its advantages are-

- **Extensive Support Libraries**

It provides large standard libraries that include the areas like string operations, Internet, web service tools, operating system interfaces and protocols. Most of the highly used programming tasks are already scripted into it that limits the length of the codes to be written in Python.

- **Integration Feature**

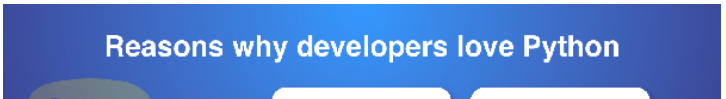
Python integrates the Enterprise Application Integration that makes it easy to develop Web services by invoking COM or COBRA components. It has powerful control capabilities as it calls directly through C, C++ or Java via Jython. Python also processes XML and other markup languages as it can run on all modern operating systems through same byte code.

- **Improved Programmer's Productivity**

The language has extensive support libraries and clean object-oriented designs that increase two to ten fold of programmer's productivity while using the languages like Java, VB, Perl, C, C++ and C#.

- **Productivity**

With its strong process integration features, unit testing framework and enhanced control capabilities contribute towards the increased speed for most applications and productivity of applications. It is a great option for building scalable multi-protocol network applications.



Why is Python is best for

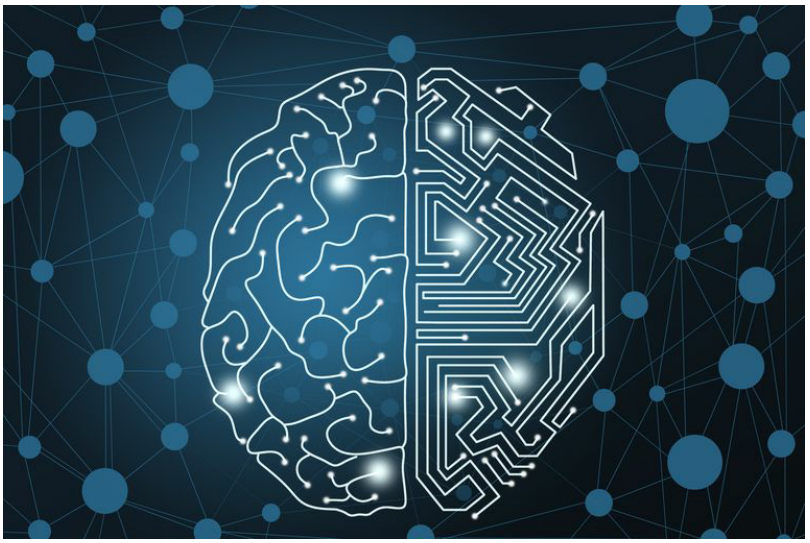
AI and Python: Why?

The obvious question that we need to encounter at this point is why we should choose Python for AI over others.

Python offers the least code among others and is in fact 1/5 the number compared to other OOP languages. No wonder it is one of the most popular in the market today.

- Python has Prebuilt Libraries like Numpy for scientific computation, Scipy for advanced computing and Pybrain for machine learning (Python Machine Learning) making it one of the best languages For AI.
- Python developers around the world provide comprehensive support and assistance via forums and tutorials making the job of the coder easier than any other popular languages.
- Python is platform Independent and is hence one of the most flexible and popular choiceS for use across different platforms and technologies with the least tweaks in basic coding.

- Python is the most flexible of all others with options to choose between OOPs approach and scripting. You can also use IDE itself to check for most codes and is a boon for developers struggling with different algorithms.



Artificial Intelligence

What is Artificial Intelligence?

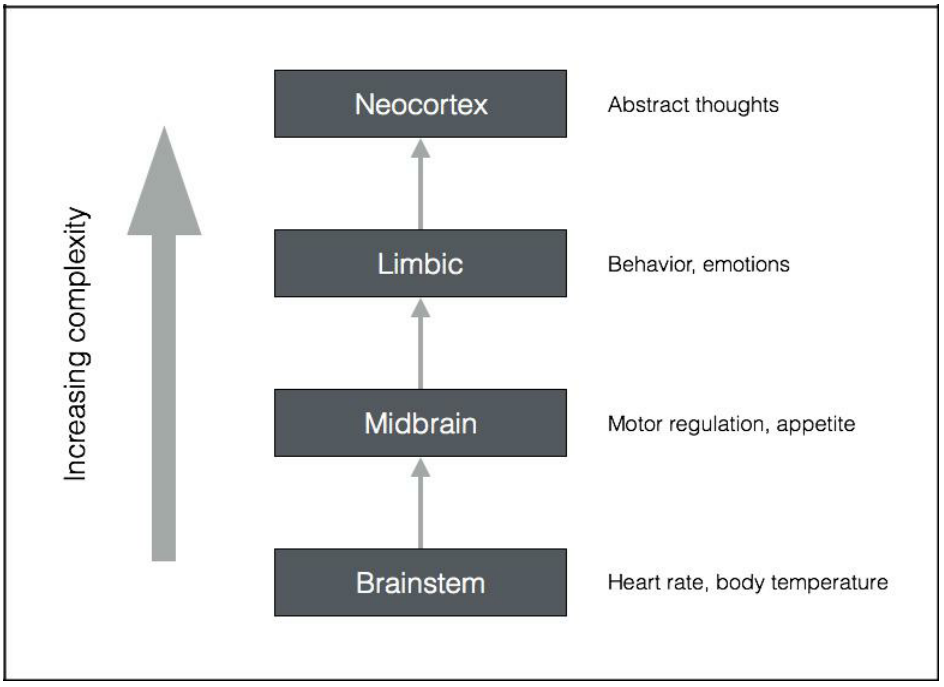
Artificial Intelligence (**AI**) is a way to make machines think and behave intelligently. These machines are controlled by software inside them, so AI has a lot to do with intelligent software programs that control these machines. It is a science of finding theories and methodologies that can help machines understand the world and accordingly react to situations in the same way that humans do.

If we look closely at how the field of AI has emerged over the last couple of decades, you will see that different researchers tend to focus on different concepts to define AI. In the modern world, AI is used across many verticals in many different forms. We want the machines to sense, reason, think, and act. We want our machines to be rational too.

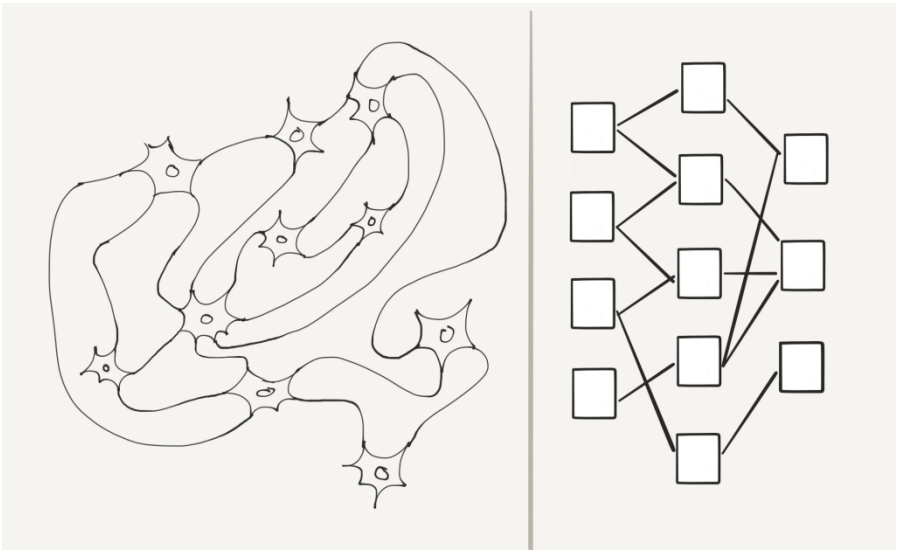
AI is closely related to the study of human brain. Researchers believe that AI can be accomplished by understanding how the human brain works. By mimicking the way the human brain learns, thinks, and takes action, we can build a machine that can do the same. This can be used as a platform to develop intelligent systems that are capable of learning.

Why do we need to study AI?

AI has the ability to impact every aspect of our lives. The field of AI tries to understand patterns and behaviors of entities. With AI, we want to build smart systems and understand the concept of intelligence as well. The intelligent systems that we construct are very useful in understanding how an intelligent system like our brain goes about constructing another intelligent system.Let's take a look at how our brain processes information:



Compared to some other fields such as Mathematics or Physics that have been around for centuries, AI is relatively in its infancy. Over the last couple of decades, AI has produced some spectacular products such as self-driving cars and intelligent robots that can walk.
Based on the direction in which we are heading, it's pretty obvious that achieving intelligence will have a great impact on our lives in the coming years.



Using search algorithms in games

Search algorithms are used in games to figure out a strategy. The algorithms search through the possibilities and pick the best move. There are various parameters to think about – speed, accuracy, complexity, and so on. These algorithms consider all possible actions available at this time and then evaluate their future moves based on these options. The goal of these algorithms is to find the optimal set of moves that will help them arrive at the final condition. Every game has a different set of winning conditions. These algorithms use those conditions to find the set of moves.

The description given in the previous paragraph is ideal if there is no opposing player. Things are not as straightforward with games that have multiple players. Let's consider a two-player game. For every move made by a player, the opposing player will make a move to prevent the player from achieving the goal. So when a search algorithm finds the optimal set of moves from the current state, it cannot just go ahead and make those moves because the opposing player will stop it. This basically means that search algorithms need to constantly re-evaluate after each move.

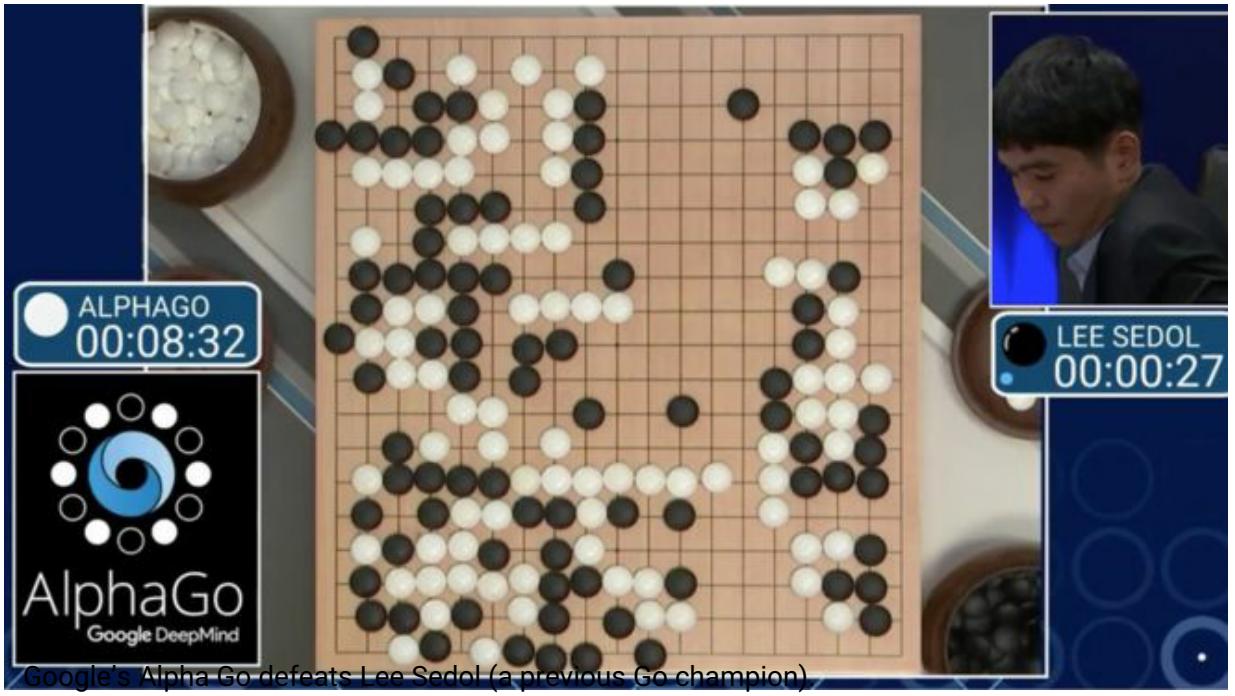
Let's discuss how a computer perceives any given game. We can think of a game as a search tree. Each node in this tree represents a future state. For example, if you are playing **Tic–Tac–Toe** (Noughts and Crosses), you can construct this tree to represent all possible moves. We start from the root of the tree, which is the starting point of the game. This node will have several children that represent various possible moves. Those children, in turn, will have more children that represent game states after more moves by the opponent. The terminal nodes of the tree represent the final results of the game after making various moves. The game would either end in a draw or one of the players would win it. The search algorithms search through this tree to make decisions at each step of the game.

Combinatorial search

Search algorithms appear to solve the problem of adding intelligence to games, but there's a drawback. These algorithms employ a type of search called exhaustive search, which is also known as brute force search. It basically explores the entire search space and tests every possible solution. It means that, in the worst case, we will have to explore all the possible solutions before we get the right solution.

As the games get more complex, we cannot rely on brute force search because the number of possibilities gets enormous. This quickly becomes computationally intractable. In order to solve this problem, we use combinatorial search to solve problems. It refers to a field of study where search algorithms efficiently explore the solution space using heuristics or by reducing the size of the search space. This is very useful in games like Chess or Go.

Combinatorial search works efficiently by using pruning strategies. These strategies help it avoid testing all possible solutions by eliminating the ones that are obviously wrong. This helps save time and effort.



Google's Alpha Go defeats Lee Sedol (a previous Go champion)

Minimax algorithm

Now that we have briefly discussed combinatorial search, let's talk about the heuristics that are employed by combinatorial search algorithms. These heuristics are used to speed up the search strategy and the Minimax algorithm is one such strategy used by combinatorial search. When two players are playing against each other, they are basically working towards opposite goals. So each side needs to predict what the opposing player is going to do in order to win the game. Keeping this in mind, Minimax tries to achieve this through strategy. It will try to minimize the function that the opponent is trying to maximize.

As we know, brute forcing the solution is not an option. The computer cannot go through all the possible states and then get the best possible set of moves to win the game. The computer can only optimize the moves based on the current state using a heuristic. The computer constructs a tree and it starts from the bottom. It evaluates which moves would benefit its opponent. Basically, it knows which moves the opponent is going to make based on the premise that the opponent will make the moves that would benefit them the most, and thereby be of the least benefit to the computer. This outcome is one of the terminal nodes of the tree and the computer uses this position to work backwards. Each option that's available to the computer can be assigned a value and it can then pick the highest value to take an action.

Alpha-Beta pruning

Minimax search is an efficient strategy, but it still ends up exploring parts of the tree that are irrelevant. Let's consider a tree where we are supposed to search for solutions. Once we find an indicator on a node that tells us that the solution does not exist in that sub-tree, there is no need to evaluate that sub-tree. But Minimax search is a bit too conservative, so it ends up exploring that sub-tree.

We need to be smart about it and avoid searching a part of a tree that is not necessary. This process is called **pruning** and Alpha-Beta pruning is a type of avoidance strategy that is used to avoid searching parts of the tree that do not contain the solution.

The Alpha and Beta parameters in alpha-beta pruning refer to the two bounds that are used during the calculation. These parameters refer to the values that restrict the set of possible solutions. This is based on the section of the tree that has already been explored. Alpha is the maximum lower bound of the number of possible solutions and Beta is the minimum upper bound on the number of possible solutions.

As we discussed earlier, each node can be assigned a value based on the current state. When the algorithm considers any new node as a potential path to the solution, it can work out if the current estimate of the value of the node lies between alpha and beta. This is how it prunes the search.

Negamax algorithm

The **Negamax** algorithm is a variant of Minimax that's frequently used in real world implementations. A two-player game is usually a zero-sum game, which means that one player's loss is equal to another player's gain and vice versa. Negamax uses this property extensively to come up with a strategy to increases its chances of winning the game.

In terms of the game, the value of a given position to the first player is the negation of the value to the second player. Each player looks for a move that will maximize the damage to the opponent. The value resulting from the move should be such that the opponent gets the least value. This works both ways seamlessly, which means that a single method can be used to value the positions. This is where it has an advantage over Minimax in terms of simplicity. Minimax requires that the first player select the move with the maximum value, whereas the second player must select a move with the minimum value. Alpha-beta pruning is used here as well.

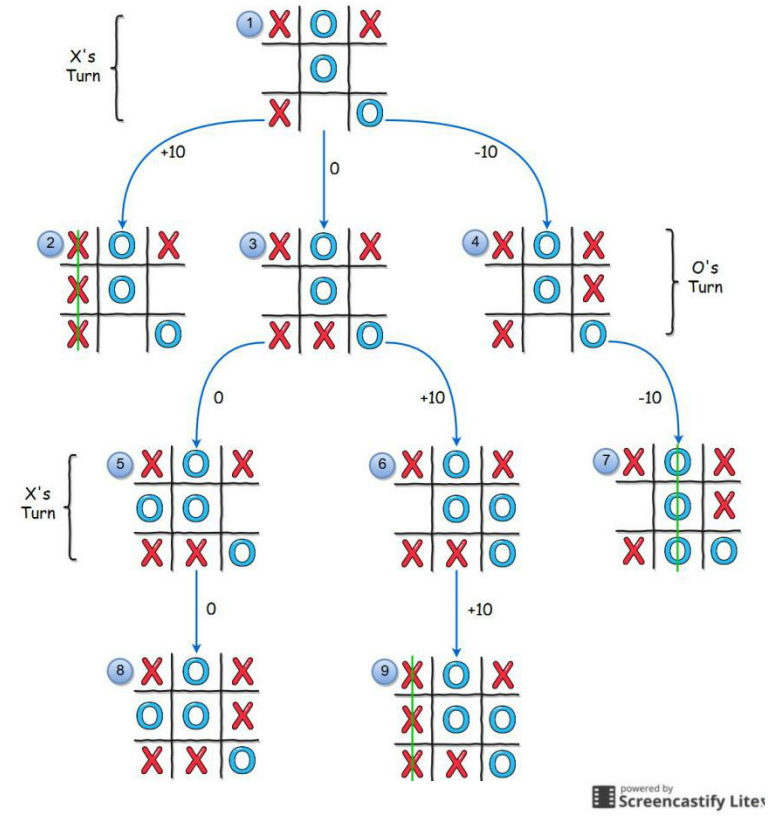
Installing easyAI library

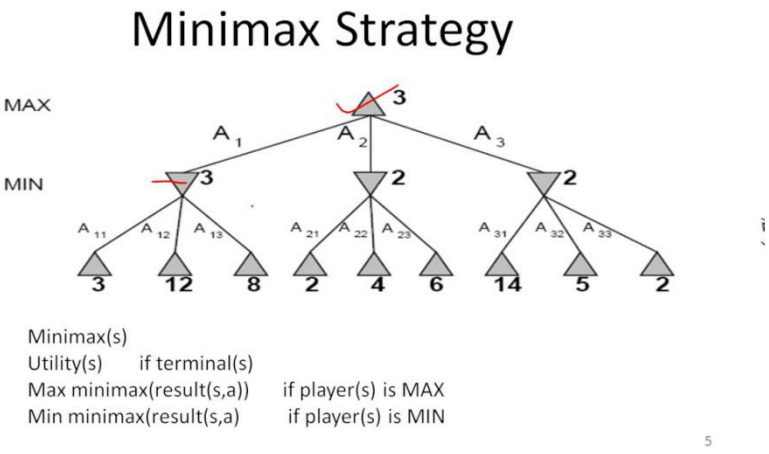
We will be using a library called easyAI in this chapter. It is an artificial intelligence framework and it provides all the functionality necessary to build two-player games. You can learn about it at http://zulko.github.io/easyAI.

Install it by running the following command on your Terminal:

```
$ pip3 install easyAI
```

We need some of the files to be accessible in order to use some of the pre-built routines. For ease of use, the code provided with this book contains a folder called easyAI. Make sure you place this folder in the same folder as your code files. This folder is basically a subset of the easyAI GitHub repository available at https://github.com/Zulko/easyAI. You can go through the source code to make yourself more familiar with it.





Minimax(s)
Utility(s) if terminal(s)
Max minimax(result(s,a)) if player(s) is MAX
Min minimax(result(s,a) if player(s) is MIN

5

Source Code

```python
#AI powered tic-tac-toe game
#Group project using minimax theorem NIIT
#Rounak Saha , Arpan Goswami ,Animesh Arya ,Shailesh Das, Ayush , Ritik

from math import inf as infinity
from random import choice
import platform
import time
from os import system


HUMAN = -1
COMP = +1
board = [
    [0, 0, 0],
    [0, 0, 0],
    [0, 0, 0],
]
```

Tic-Tac-Toe (Noughts and Crosses) is probably one of the most famous games. Let's see how to build a game where the computer can play against the user.

```python
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:
```

```
    ]
    if [player, player, player] in win_state:
        return True
    else:
        return False


def game_over(state):
    """
    This function test if the human or computer wins
    :param state: the state of the current board
    :return: True if the human or computer wins
    """
    return wins(state, HUMAN) or wins(state, COMP)


def empty_cells(state):
    """
    Each empty cell will be added into cells' list
```

```
def minimax(state, depth, player):
    """
    AI function that choice the best move
    :param state: current state of the board
    :param depth: node index in the tree (0 <= depth <= 9),
    but never nine in this case (see iaturn() function)
    :param player: an human or a computer
    :return: a list with [the best row, best col, best score]
    """
    if player == COMP:
        best = [-1, -1, -infinity]
    else:
        best = [-1, -1, +infinity]

    if depth == 0 or game_over(state):
        score = evaluate(state)
        return [-1, -1, score]

    for cell in empty_cells(state):
        x, y = cell[0], cell[1]
        state[x][y] = player
        score = minimax(state, depth - 1, -player)
        state[x][y] = 0
        score[0], score[1] = x, y

        if player == COMP:
            if score[2] > best[2]:
                best = score  # max value
```

```
        else:
            if score[2] < best[2]:
                best = score  # min value

    return best


def clean():
    """
    Clears the console
    """
    os_name = platform.system().lower()
    if 'windows' in os_name:
        system('cls')
    else:
        system('clear')


def render(state, c_choice, h_choice):
    """
    Print the board on console
    :param state: current state of the board
    """
    print('----------------')
    for row in state:
        print('\n----------------')
        for cell in row:
            if cell == +1:
                print('|', c_choice, '|', end='')
            elif cell == -1:
                print('|', h_choice, '|', end='')
```

```
            else:
                print('|', ' ', '|', end='')
        print('\n----------------')


def ai_turn(c_choice, h_choice):
    """
    It calls the minimax function if the depth < 9,
    else it choices a random coordinate.
    :param c_choice: computer's choice X or O
    :param h_choice: human's choice X or O
    :return:
    """
    depth = len(empty_cells(board))
    if depth == 0 or game_over(board):
        return

    clean()
    print('Computer turn [{}]'.format(c_choice))
    render(board, c_choice, h_choice)

    if depth == 9:
        x = choice([0, 1, 2])
        y = choice([0, 1, 2])
    else:
        move = minimax(board, depth, COMP)
        x, y = move[0], move[1]

    set_move(x, y, COMP)
    time.sleep(1)
```

```
def human_turn(c_choice, h_choice):
    """
    The Human plays choosing a valid move.
    :param c_choice: computer's choice X or O
    :param h_choice: human's choice X or O
    :return:
    """
    depth = len(empty_cells(board))
    if depth == 0 or game_over(board):
        return

    # Dictionary of valid moves
    move = -1
    moves = {
        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],
    }

    clean()
    print('Human turn [{}]'.format(h_choice))
    render(board, c_choice, h_choice)

    while (move < 1 or move > 9):
        try:
            move = int(input('Use numpad (1..9): '))
            coord = moves[move]
            try_move = set_move(coord[0], coord[1], HUMAN)

            if try_move == False:
                print('Bad move')
```

```
                move = -1
            except KeyboardInterrupt:
                print('Bye')
                exit()
            except:
                print('Bad choice')


def main():
    """
    Main function that calls all functions
    """
    clean()
    h_choice = '' # X or O
    c_choice = '' # X or O
    first = ''  # if human is the first

    # Human chooses X or O to play
    while h_choice != 'O' and h_choice != 'X':
        try:
            print('')
            h_choice = input('Choose X or O\nChosen: ').upper()
        except KeyboardInterrupt:
            print('Bye')
            exit()
        except:
            print('Bad choice')

    # Setting computer's choice
    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 KeyboardInterrupt:
        print('Bye')
        exit()
    except:
        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('Human turn [{}]'.format(h_choice))
    render(board, c_choice, h_choice)
    print('YOU WIN!')
elif wins(board, COMP):
    clean()
```

```
            print('Computer turn [{}]'.format(c_choice))
            render(board, c_choice, h_choice)
            print('YOU LOSE!')
        else:
            clean()
            render(board, c_choice, h_choice)
            print('DRAW!')

        exit()


if __name__ == '__main__':
    main()
```

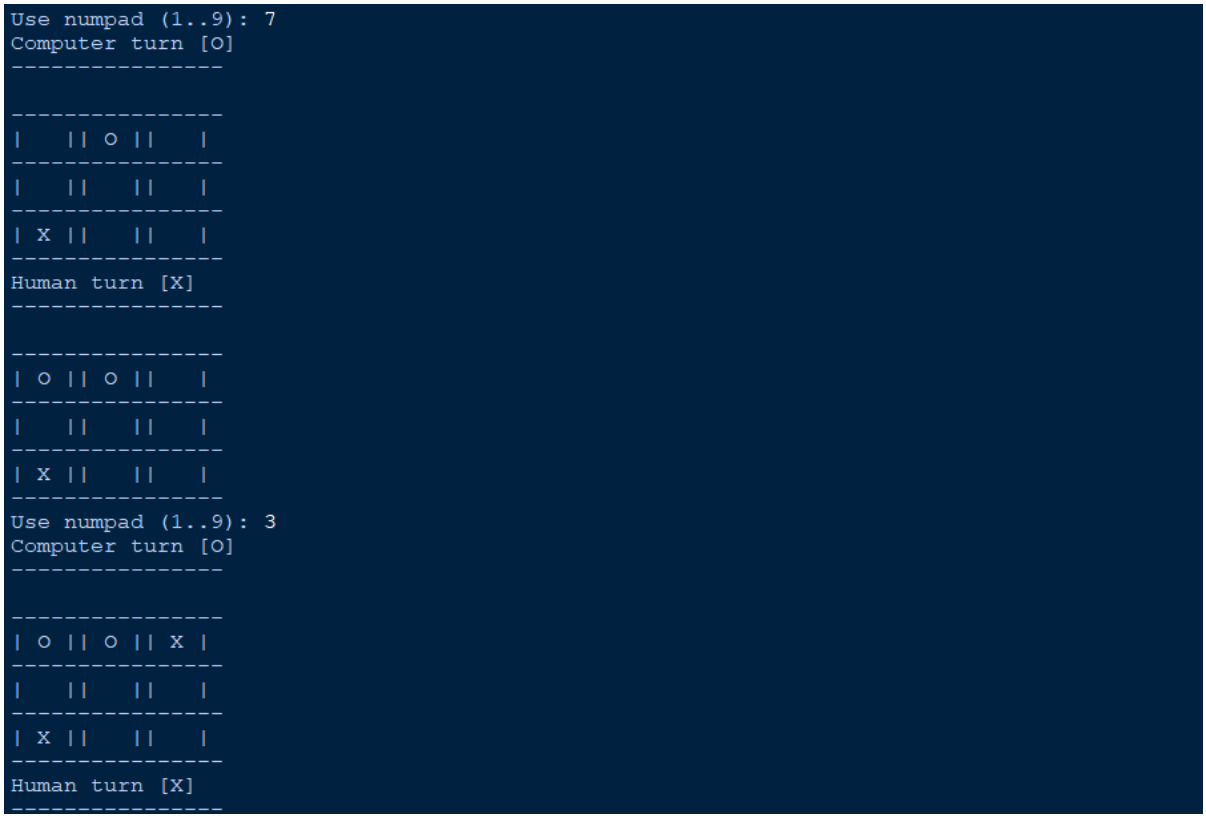
Output

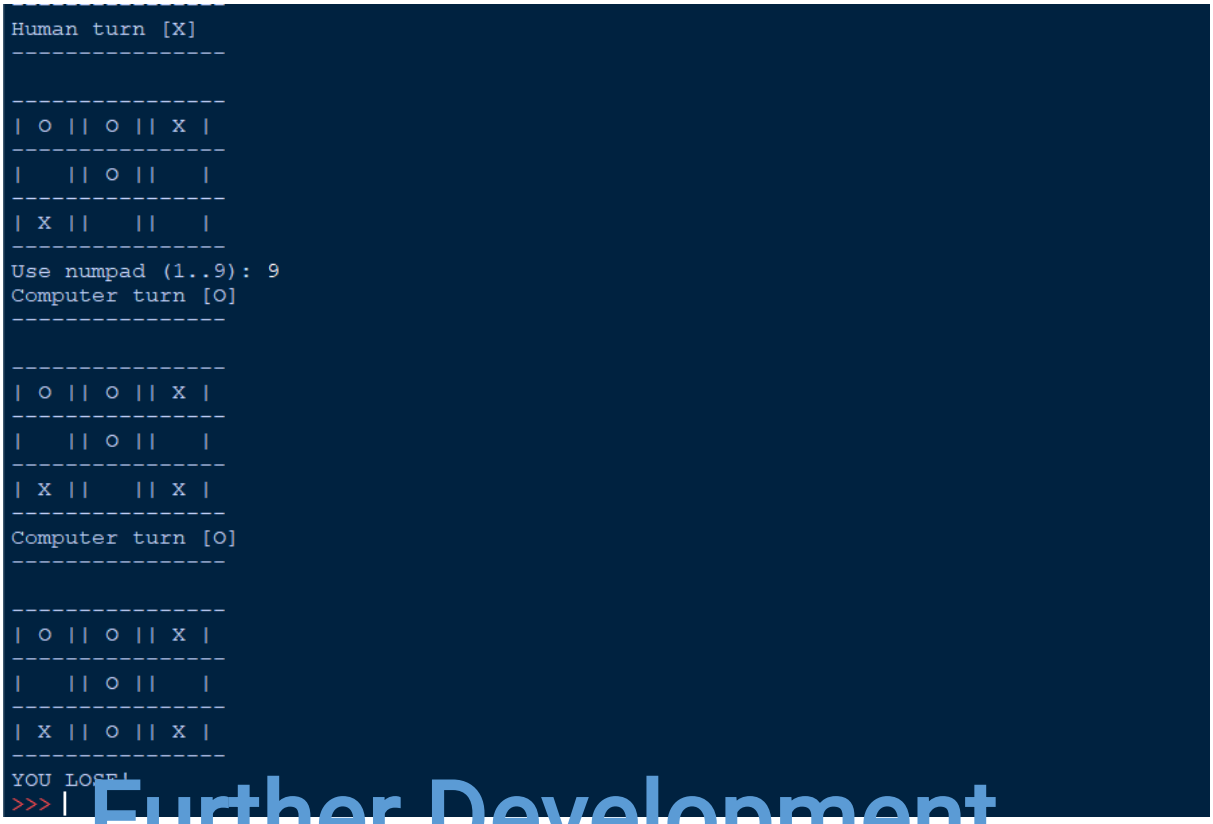
```
======= RESTART: C:\Users\Rounak\Desktop\Tic tac toe game using AI.py =======

Choose X or O
Chosen: X
First to start?[y/n]: n
Computer turn [O]
----------------

----------------
|   ||   ||   |
----------------
|   ||   ||   |
----------------
|   ||   ||   |
----------------
Human turn [X]
----------------

----------------
|   || O ||   |
----------------
|   ||   ||   |
----------------
|   ||   ||   |
----------------
Use numpad (1..9): 7
Computer turn [O]
----------------
```





Further Development

1) Applying this code with the proper GUI such that this program

becomes more user friendly.

2) This logic can be further applied to complicated games which was witnessed by the world by the unforeseen success of Google's Alpha Go.

Conclusion

1) This game has been developed largely error free. However some errors may have inadvertently crept in. Suggestions to improve the code are welcome.

2) This project was a nice learning curve for us in the field of AI and python.

Bibliography

1) Artificial Intelligence with Python by Pratik Joshi - Packt publishers.

2) https://stackoverflow.com/

3) https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

4) https://www.geeksforgeeks.org/combinatorial-game-theory-set-4-sprague-grundy-theorem/

5) https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-5-zobrist-hashing/

THANK YOU