Project Based Learning(PBL) Report

TIC-TAC-TOE Game with AI

BACHELOR OF TECHNOLOGYIN

COMPUTER SCIENCE AND ENGINEERING

Under the esteemed guidance of

Dr. A. Hariprasad

Associate Professor

By

E Saikiran (22R15A0514)
B Prabhaker(22R15A0513)

Department of Computer Science and Engineering Accredited by NBA

Geethanjali College of Engineering and Technology(UGC Autonomous)
(Affiliated to J.N.T.U.H, Approved by AICTE, New Delhi)Cheeryal (V), Keesara (M), Medchal.Dist.-501 301.

August-2023

Geethanjali College of Engineering & Technology

(UGC Autonomous)
(Affiliated to JNTUH Approved by AICTE,New Delhi)
Cheeryal (V), Keesara(M), MedchalDist.-501 301.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Accredited by NBA

DECLARATION BY THE CANDIDATE

I/We **E Saikiran(22R15A0514),B Prabhaker(22R15A0513)** hereby declare that the PBL report entitled **"TIC-TAC-TOE Game with AI "** is done under the guidance of**, Dr.Hariprasad, Associate Professor**, Department of Computer Science and Engineering, Geethanjali College of Engineering andTechnology.

E Sai kiran(22R15A0514),

B Prabhaker(22R15A0513),

Department of CSE,

GCET

Geethanjali College of Engineering & Technology

(UGC Autonomous)
(Affiliated to JNTUH Approved by AICTE,New Delhi)
Cheeryal (V), Keesara(M), MedchalDist.-501 301.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Accredited by NBA

CERTIFICATE

This is to certify that the B.Tech Project Based Learning(BPL) report entitled "TIC-TAC-TOE Game with AI" is a bonafide work submitted by Ejumalla Saikiran(22R15A0514),Banala Prabhaker,under the course of ARTFICIAL INTELLIGENCE(AI).

Course Instructor

Mr. Hariprasad

HOD-CSE

Dr.A Sree Lakshmi

Professor

ACKNOWLEDGEMENT

It is our pleasure to be indebted to various people, who directly or indirectly contributed in the development of this work and who influenced our thinking, behavior and acts during the course of study.

We express our sincere gratitude to Prof. (Dr.) A Sree Lakshmi (HOD of CSE.) for providing us an opportunity to undergo this PBL as the part of the curriculum.

We are thankful Dr. Hariprasad, Associate Professor, for his support, cooperation, and motivation provided to us during the training for constant inspiration, presence and blessings.

Lastly, we would like to thank the almighty and our parents for their moral support and friends with whom we shared our day-to-day experience and received lots of suggestions that improved our quality of work.

ABSTRACT

This is the game named Tic-Tac-Toe. Generally it is played between two players. But we are implementing a code that can create an opportunity to a human player with the machine player through coding. Certainly! Tic Tac Toe is a classic two-player game that is played on a 3x3 grid. The objective of the game is to get three of your symbols (usually "X" or "O") in a row, either horizontally, vertically, or diagonally, before your opponent does.

INTRODUCTION

Certainly! Tic Tac Toe is a classic two-player game that is played on a 3x3 grid. The objective of the game is to get three of your symbols (usually "X" or "O") in a row, either horizontally, vertically, or diagonally, before your opponent does.

Here are the basic rules of Tic Tac Toe:

Game Board: The game is played on a 3x3 grid, forming a total of 9 cells.

Players: There are two players, often denoted as "X" and "O."

Turns: Players take turns placing their symbol in an empty cell of the grid.

Winning Conditions: The game is won when a player has three of their symbols in a row (horizontally, vertically, or diagonally). The game ends as soon as one player achieves this.

Draw (Tie): If all cells are filled, and no player has three in a row, the game is a draw.

Game Flow:

Player X makes the first move, placing an "X" in any empty cell.
Player O then places an "O" in any remaining empty cell.
Players take turns until one player wins or the game is a draw.
Here's a simple example of a Tic Tac Toe board:

markdown
Copy code
```
1 | 2 | 3
---------
4 | 5 | 6
---------
7 | 8 | 9
```
Players can choose any empty cell to place their symbol by referencing the corresponding number.

Here's an example of a game in progress:

markdown
Copy code
```
X | O | X
---------
O | X | O
---------
X |   | O
```
In this example, Player X has won by having three "X" symbols in a diagonal row.

Tic Tac Toe is a game that is often used as a teaching tool for programming, as it involves concepts like decision-making, conditionals, and loops. There are also more advanced versions of the game, such as 4x4 or larger grids, which add complexity to the gameplay.

SOFTWARE USED

PYTHON 3:

Certainly! Python 3 is the latest major version of the Python programming language. Here are some key points about Python 3:

Syntax: Python 3 introduced several syntactical changes compared to Python 2. The most notable change is the print statement becoming a print function. In Python 3, you use print() with parentheses.

Example in Python 2:

python
Copy code
```
print "Hello, World!"
```
Equivalent code in Python 3:

python
Copy code
```
print("Hello, World!")
```
Unicode Support: Python 3 handles strings as Unicode by default, whereas Python 2 treated them as ASCII. This is important for internationalization and better supports a wide range of characters and symbols.

Division Operator: In Python 3, the division operator / performs true division, returning a floating-point result, even if the operands are integers. To perform integer division, you use the // operator.

Example:

python
Copy code
```
# Python 3
result = 5 / 2  # Result: 2.5

# Integer division
result = 5 // 2  # Result: 2
```
Print Function: As mentioned earlier, the print() function requires parentheses in Python 3. This change was made for consistency and to make print a true function.

Range and xrange: In Python 2, there were two functions for creating ranges: range() and xrange(). In Python 3, range() is the only option, and it behaves like Python 2's xrange().

Example:

python
Copy code
```
# Python 2
for i in xrange(5):
    print(i)

# Python 3
for i in range(5):
    print(i)
```

Error Handling: The as keyword is used for exception handling in Python 3 instead of a comma.

Example in Python 2:

python
Copy code
```
try:
    # some code
except IOError, e:
    # handle the exception
```
Equivalent code in Python 3:

python
Copy code
```
try:
    # some code
except IOError as e:
    # handle the exception
```
Input Function: In Python 3, the input() function is used for taking user input, while raw_input() from Python 2 is removed.

Example:

python
Copy code
```
# Python 3
name = input("Enter your name: ")

# Python 2
# name = raw_input("Enter your name: ")
```
It's important to note that Python 2 reached its end of life on January 1, 2020, which means it no longer receives official support or updates. If you are starting new projects, it's recommended to use Python 3.

SOURCE CODE

1) Code for game between two Human players:

```
print("Select your number from the table")
table=[['1','2','3'],
      ['4','5','6'],
      ['7','8','9']]
for i in range (3):
   for j in range (3):
      print(table[i][j],end=" ")
   print()
n=1
while n!=0:
   print("Enter the number 1->> to play player-1, 2->> to play player-2")
   n=int(input())
   if n==1:
      p1=input("player-1 Select the number from above tic-tac-toe table:")
      for i in range (3):
         for j in range (3):
            if table[i][j]==p1:
               table[i][j]='X'
            if table[i][j]=='X':
               print("Alredy Marked this place!")
      print()
      for i in range (3):
         for j in range (3):
            print(table[i][j],end=" ")

         print()

   if (table[0][0]=='X' and table[0][1]=='X' and table[0][2]=='X'):
       print("****PLAYER-1 WIN****")
   elif (table[1][0]=='X' and table[1][1]=='X' and table[1][2]=='X'):
       print("****PLAYER-1 WIN****")
   elif (table[2][0]=='X' and table[2][1]=='X'and table[2][2]=='X'):
       print("****PLAYER-1 WIN****")
   elif (table[0][0]=='X' and table[1][0]=='X' and table[2][0]=='X'):
       print("****PLAYER-1 WIN****")
   elif (table[0][1]=='X' and table[1][1]=='X' and table[2][1]=='X'):
       print("****PLAYER-1 WIN****")
   elif (table[0][2]=='X' and table[1][2]=='X' and table[2][2]=='X'):
       print("****PLAYER-1 WIN****")
   elif (table[0][0]=='X' and table[1][1]=='X' and table[2][2]=='X'):
       print("****PLAYER-1 WIN****")
   elif (table[0][2]=='X' and table[1][1]=='X' and table[2][0]=='X'):
       print("*****PLAYER-1 WIN****")
```

```
   if n==2:
      print("player-2 play ####tic-tac-toe table:####")
      for i in range (3):
         for j in range (3):
```

```
        if table[i][j]=='X':
            if


for i in range (3):
    for j in range (3):
        print(table[i][j],end=" ")

    print()

if (table[0][0]=='O' and table[0][1]=='O' and table[0][2]=='O'):
    print("PLAYER-2 WIN")
if (table[1][0]=='O' and table[1][1]=='O' and table[1][2]=='O'):
    print("PLAYER-2 WIN")
if (table[2][0]=='O' and table[2][1]=='O'and table[2][2]=='O'):
    print("PLAYER-2 WIN")
if (table[0][0]=='O' and table[1][0]=='O' and table[2][0]=='O'):
    print("PLAYER-2 WIN")
if (table[0][1]=='O' and table[1][1]=='O' and table[2][1]=='O'):
    print("PLAYER-2 WIN")
if (table[0][2]=='O' and table[1][2]=='O' and table[2][2]=='O'):
    print("PLAYER-2 WIN")
if (table[0][0]=='O' and table[1][1]=='O' and table[2][2]=='O'):
    print("PLAYER-2 WIN")
if (table[0][2]=='O' and table[1][1]=='O' and table[2][0]=='O'):
    print("PLAYER-2 WIN")
```

2) Code between Human player and Machine play using Random function:

```
import random

def print_board(board):
    for row in board:
        print(" ".join(row))
    print()

def check_winner(board, player):
    # Check rows, columns, and diagonals
    for i in range(3):
        if all(board[i][j] == player for j in range(3)) or all(board[j][i] == player for j in range(3)):
            return True
    if all(board[i][i] == player for i in range(3)) or all(board[i][2 - i] == player for i in range(3)):
        return True
    return False

def is_board_full(board):
    return all(board[i][j] != ' ' for i in range(3) for j in range(3))

def get_empty_cells(board):
    return [(i, j) for i in range(3) for j in range(3) if board[i][j] == ' ']

def player_move(board):
    while True:
        try:
            row = int(input("Enter the row (0, 1, or 2): "))
            col = int(input("Enter the column (0, 1, or 2): "))
            if board[row][col] == ' ':
                return row, col
            else:
                print("That cell is already taken. Try again.")
        except (ValueError, IndexError):
            print("Invalid input. Please enter a number between 0 and 2.")

def computer_move(board):
    empty_cells = get_empty_cells(board)
    return random.choice(empty_cells)

def play_game():
    board = [[' ' for _ in range(3)] for _ in range(3)]
    players = ['X', 'O']
    turn = 0

    while True:
        print_board(board)

        player = players[turn % 2]

        if player == 'X':
            row, col = player_move(board)
        else:
            print("Computer's turn:")
            row, col = computer_move(board)

        board[row][col] = player

        if check_winner(board, player):
            print_board(board)
            print(f"{player} wins!")
            break

        if is_board_full(board):
            print_board(board)
            print("It's a tie!")
            break

        turn += 1


table=[['00 ','01 ','02 '],
        ['10 ','11 ','12 '],
        ['20 ','21 ','22 ']]

for m in range (3):
    for l in range (3):                    i
        print(table[m][l],end="")

    print()
play_game()
```

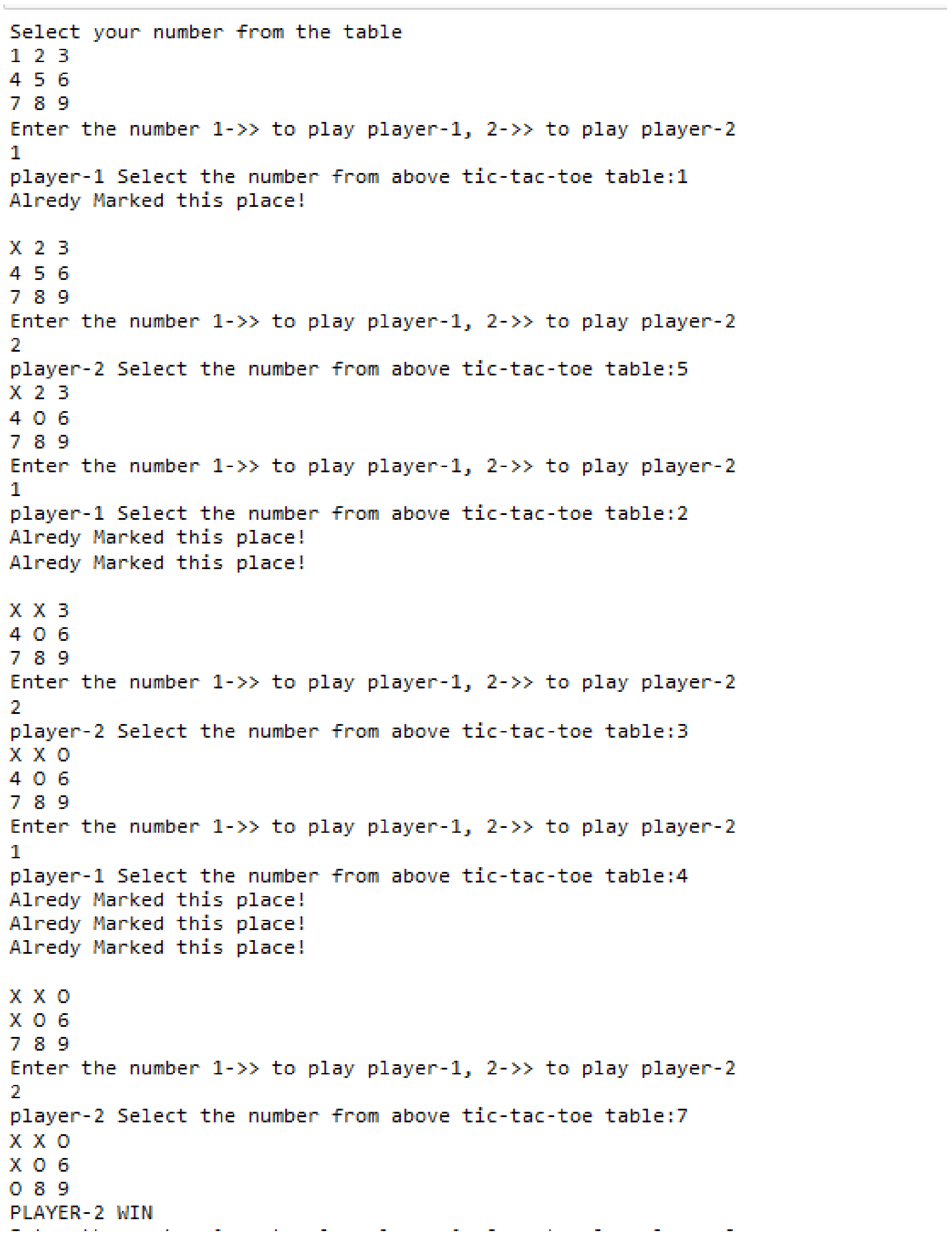
3) Code between Human player and Machine play using Alpha Beta Pruning Algorithm :

```
import copy

def print_board(board):
   for row in board:
      print(" ".join(row))
   print()

def check_winner(board, player):
   # Check rows, columns, and diagonals
   for i in range(3):
      if all(board[i][j] == player for j in range(3)) or all(board[j][i] == player for j in range(3)):
         return True
   if all(board[i][i] == player for i in range(3)) or all(board[i][2 - i] == player for i in range(3)):
      return True
   return False

def is_board_full(board):
   return all(board[i][j] != ' ' for i in range(3) for j in range(3))

def get_empty_cells(board):
   return [(i, j) for i in range(3) for j in range(3) if board[i][j] == ' ']

def player_move(board):
   while True:
      try:
         row = int(input("Enter the row (0, 1, or 2): "))
         col = int(input("Enter the column (0, 1, or 2): "))
         if board[row][col] == ' ':
            return row, col
         else:
            print("That cell is already taken. Try again.")
      except (ValueError, IndexError):
         print("Invalid input. Please enter a number between 0 and 2.")

def minimax(board, depth, maximizing_player):
   scores = {'X': -1, 'O': 1, 'Tie': 0}

   if check_winner(board, 'X'):
      return -1
   elif check_winner(board, 'O'):
      return 1
   elif is_board_full(board):
      return 0

   if maximizing_player:
      max_eval = float('-inf')
      for i, j in get_empty_cells(board):                    i
         new_board = copy.deepcopy(board)
         new_board[i][j] = 'O'
         eval = minimax(new_board, depth + 1, False)
```

```
            max_eval = max(max_eval, eval)
        return max_eval
    else:
        min_eval = float('inf')
        for i, j in get_empty_cells(board):
            new_board = copy.deepcopy(board)
            new_board[i][j] = 'X'
            eval = minimax(new_board, depth + 1, True)
            min_eval = min(min_eval, eval)
        return min_eval

def computer_move(board):
    best_move = None
    best_eval = float('-inf')

    for i, j in get_empty_cells(board):
        new_board = copy.deepcopy(board)
        new_board[i][j] = 'O'
        eval = minimax(new_board, 0, False)

        if eval > best_eval:
            best_eval = eval
            best_move = (i, j)

    return best_move

def play_game():
    board = [[' ' for _ in range(3)] for _ in range(3)]
    players = ['X', 'O']
    turn = 0

    while True:
        print_board(board)

        player = players[turn % 2]

        if player == 'X':
            row, col = player_move(board)
        else:
            print("Computer's turn:")
            row, col = computer_move(board)

        board[row][col] = player

        if check_winner(board, player):
            print_board(board)
            print(f"{player} wins!")
            break
       if is_board_full(board):
            print_board(board)
            print("It's a tie!")                              i
            break
        turn += 1
     if __name__ == "__main__":
         play_game()
```

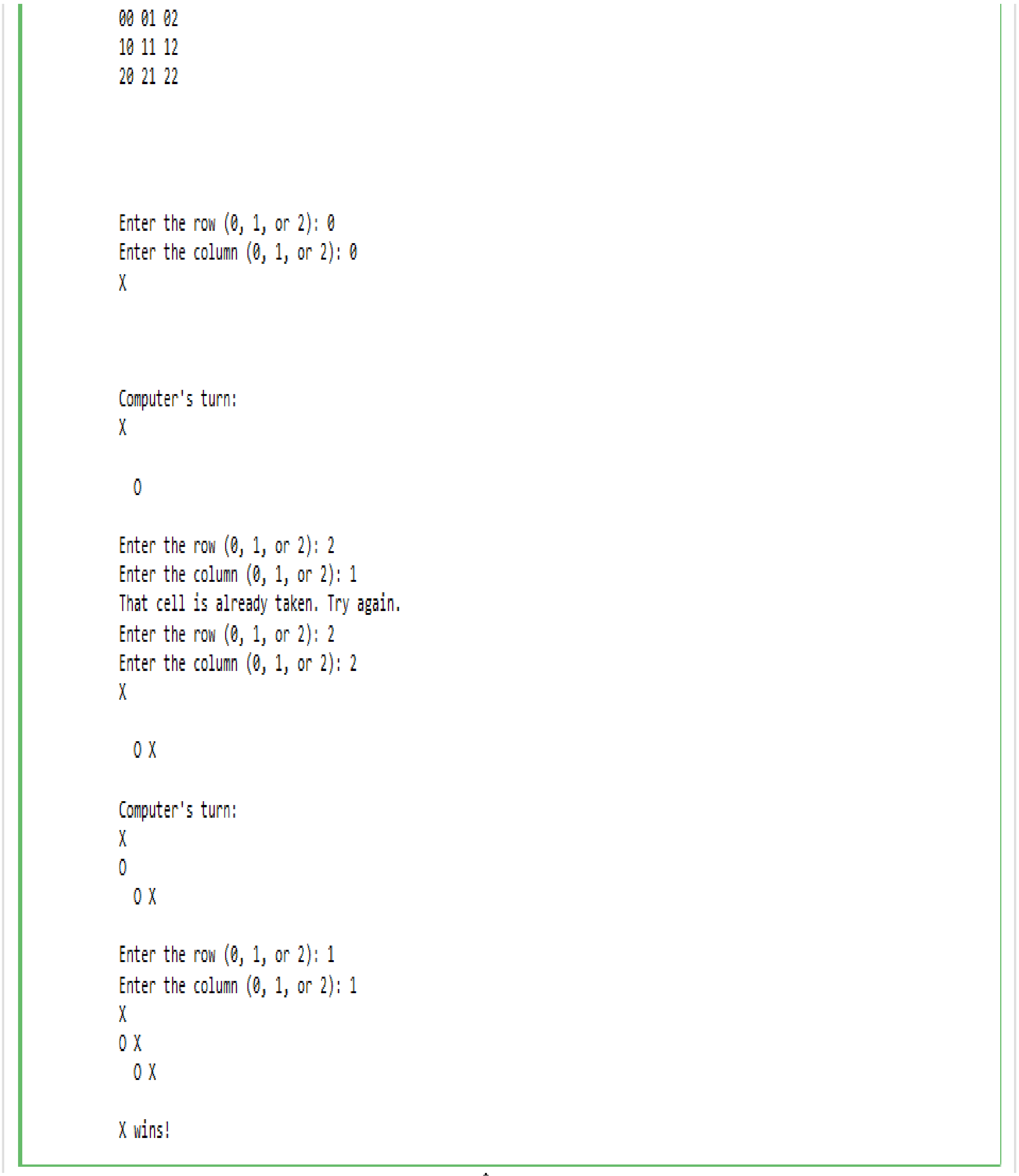
OUTPUT SCREEN SHOTS

Screen shot 1: The play is between HUMAN-HUMAN.



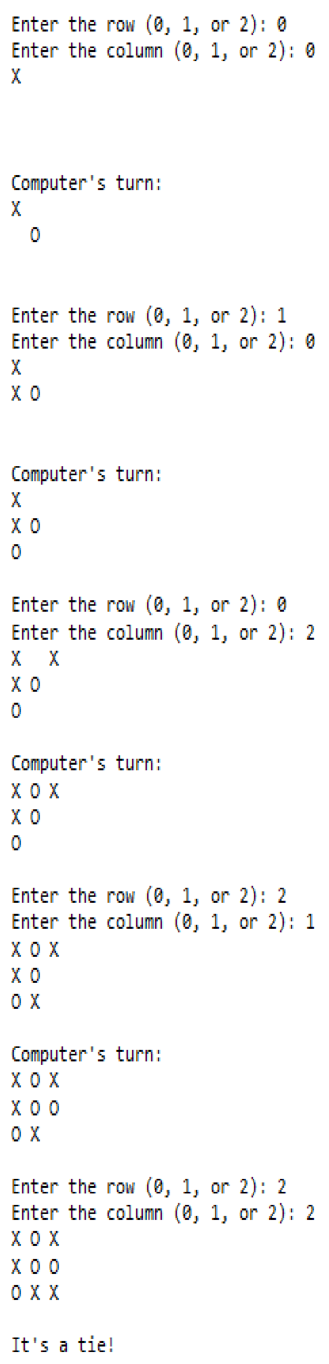
```
Select your number from the table
1 2 3
4 5 6
7 8 9
Enter the number 1->> to play player-1, 2->> to play player-2
1
player-1 Select the number from above tic-tac-toe table:1
Alredy Marked this place!

X 2 3
4 5 6
7 8 9
Enter the number 1->> to play player-1, 2->> to play player-2
2
player-2 Select the number from above tic-tac-toe table:5
X 2 3
4 O 6
7 8 9
Enter the number 1->> to play player-1, 2->> to play player-2
1
player-1 Select the number from above tic-tac-toe table:2
Alredy Marked this place!
Alredy Marked this place!

X X 3
4 O 6
7 8 9
Enter the number 1->> to play player-1, 2->> to play player-2
2
player-2 Select the number from above tic-tac-toe table:3
X X O
4 O 6
7 8 9
Enter the number 1->> to play player-1, 2->> to play player-2
1
player-1 Select the number from above tic-tac-toe table:4
Alredy Marked this place!
Alredy Marked this place!
Alredy Marked this place!

X X O
X O 6
7 8 9
Enter the number 1->> to play player-1, 2->> to play player-2
2
player-2 Select the number from above tic-tac-toe table:7
X X O
X O 6
O 8 9
PLAYER-2 WIN
```

Screen shot 2: The play is between HUMAN-[WEEK AI] using Random function.



```
00 01 02
10 11 12
20 21 22




Enter the row (0, 1, or 2): 0
Enter the column (0, 1, or 2): 0
X



Computer's turn:
X

  O

Enter the row (0, 1, or 2): 2
Enter the column (0, 1, or 2): 1
That cell is already taken. Try again.
Enter the row (0, 1, or 2): 2
Enter the column (0, 1, or 2): 2
X

  O X

Computer's turn:
X
O
  O X

Enter the row (0, 1, or 2): 1
Enter the column (0, 1, or 2): 1
X
O X
  O X

X wins!
```

Screen shot 1: The play is between HUMAN-STRONG AI using Alpha-Beta Pruning Agorithm.



```
Enter the row (0, 1, or 2): 0
Enter the column (0, 1, or 2): 0
X



Computer's turn:
X
  O


Enter the row (0, 1, or 2): 1
Enter the column (0, 1, or 2): 0
X
X O


Computer's turn:
X
X O
O

Enter the row (0, 1, or 2): 0
Enter the column (0, 1, or 2): 2
X   X
X O
O

Computer's turn:
X O X
X O
O

Enter the row (0, 1, or 2): 2
Enter the column (0, 1, or 2): 1
X O X
X O
O X

Computer's turn:
X O X
X O O
O X

Enter the row (0, 1, or 2): 2
Enter the column (0, 1, or 2): 2
X O X
X O O
O X X

It's a tie!
```

CONCLUSION

Tic-Tac-Toe AI:

Implementing Tic-Tac-Toe AI involves creating a program that can play the game intelligently against a human opponent. Key components of a Tic-Tac-Toe AI include:

Game Logic: Develop the core logic of the game, including the board representation, rules, and conditions for winning or drawing.

Evaluation Function: Create an evaluation function that assesses the current state of the board and assigns a score, indicating how favorable the position is for the AI. This function guides the AI in making strategic moves.

Minimax Algorithm: Implement the Minimax algorithm, a recursive algorithm that explores all possible moves and outcomes to determine the best move for the AI. This algorithm ensures that the AI makes optimal decisions.

Alpha-Beta Pruning: Enhance the Minimax algorithm with Alpha-Beta pruning to reduce the number of unnecessary evaluations, improving the efficiency of the AI.

User Interface: Develop a user interface to allow users to play against the AI. This can be a command-line interface or a graphical user interface, depending on the project's complexity.

Difficulty Levels: Optionally, implement different difficulty levels by adjusting the depth of the search in the Minimax algorithm. Higher depths result in a more challenging AI opponent.

Tic-Tac-Toe for 2 Users:

Creating a Tic-Tac-Toe game for two users involves building a platform for players to take turns and compete against each other. Key considerations include:

Game Board: Design a 3x3 grid to represent the Tic-Tac-Toe board where users can place their symbols (X or O) during their turns.

User Input: Implement a mechanism for users to input their moves. This can be achieved through a graphical interface with buttons or a command-line interface with coordinate inputs.

Game Logic: Develop the logic to check for a win or draw after each move. This involves examining rows, columns, and diagonals to determine if a player has achieved three in a row.

Turn-Based System: Establish a turn-based system where players alternate making moves. Ensure that the game enforces valid moves and prevents players from overwriting each other's moves.

User Interface: Create a user-friendly interface that displays the current state of the board, informs players of the game's progress, and announces the winner or a draw at the end.

Replayability: Allow users to play multiple rounds and implement a reset option to start a new game without restarting the entire program.

In conclusion, while Tic-Tac-Toe AI involves more sophisticated algorithms and strategies to create a challenging opponent, a Tic-Tac-Toe game for two users focuses on providing a simple and enjoyable platform for human players to compete against each other. Both implementations can be valuable for learning and practicing programming concepts.