Algorithms and Data Structures

ALPHA-BETA PRUNING
programming project

Three in a row
Tic - Tac - Toe

Students:
Mònica Torner
Núria Mitjavila

Teacher Supervisor:
José Luis Balcázar

Algorithms and Data Structures
Second term, Second year, 2022 - 2023
Bioinformatics degree, ESCI - UPF

1. Introduction

For the programming project of the Algorithms and Data Structures, we decide to do a version of Tic-Tac-Toe that enabled to play with another person (using the same computer) or with an AI. To do that, we applied alpha-beta pruning.

Alpha-beta pruning is a search algorithm commonly used in game theory and artificial intelligence to optimize the search process in game trees. The algorithm works by pruning or eliminating unnecessary branches in the tree that do not affect the final outcome of the game.

The process begins by evaluating the root node of the tree and assigning two values to it: alpha and beta. Alpha represents the best score that the player can achieve, and beta represents the best score that the opponent can have. It recursively explores the tree, so updates the alpha and beta values of each node based on the scores of its children nodes.

When the algorithm encounters a node with a beta value that is less than or equal to its alpha value, it knows that the opponent will never choose this path because it will result in a score that is worse than or equal to its current best option. Similarly, when the algorithm encounters a node with an alpha value that is greater than or equal to its beta value, it knows that the opponent will never choose this path because it will result in a score that is worse than or equal to its current best option.

Overall, alpha-beta pruning is a powerful technique for reducing the search space in game trees, allowing artificial intelligence systems to efficiently find optimal strategies and make better decisions.

2. 1 vs 1 game

Firstly, we import the random module to randomly select which player goes first, and the Game1vsAI module to check for a winner.

```
import random
import Game1vsAI
```

The script contains two functions, "start" and "end". The "end" function checks if the game board is full. If it is the case, it means that the game is over and the function returns True, otherwise, it returns False.

The "start" function takes a board as an argument and begins the game. It starts by randomly selecting a player to go first, then it prints out the state of the board and tells the current player to make a move. The function checks if the move is valid by checking that the input is in the range from 1 to 9 and that the selected space is empty. If the move is valid, the function updates the board with the player's move and checks for a winner using the "win" function from Game1vsAI that we imported.

```
def end(board):
    for row in board:
        for item in row:
            if item == '   ':
                return False
    return True

def start(board):
    player = random.choice(['x', 'o'])
    while True:
        print("Player", player, "turn")
        for row in board:
            for place in row:
                print(place, end=" ")
            print()
        a = True
        while a:
            try:
```

```
19                position = int(input("Enter the position (1-9) to put in an
   empty space: "))
20                print()
21                if position not in range(1, 10):
22                    print("Invalid number! Please choose a number between
   1-9")
23                row, column = (position-1) // 3, (position-1) % 3
24                if board[row][column] != '   ':
25                    print("Invalid move! Please choose an empty space.")
26                else:
27                    board[row][column] = player
28                    a = False
29            except(KeyError, ValueError):
30                print("Invalid input! Please enter a number between 1-9")
31
32        if Game1vsAI.win(board, player):
33            print("Player", player, "has won the game!!!")
34            break
35        if end(board):
36            print("There is no winner!")
37            break
38        if player == 'x': player = 'o'
39        else: player = 'x'
40    for row in board:
41        for place in row:
42            print(place, end=" ")
43        print()
```

If there is a winner, the function prints a message telling the winner and ends the game. If there is no winner and the board is full, the function prints a message indicating that the game has ended in a tie. While the board is not full or no player has won, the function switches the player turn and continues the game until there is a winner or a tie.

3. 1 vs AI game

First of all, the Gameboard function takes a 3x3 board and displays it on the screen using x, o, and symbols to represent occupied by one player, occupied by the other player, and unoccupied, respectively.

```
import random

def Gameboard(board):
    symbols = {1: 'x', -1: 'o', 0: '   '}
    for row in board:
        for cell in row:
            print(symbols[cell], end=' ')
        print()
    print()
```

The win function checks the board to see if a player has won by checking all the rows, columns, and diagonals to see if they have three in a row. The blanks function returns a list of all the empty spaces on the board.

```
def win(board, player):
    for i in range(3):
        if board[i][0] == board[i][1] == board[i][2] == player:
            return True

    for j in range(3):
        if board[0][j] == board[1][j] == board[2][j] == player:
            return True

    if board[0][0] == board[1][1] == board[2][2] == player:
        return True

    if board[0][2] == board[1][1] == board[2][0] == player:
        return True
    return False

def blanks(board):
    blank = []
```

```
    for x, row in enumerate(board):
        for y, col in enumerate(row):
            if board[x][y] == 0:
                blank.append([x, y])
    return blank
```

The minimax function uses the minimax algorithm to determine the best move for the AI opponent. It recursively searches through the game tree to evaluate all possible moves and their outcomes, and chooses the move that maximizes the AI's chance of winning. We decided to use minimax instead of alpha beta pruning, even if it is not the best implementation of alpha beta pruning, as it guarantees optimality for small games like our and is straight forward.

```
def minimax(board, length, a, b, player):
    row = -1
    col = -1
    if length == 0 or (win(board, 1) or win(board, -1)):
        if win(board, -1):
            return row, col, -10
        elif win(board, 1):
            return row, col, 10
        else:
            return row, col, 0
    else:
        for cell in blanks(board):
            board[cell[0]][cell[1]] = player
            score = minimax(board, length - 1, a, b, -player)
            if player == -1:
                if score[2] < b:
                    b, row, col = score[2], cell[0], cell[1]
            else:
                if score[2] > a:
                    a, row, col = score[2], cell[0], cell[1]
            board[cell[0]][cell[1]] = 0
            if a >= b:
                break
        if player == 1:
            return row, col, a
        else: return row, col, b
```

The start function initializes the game board and starts the game loop. It asks the player to make a move by entering a number between 1-9 to indicate which space they want to occupy. It then updates the game board with the player's move and displays it on the screen. If it's the AI's turn, it uses the minimax function to determine its move and updates the game board accordingly. This continues until either a player has won or the board is full.

Finally, the code outputs the winner of the game or indicates if it's a tie.

```
def start(board):
    player = 'x'
    for x, row in enumerate(board):
        for y, col in enumerate(row):
            board[x][y] = 0

    if player == 'o':
        print("It's the IA turn")

    while not (len(blanks(board)) == 0 or (win(board, 1) or win(board, -1)))
   :
        if player == 'x':
            a = True
            while a:
                try:
                    position = int(input('Is your turn, enter a number
   between 1-9: '))
                    row, column = (position-1) // 3, (position-1) % 3
                    if position not in range(1, 10):
                        print("Invalid number! Please choose a number
   between 1-9")
                    elif not ([row, column] in blanks(board)):
                        print('Invalid Move! Try again!')
                    else:
                        board[row][column] = 1
                        Gameboard(board)
                        a = False
                except(KeyError, ValueError):
                    print('Invalid input! Please enter a number between 1-9'
   )
```

```
27
28          else:
29              if len(blanks(board)) == 9:
30                  board[random.choice([0, 1, 2])][random.choice([0, 1, 2])] =
    -1
31                  Gameboard(board)
32              else:
33                  result = minimax(board, len(blanks(board)), -10**9, 10**9,
    -1)
34                  board[result[0]][result[1]] = -1
35                  Gameboard(board)
36
37          if player == 'x': player = 'o'
38          else: player = 'x'
39
40      if win(board, 1):
41          print('Player x has won the game!!!')
42      elif win(board, -1):
43          print('Player o has won the game!!!')
44      else:
45          print('There is no winner!')
```