Mauricio Verduzco Chavira
A20552976

CS 480 Fall 2023 Programming Assignment #01

Due: **Sunday, October 29, 2023, 11:59 PM CST**
Points: **100**

Instructions:

1. Place **all your deliverables (as described below) into a single ZIP** file named:

```
LastName_FirstName_CS480_Programming01.zip
```

2. Submit it to Blackboard Assignments section before the due date. **No late submissions will be accepted. Submit partial work for partial credit.**

Objectives:

1. (100 points) Implement MiniMax adversarial search algorithm.

Problem description:

Your task is to implement in Python the following adversarial search algorithms **(refer to lecture slides and/or your textbook for details | pseudocode provided below)**:

- MiniMax (as specified by the MINIMAX-SEARCH pseudocode below)

```
function MINIMAX-SEARCH(game, state) returns an action
  player ← game.TO-MOVE(state)
  value, move ← MAX-VALUE(game, state)
  return move

function MAX-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a))
    if v2 > v then
      v, move ← v2, a
  return v, move

function MIN-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a))
    if v2 < v then
      v, move ← v2, a
  return v, move
```

- MiniMax with alpha-beta pruning (as specified by the ALPHA-BETA-SEARCH pseudocode below),

```
function ALPHA-BETA-SEARCH(game, state) returns an action
  player ← game.TO-MOVE(state)
  value, move ← MAX-VALUE(game, state, −∞, +∞)
  return move

function MAX-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a), α, β)
    if v2 > v then
      v, move ← v2, a
      α ← MAX(α, v)
    if v ≥ β then return v, move
  return v, move

function MIN-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a), α, β)
    if v2 < v then
      v, move ← v2, a
      β ← MIN(β, v)
    if v ≤ α then return v, move
  return v, move
```

and apply them to play the game of Tic-Tac-Toe (computer). **Using any other approach is not going to be accepted**.

Problem input/command line interface:

Your program should:

- Accept three (3) command line arguments, so your code could be executed with

```
python cs480_P01_AXXXXXXXX.py ALGO FIRST MODE
```

  where:

  - `cs480_P01_AXXXXXXXX.py` is your python code file name,
  - `ALGO` specifies which algorithm the computer player will use:
    - `1` – MiniMax,
    - `2` – MiniMax with alpha-beta pruning,
  - `FIRST` specifies who begins the game:
    - `X`
    - `O`
  - `MODE` is mode in which your program should operate:
    - `1` – human (`X`) versus computer (`O`),
    - `2` – computer (`X`) versus computer (`O`),

  Example:

```
python cs480_P01_A11111111.py 2 X 1
```

  If the number of arguments provided is NOT three (none, one, two or more than three) or arguments are invalid (incorrect `ALGO`, `FIRST` or `MODE`) your program should display the following error message:

```
ERROR: Not enough/too many/illegal input arguments.
```

  and exit.

Program details:

Specific program details:

- The Tic-Tac-Toe game board is represented by 3 x 3 grid with cells numbered as follows

1	2	3
4	5	6
7	8	9

- Possible moves/actions for both players match cell numbers (if a player wants to place an 'X' in the middle of the board, the move/action is '5',

- Your program should begin by displaying the following information:

```
Last Name, First Name, AXXXXXXXX solution:
Algorithm: MiniMax with alpha-beta pruning
First: X
Mode: human versus computer
```

where:

- AXXXXXXXX is your IIT A number,
- Algorithm is the algorithm specified by a command line argument,
- First is the information who makes the first move as specified by a command line argument,
- Mode is the game mode as specified by a command line argument,

- If the game mode is human versus computer display an empty board first and prompt the user to pick the move (see below)

```
   |   |
---+---+---
   |   |
---+---+---
   |   |
```

- When it is human player turn, your program should display the following prompt:

```
X's move. What is your move (possible moves at the moment are: <list of possible moves> | enter 0 to exit the game)?
```

where: `<list of possible moves>` is a sorted list of all available moves at the moment, for example, if the board arrangement is:

X		
O	O	X
	O	

and it is X's move, the prompt should be:

```
What is your move (possible moves at the moment are: 2, 3,7,9) | enter 0 to exit the game)??
```

If the user enters anything other than 0 / valid move number (0 should terminate the game) your program should repeat the prompt above.

Once the user enters a valid move, display the updated game board on screen.

- When it is the computer turn (regardless of the game mode), your program should display (it could be an 'X' or 'O' move):

  `X's selected move: Z. Number of search tree nodes generated: AAAA`

  where:
  - Z is the move/action number (a positive integer from the {1, 2, 3, 4, 5, 6, 7, 8, 9} set) selected by computer
  - AAAA is the **number of search tree nodes generated** (the number of MiniMax nodes computer explored before you made the decision [including "root"]) to select it.

  Follow it with the updated game board on screen.

  - **NOTE!!! Computer's search tree move exploration order should be in a sorted fashion (1, 2, 3, 4, 5, 6, 7, 8, 9 | assuming HERE that ALL moves are available).**

- When the game is complete, your program should display a corresponding message:

  - X WON or O WON
  - TIE
  - X LOST or O LOST

Deliverables:

Your submission should include:

- Python code file(s). Your python source code py file should be named:

  `cs480_P01_AXXXXXXXX.py`

  where AXXXXXXXX is your IIT A number (this is REQUIRED!). If your solution uses multiple files, makes sure that the main (the one that will be run to solve the problem) is named that way and others include your IIT A number in their names as well.
- this document with your results and conclusions. You should rename it to:

  `LastName_FirstName_CS480_Programming01.doc` or `pdf`

Analysis:

Play nine (9) human versus computer (**using both algorithms**) games, each starting with a different move. Count the total number of expanded nodes (sum of expanded nodes for every computer move) and report them in the table below.

Your (X) First move	Computer (0) with MiniMax algorithm. Total (for every move) number of generated nodes	Computer (0) with MiniMax with alpha beta pruning algorithm. Total (for every move) number of generated nodes
1	59705+1053+47+5=**60810**	2338+318+42+5=**2703**
2	63905+1055+53+5=**65018**	2869+269+31+4=**3173**
3	59705+1277+47+5=**61034**	3275+365+23=**3663**
4	63905+899+43+4=**64851**	3574+193+20=**3787**
5	55505+1055+45+4=**56609**	2316+116+39+5=**2476**
6	63905+1405+61+4=**65375**	3590+209+43+5=**3847**
7	59705+927+61+5=**60698**	3809+453+47+5=**4314**
8	63905+1019+49+5=**64978**	4981+189+33+5=**5208**
9	59705+1053+47+5=**60810**	3957+318+42+5=**4322**

What are your conclusions? Which algorithm performed better? Write a short summary below.

Conclusions
We can see that the alpha beta pruning does way less tree nodes than the regular minimax algorithm, something about 5%. Even when we make computer vs computer the regular algorithm would take several seconds but the AB pruning appears to be an instant response. Both algorithms gave a complete response and the desired response ad both wouldn´t allow me to win, and as soon as I made a mistake they won. But the fact that one of them uses less steps, less memory and less time means that is better. Luckily this is a game if TicTacToe but if this were to be something with more possible scenarios (like chess or domino) we could take minutes to make a move. So, in conclusion, pruning is preferred. The two extra spaces of memory for alpha and beta and the little extra steps of comparing and saving new values for variables are a very little price to pay compared with the efficiency of the trade-off.