TCSS 435
Programming Assignment 2

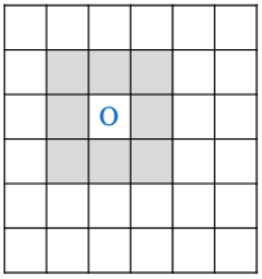
Note:
Be sure to adhere to the University's **Policy on Academic Integrity** as discussed in class. Programming assignments are to be written individually and submitted programs must be the result of your own efforts. Any suspicion of academic integrity violation will be dealt with accordingly.

Objective:
Expressing the problem as a game search problem and identifying proper game strategy. Specifying, designing, and implementing adversarial search methods.

Assignment Details:
In this programming assignment you will implement a program that will engage the user in a 2-person game of **Obstruction**. Your program should use a minimax approach with several-move look-ahead and alpha-beta pruning. The rules of the game are summarized as follows:

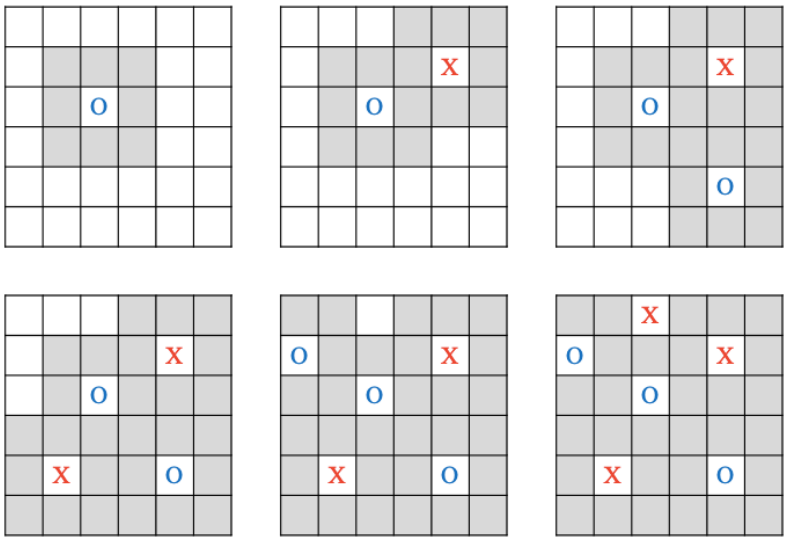
Obstruction:
The game was invented and analyzed by László Kozma, a Romanian mathematician. It is described on his web site here: http://www.lkozma.net/game.html

- **Players:** Two
  Players take turns in marking squares on a grid. The first player unable to move loses.
- **Description:** The game is played on a grid; 6 x 6 is a good size. One player is 'O' and the other is 'X'. The players take turns in writing their symbol in an empty cell. Placing a symbol blocks all the neighboring cells from both players, and you can optionally indicate this by shading them:



The first player unable to move loses.

- **Example:** For example, in the following game the second player 'X' wins because 'O' has nowhere to play:



START: Start with an empty board and decide who starts. The 1st player is 'O' and the 2nd player is 'X'.

GOAL: The first player unable to move loses.

PLAY: Each turn consists of placing a symbol, anywhere on the board and blocking the neighboring cells from both players.

TERMINAL STATE: Win for 'O' or win for 'X'. There is no tie situation, as the next player unable to make a move loses.

Assignment Summary:

The search algorithms you are expected to implement are:

- Minimax (due to space constraints it will most likely be a depth limited minimax)
- Minimax with Alpha-Beta Pruning

Input:

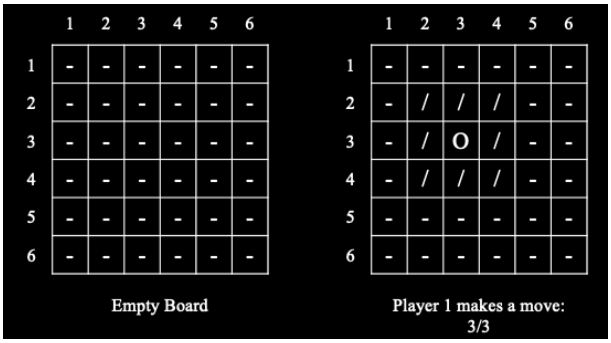
There are 2 inputs for this assignment – game initiation and the game moves.

- Game Initiation: Your program will accept initial instructions to run the game from command line. The program should accept the following inputs in the following format:
  `[player] [searchmethod]`
  - `[player]` choose if AI is player "1" or player "2". You are playing against a Human player. The 1st player (MAX) will always play 'O' symbol and the 2nd player (MIN) will always play 'X' symbol.
  - `[searchmethod]` can be MM (for minimax) or AB (for minimax with alpha-beta pruning)

- Examples:
  - `1 MM`
  - `2 AB`

- Game Moves: Once you start playing the game (via console) you will accept player (Human player) moves and display AI moves in the following form – **r/c,** where **r** is the row and **c** is the column where the symbol needs to be placed.

  Example:



  Please note that '/' indicates blocked cells and no player can make a move, i.e., place a symbol in these cells.

Output:
Your program will generate 2 different outputs – one to the console & and another to a readme file.

- Console Output: Your program output should show the following information
  - Player Info - Example
    ```
    Player 1: Human
    Player 2: AI
    ```
  - Game moves for each player (for both Human and AI) for each turn in the form of **row/column**
  - Board configuration for each turn

- `Readme.txt:`
  The Readme should include –
  - Evaluation function: formula for calculating the utility value of non-terminal leaf nodes.
  - Minmax Algorithm: for at-least two different depth level
    - Number of nodes expanded
    - Depth level for look-ahead
  - Minmax Algorithm with Alpha-Beta pruning: for at-least two different depth level
    - Number of nodes expanded
    - Depth level for look-ahead

Additional Component for TCSS 535:

Additionally, you will be solving this problem for following grid sizes:

- 7 x 6
- 8 x 7
- 8 x 8

In your output file make sure to report the data for all grid sizes.

Hints:

- Implementing Game Tree: You are not creating a game tree of links (as seen in Data Structures). Here we are representing a game tree as a list of list representation, where each node is a state representation of the game board. Each node (game board state) will have a utility value. Number of nodes generated at each depth level will depend on the branching factor.
  Example:
  `gameTree=['a',['b',['d',[],[]],['e',[],[]]],['c',['f',[],[]],[]]]`
- Implementing minimax: Ideally, we would like to have an entire game tree, but practically it is not feasible to do so (space constraint due to high branching factor). So, you are implementing a depth-limited minimax. Make sure to start with a smaller depth before expanding further.
- Utility Value: Utility values are only calculated for the leaf nodes of your game tree. Hence if the leaf node of your game tree is a terminal node (Win for MAX, Win for MIN), then assign them a utility value that will make MAX always choose a Win for MAX (e.g. +100) state and likewise for Min (e.g. -100). For leaf nodes that are not-terminal nodes, design an evaluation function that represents the current game state accurately.
  Hint: think about including how many neighboring cells are blocked by placing a symbol at a particular grid cell. Look at the link provided for game strategy analyzed by Kozma.
- Alpha-beta pruning: Pruning allows you to explore deeper in the game tree. Pruning doesn't change the outcome of minimax but simply discards the sub-optimal paths. Design of your game tree (left align best nodes for MAX and MIN at each level) will help you achieve good pruning factor. Something to keep in mind while generating your game tree.

Submission Guidelines:

Zip and upload the following files on Canvas using the Programming Assignment 2 submission link:

- `Board.java/Board.py`: Source code that models the game board.
- `Solver.java/Solver.py`: Source code that implements minimax and minimax with alpha-beta pruning.
- `Tester.java/Tester.py`: A tester file that connects your Board and Solver code.
- `Readme.txt`: Analysis of the search methods in the format explained in the output section.

Please refer to the grading rubric for breakdown of the points.