TCSS 435
Programming Assignment 2

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

Instructions: In this assignment you will implement a program that will engage the user in a 2-person game of **Maze chase**. Your program should use a minimax approach with several-move look-ahead and alpha-beta pruning. Theoretically solving the problems as a search process includes progressive construction of a solution:

- Establish the problem's components
  - Players: Two players MAX & MIN
  - Initial state
  - Terminal state
  - Operators (successor functions)
    - Game movies & Utility function
  - Solution
- Defining the game search
- Establish the strategy for search:
  - Minimax (MM) algorithm
  - Minimax with alpha-beta pruning (AB)

Game Description: The game is played on a Maze (of size M x N). One player is 'MAX' and the other is 'MIN'. The players take turns in making moves within the maze. The first player to reach the goal state wins the game.

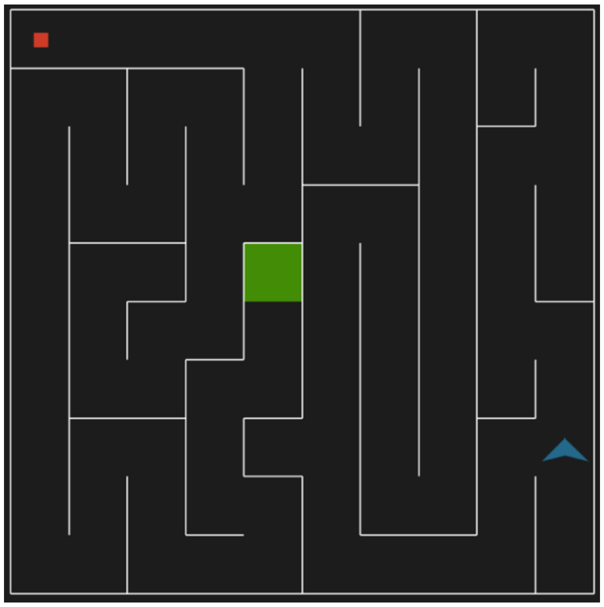


Figure 1: Initial State of Players & Goal State

Step 1: Defining our Maze & Agents

Update your `MazeRunner.py` source code (from Assignment 0) to create a Maze using following configurations *(Note: The `MazeRunner.py` will be our tester file)*:

- Maze configuration (M x N): We will test for following two Maze sizes- default size (10x10) and 20 x 30
    - You can choose the theme to be light or dark and pattern to be vertical or horizontal.
    - Goal State will be generated randomly using random number generator.
    - Set loop percent to be 100.
- Two players `max` and `min` positioned at random location within the maze.

```
max = maze.agent(m, randrow1, randcol1, shape = ‘arrow’, footprints=True)
min = maze.agent(m, randrow2, randcol2, footprints=True, color = maze.COLOR.red)
```

Step 2: Implementing Game Tree & Search Algorithms

Create a class file `GameSearch.py` to implement the following search strategies:

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

Consider the following notes to help with the implementation:

1. Implementing Game Tree: You are not creating a game tree of links (as learned 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 maze. Each node 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’,[],[]],[]]]
   Where, nodes a, b, c, d, e, f,.. represents the maze configuration
   ```
    a. Your game tree will need the `maze_map` to know open spaces vs. closed walls.
    b. Agent can only navigate through open spaces (i.e. agent cannot jump over the wall)
2. 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.
    a. 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.
3. 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.

Step 3: Passing inputs

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

- Game Initiation: Your program *`MazeRunner.py`* will accept input arguments from the command line in the following format:
  `MazeRunner.py[player][searchmethod][size]`
  - `[player]` choose if AI is player "1" or player "2". You are playing against a Human player. The 1st player is always 'MAX' and the 2nd player is always 'MIN'
  - `[searchmethod]` can be MM (for minimax) or AB (for minimax with alpha-beta pruning)
  - `[size]` maze size options are `10` (10 x 10) or `20` (20 x 30)

  Examples:
    - `1 MM 10`
    - `2 AB 20`
- Game Moves: Once you start playing the game you will accept player (Human player) moves and display AI moves (output - via console). Here is an example:

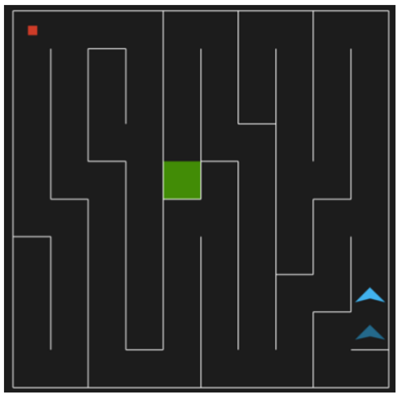
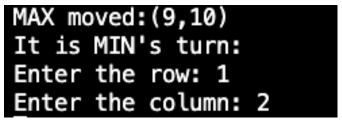


Figure 2: MAX (AI) makes a move



```
MAX moved:(9,10)
It is MIN's turn:
Enter the row: 1
Enter the column: 2
```

Figure 3: Console output/input for each player

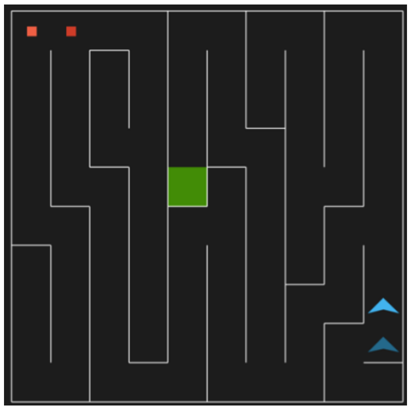


Figure 4: MIN (Human) makes a move based on user input

Step 4: Generating Outputs

Your program will generate 3 different outputs – GUI, console interactions & and a readme file.

1. GUI Output: Trace each player's move (turn-taking) from their initial state to the terminal state.
   Note: Game ends when a player reaches the goal state.
2. Console Output: Your program must display moves for AI player on the console (see Figure 3) and announce the winner (i.e. AI beats Human or Human beats AI)
3. `Readme.txt:` In your readme file you will include the following information:
   - Evaluation function: formula for calculating the utility value of non-terminal leaf nodes.
   - For each maze `size` report (for at-least 2 different depth levels):
     - Number of nodes expanded (for both MM & AB)
     - Depth-level for look-ahead (for both MM & AB)

Submission Guideline: Zip & upload the following files to the assignment submission link:

- `GameSearch.py`: Class file that implements Minimax & Alpha-Beta search strategies.
  - Note: You can have additional source code file(s) to build the game tree (optional).
- *`MazeRunner.py`*: The main source code file that accepts the input and connects your pyamaze to the search strategies.
- `Readme.txt`: Formula for utility function & output for each of the algorithm as explained in the output section.