Project properties:

This project includes 3 mini-games implementing 3 of the most famous AI algorithms.

1. A pathfinding game in a maze -> implementing simple A*
2. A N-Queen solution finder -> using Simulated Annealing
3. A simple Tic Tac Toe game with Visuals -> implementing Minimax with pruning

1: Maze pathfinder

This Maze problem is solved using A* algorithm. The algorithm applies A* node expanding and searching pattern and finds the optimal

path to a given goal starting with a given state.

Note: The maze world, starting state and goal state are fully changeable depending on the user input.

Note: The blocks of maze world can also be mutable. (using a walls.txt file in CWD)

Note: the heuristic implementation of A* is done using Manhattan distance

Code explanation =>

The program includes 4 functions:

maze initiator - Maze_init(number of rows, number of columns, goal, walls)

successor finder - Successor(Maze, node, blocked)

A* - A_star(maze, blocked, start, goal)

main()

in the following you can find the use of each function:

1. Maze_init(number of rows, number of columns, goal, walls) -> returns an initialized maze world and set of blocked pathes.

   Note: also initiates every node with [row, col, father, hx, gx, fx] only the values of row and col are assigned.

2. Successor(Maze, node, blocked) -> returns possible successors of a given node

3. A_star(maze, blocked, start, goal) -> returns True if a possible path is found and False otherwise. Also prints the path in 2 formats

4. main() -> gets required inputs, standardizes the walls format and runs the program

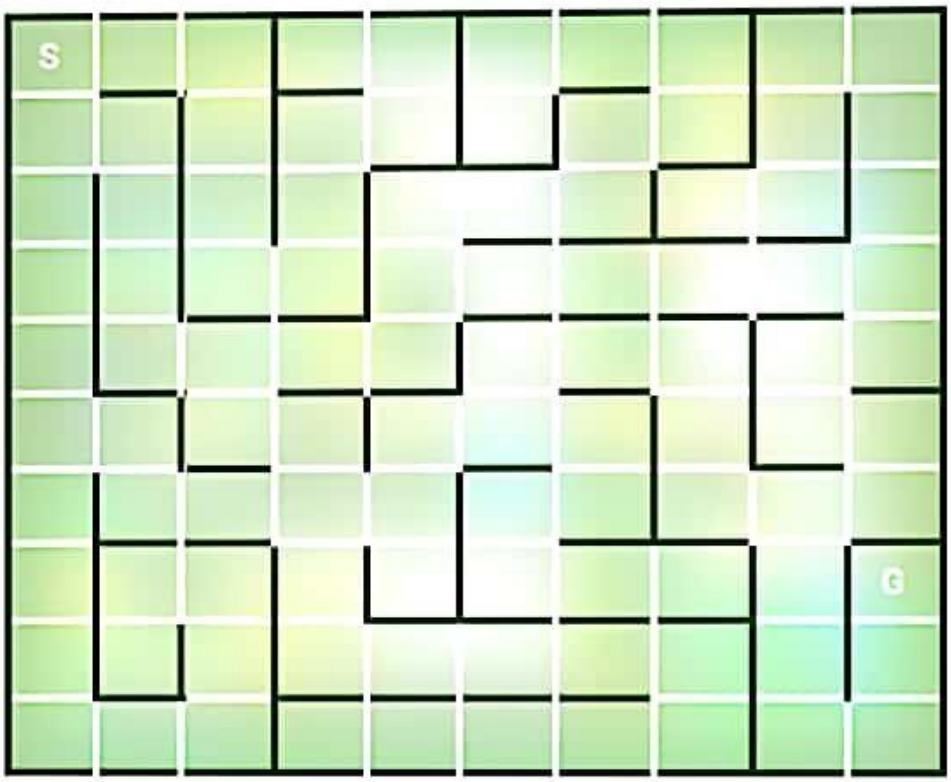
Note: walls.txt formatting must be like following:

first_row,first_col | second_row,second_col

this puts a wall between the two given states

any new restriction must be wrote in a new line

visuals:




```
PS C:\Users\asus\Desktop\AI-Alghoritmes\PathFinder> & C:/Users/asus/AppData/Local/Programs/Python/Python39/
please enter number of rows => 10
please enter number of columns => 10
please enter the row number of Starting state => 0
please enter the column number of Starting state => 0
please enter the row number of Goal state => 7
please enter the column number of Goal state => 9
Before you continue, make sure you have added the walls position in walls.txt file in CWP


[0, 0] => [1, 0] => [2, 0] => [3, 0] => [4, 0] => [5, 0] => [5, 1] => [6, 1] => [6, 2] => [6, 3] => [6, 4] =
> [5, 4] => [5, 5] => [4, 5] => [4, 6] => [4, 7] => [5, 7] => [6, 7] => [6, 8] => [7, 8] => [8, 8] => [9, 8]
 => [9, 9] => [8, 9] => [7, 9] = Goal

['start', 'down', 'down', 'down', 'down', 'down', 'right', 'down', 'right', 'right', 'right', 'up', 'right',
 'up', 'right', 'right', 'down', 'down', 'right', 'down', 'down', 'down', 'right', 'up', 'up', 'goal']
```

2: N-Queen solver:

The N-Queen solution is found through an implementation of SA. The algorithm tries to find a global minimum cost.

Note: cost is assigned to the number of conflicting Queens.

Note: the algorithm gets an initial Temperature and N of problem and returns a simple visualization of chess board. If SA couldn't find the solution returns failure

Note: Temperature scheduling can be done by either multiplication (*0.99) or subtraction (-0.01)

Code explanation =>

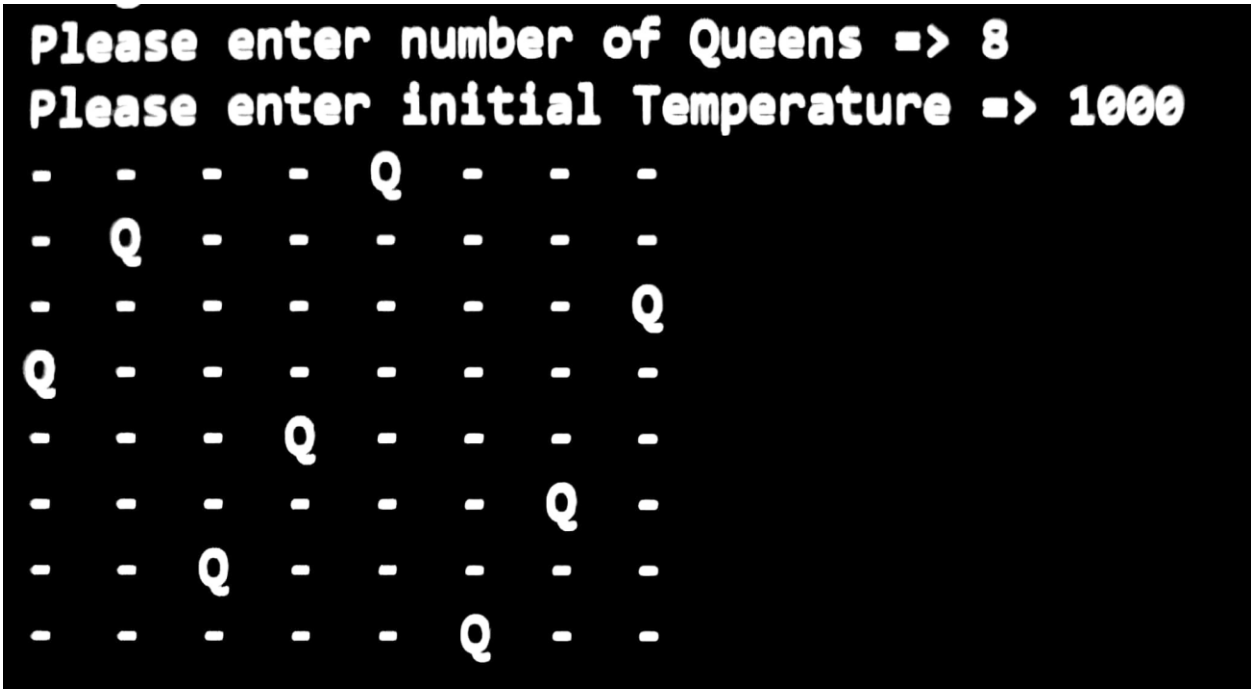
The program includes the following functions:

1. Board initializer - init_brd()
2. Cost calculator - cost(state)
3. SA - simulated_annealing()
4. Board printer - print_board(board)
5. Main()

Her is the description of each function:

1. init_brd() -> initializes a random board state with a Queen in each column
2. cost(state) -> returns the cost of the current board state
3. simulated_annealing() -> runs SA and prints the result
4. print_board(board) -> prints board
5. Main() -> gets required inputs and runs the program

Visuals:


Please enter number of Queens => 8
Please enter initial Temperature => 1000
- - - - Q - - -
- Q - - - - - -
- - - - - - - Q
Q - - - - - - -
- - - Q - - - -
- - - - - - Q -
- - Q - - - - -
- - - - - Q - -

3: Tic Tac Toe

The game contains a simple graphical and intractable display. It has two modes where either The player or the Computer makes the first move. The computer uses Minimax algorithm with alpha-beta pruning technique to beat the player.

Note: In this game, given the nature of tic tac toe, computer is unbeatable

Note: Running the program requires installation of “pygame”. It is necessary to make the app work

Note: Make sure that CWD is right

Note: Before you can start the game, you need to define the game mode.

Note: Computer is always “O” and minimizer

Code explanation =>

The program includes the following main functions:

1. Drawer functions - draw_grid() | display_message(content) | render()
2. Click function - click(board, turn)
3. Game final state checkers - who_won(board) | has_drawn(board)
4. Empty finder - emptyCells(board)
5. Minimax with pruning - AlphaBetaMM(board, depth, alpha, beta, player) | getScore(board)
6. Main()
7. And some other in-program calculator functions …

Functionalities and properties of regarded functions:

1. Drawer functions -> They Draw the visuals of game board
2. click(board, turn) -> Visualizes the moves on game board
3. Game final state checkers -> check if we have reached a game final state with current board states
4. emptyCells(board) -> finds and returns the valueless cells
5. Minimax with pruning -> applies the algorithm to determine computer’s next move

Visuals: