South Dakota School of Mines and Technology

Advanced Robotic Algorithms, Spring 2022

CSC 692 - M01

Homework # 1

This program implements the A* algorithm to find paths through a maze. Ideally, it is designed to read mazes from this website (https://keesiemeijer.github.io/maze-generator/); but it has an internal function to generate a random maze if one is not provided.

Modes of Usage

1) **python3 a_star_alg.py rand** —Randomly generates a maze and solves it.
2) **python3 a_star_alg.py <filename>** —Reads the maze from the file and solves it.

Four files have been provided in the zip folder to test out the algorithm (maze.png, maze_20x20.png, maze_64x64.png and maze_128x128.png). These mazes have been all generate and downloaded from the aforementioned link.

The main program consists of 3 subfiles - a_star_alg.py, gui.py and grid_generator.py.

In brief, **a_star_alg.py** contains all the code for implementing A* to solve a maze. It is the main point of entry and calls functions from grid_generator.py and gui.py.
grid_generator.py handles the translation of input maze .png(s) to graph style grids and random grid generation if an input maze is not provided.
gui.py as the name suggests handles the translation of grid data and path found to visuals.

Figure 1 shows the program operating in the random maze generation mode. **Figure 2**, **3** and **4** show the algorithm working in the maze input mode and solving maze_20x20.png, maze_64x64.png and maze_128x128.png respectively.

Figure 1. Random Generated Maze

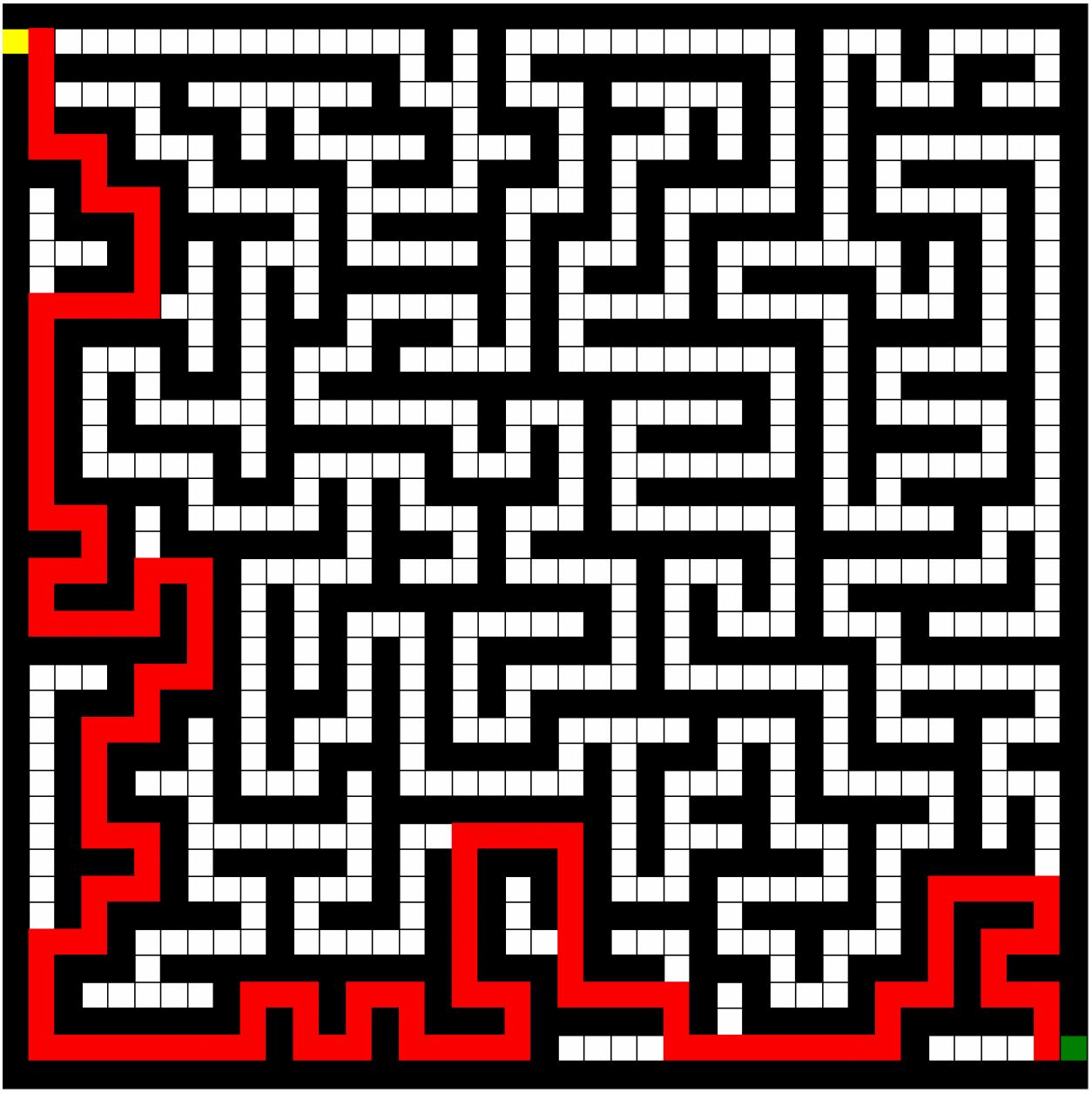

Figure 2. maze_20x20.png

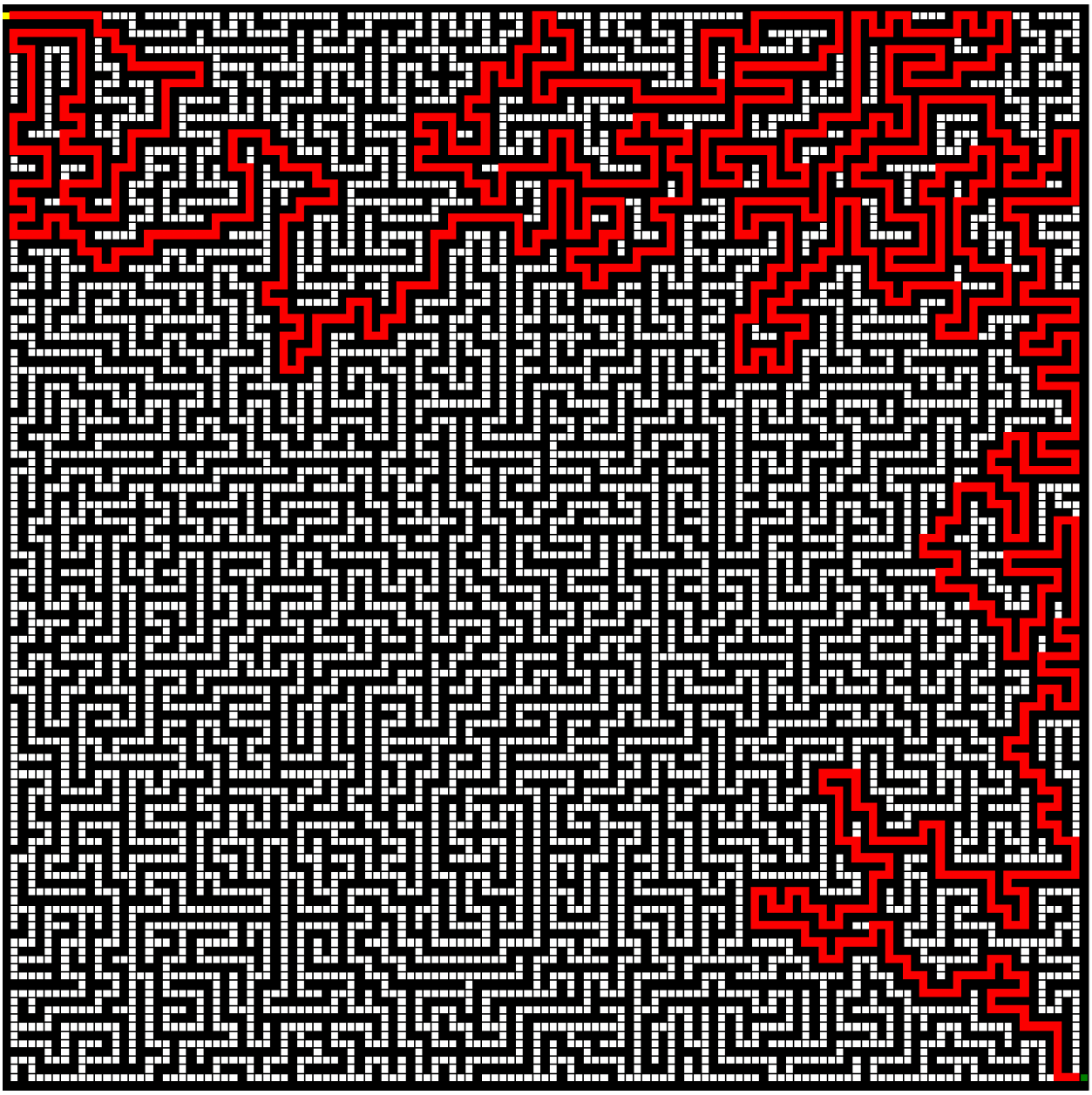

Figure 3. maze_64x64.png

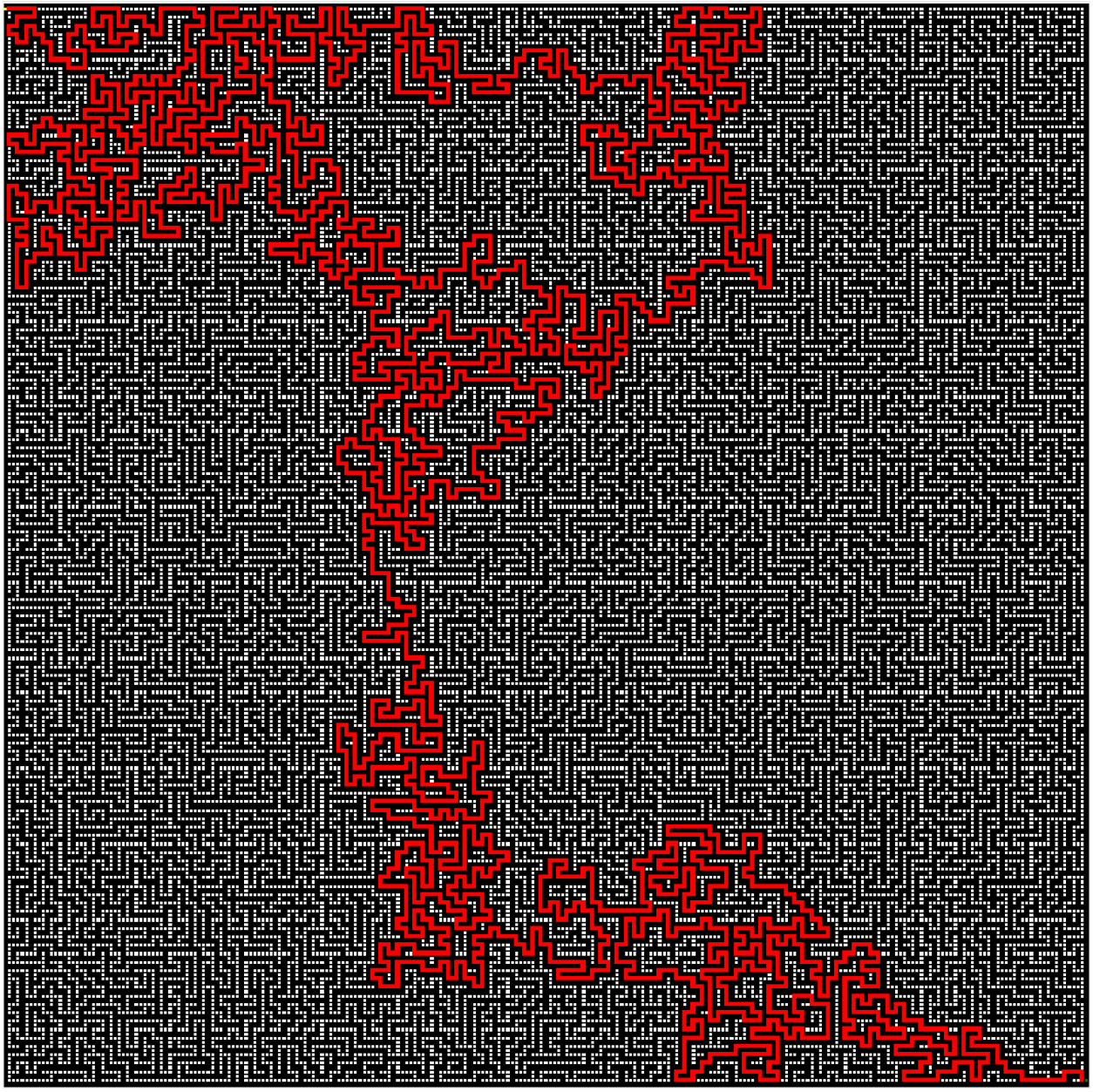

Figure 4. maze_128x128.png

General Explanation of the Algorithm

grid_generator.py creates a 2D grid that contains cells filled according to the following scheme:

-1 = obstacles
0 = free space
7 = start
8 = end
9 = point in path

For files that it inputs, it creates a grid map based on that same scheme.

** Most of the online maze generators represent each cell as 10-15 pixels wide and high. Testing has only been done for the following maze generating websites: **https://keesiemeijer.github.io/maze-generator/** and **mazegenerator.net** ; these 2 websites use a width of 10 pixels for each square. The former is preferred as it gives out more cleaner looking maze png(s).

gui.py translates this grid into visuals. As part of the above scheme, it draws black, white, yellow, green and red squares, respectively, for each position in the grid. The underlying package used for the gui visuals is **tkinter.**

a_star_al.py implements the A* algorithm in a textbook fashion. The underlying data structures used are a **Node struct** (described by 4 elements - current_index, previous_index, heuristic_distance, traversed distance), **a min heap** (implemented as an ordered list) and **a stack** to keep track of processed nodes.
The heuristic distance is the euclidean distance from current_node to the goal; previous_idx and current_idx are the cartesian co-ordinates of the ancestral node and current node.
The algorithm terminates if the node being processed is the end node or if the min heap of unprocessed nodes is empty - the end goal cannot be reached.

TO-DOs and Other Improvements

- Expand the grid input function to recognize different pixel to node ratio (thereby expanding its to input files from other websites).
- Implement a graph based approach to A* instead of a grid based approach.
- General touch ups to GUI. Currently the gui refreshes way to often resulting in larger instances of grids to be very laggy
- Improve the random grid generating algorithm so it generates fairly complicated mazes - One possible approach would be to start with a solved path and build a grid around it.
- Improve start and end point recognization from a totally unknown maze. Currently, the website, provided has the start on the top left and end on the bottom right