Cmpe 160 – Path Finding Visualization Program

GitHub: https://github.com/serhattay/Path-Finding-Visualization

Participants
Omer Ozan Mart   Student ID: 2022400267
Serhat Tay   Student ID: 2022400201

Introduction

The Pathfinding Visualization Program is a Java-based application designed to visualize the process of finding the shortest path on a grid-based map from a starting point to a destination point using the A* algorithm. This report provides a comprehensive overview of the program's functionality, its implementation details, and a detailed explanation of the A* algorithm's working.

The program consists of four main classes: Main, Graph, Tile and Map. These classes work together to create a graphical user interface, generate, and display the grid-based map, and execute the A* algorithm to find the shortest path.

The Main class initializes the program's graphical user interface using Java's StdDraw library. It sets up the canvas size, scales, and enables double buffering for smooth rendering. The user interacts with the program by selecting a destination tile using the mouse input. The Graph class is responsible for generating the grid layout and managing the map's functionality. It provides methods to generate both default and random grid layouts with obstacles, character location, and destination tile. The Map class handles the drawing of the grid, obstacles, character, and destination tile on the map. The Tile class represents individual tiles within the grid-based map. It encapsulates essential properties and functionalities necessary for pathfinding algorithms and map visualization.

A* Algorithm

The A* algorithm serves as the backbone of the Pathfinding Visualization Program, efficiently finding the shortest path from the character's location to the destination tile on the grid-based map. Let's explore how this algorithm operates step by step:

Initialization

Start Node: The algorithm commences by designating the starting node, which corresponds to the character's initial location on the map. This starting node is added to a priority queue, with its cost of reaching set to zero.

Priority Queue: A priority queue is initialized with the starting node. This data structure is significant in facilitating the exploration of nodes with the lowest total cost first, thus guiding the algorithm towards the most promising path.

Travelling

Priority Queue Loop: The algorithm iteratively explores neighboring nodes while the priority queue remains non-empty. This iterative process ensures that the algorithm carefully evaluates potential paths to the destination.

Node Selection: At each iteration, the algorithm selects the node with the lowest total cost from the priority queue. This prioritization ensures that the algorithm efficiently progresses towards the destination tile.

Destination Check: If the selected node corresponds to the destination tile, the algorithm terminates, as the shortest path has been successfully identified.

Neighboring Nodes: For each neighboring node of the selected node, the algorithm calculates the new cost to reach that node from the starting node. This calculation involves considering both the cost incurred thus far and the heuristic estimate of the remaining distance to the destination.

Cost Update: If the neighboring node has not been visited previously or if the newly computed cost is lower than its current cost, the algorithm updates the cost to reach the neighboring node. Subsequently, the neighboring node is added to the priority queue, marked as visited, and linked to its previous node.

Path Reconstruction

Once the algorithm completes its traversal, the shortest path from the starting node to the destination node is reconstructed. This reconstruction process involves tracing back from the destination node to the starting node using the pointers established during exploration.

To provide users with a comprehensive understanding of the computed shortest path, the program visualizes the path on the map. This visualization includes drawing lines and circles to outline the path's route and animating the character's movement along the path. By animating the character's progression, users can observe the dynamic nature of the pathfinding process in real-time.

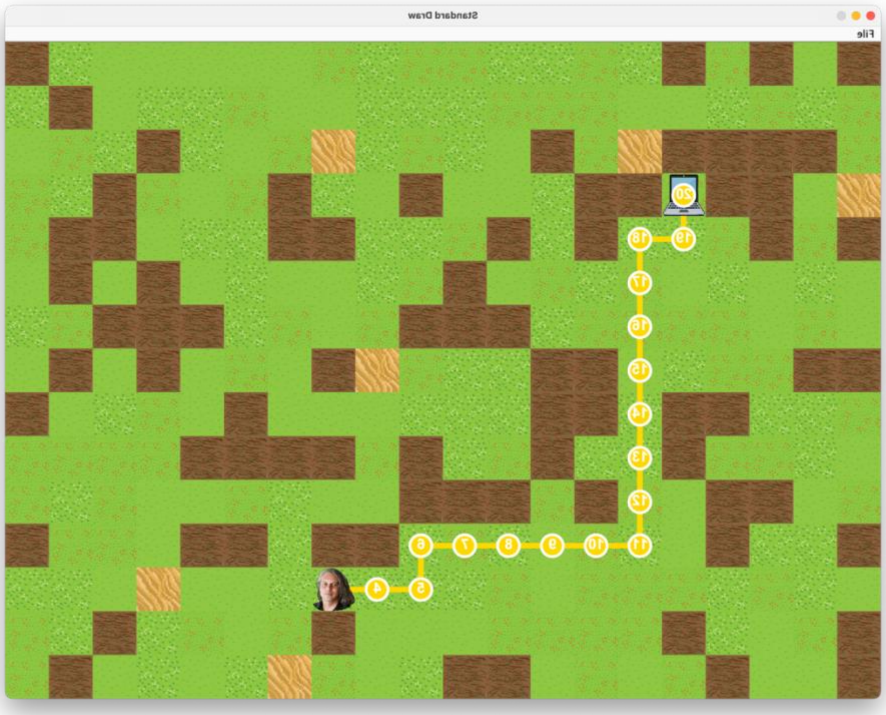
Game Videos and Images

Version 1: https://youtu.be/NpMQyyVk3xs

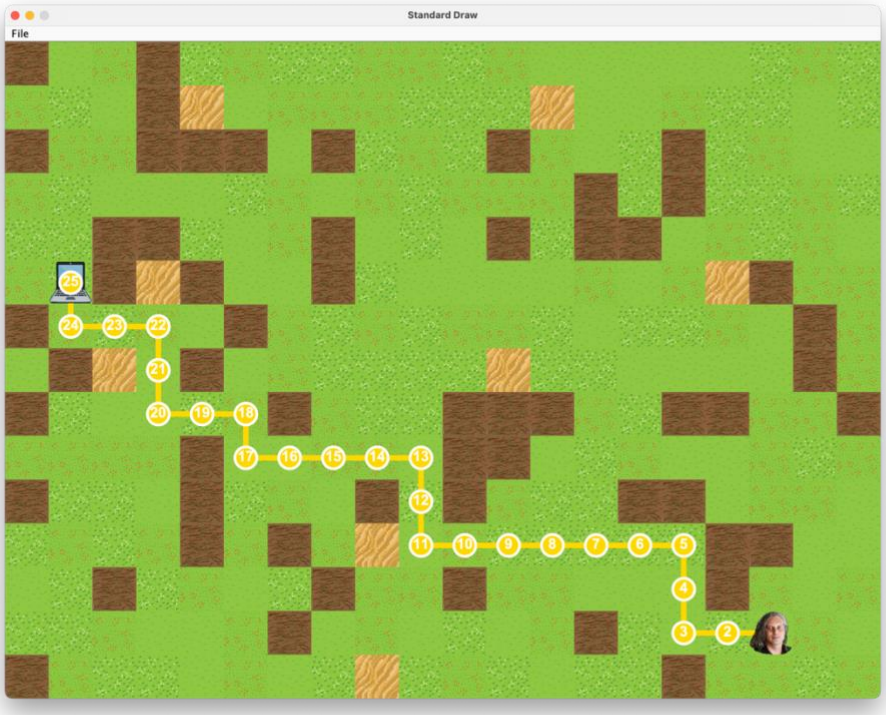
Dark-gray tiles represent obstacles while orange tiles represent tiles that have higher cost values.

Version 2: https://youtu.be/UN7pCDU7STA

Case 1:

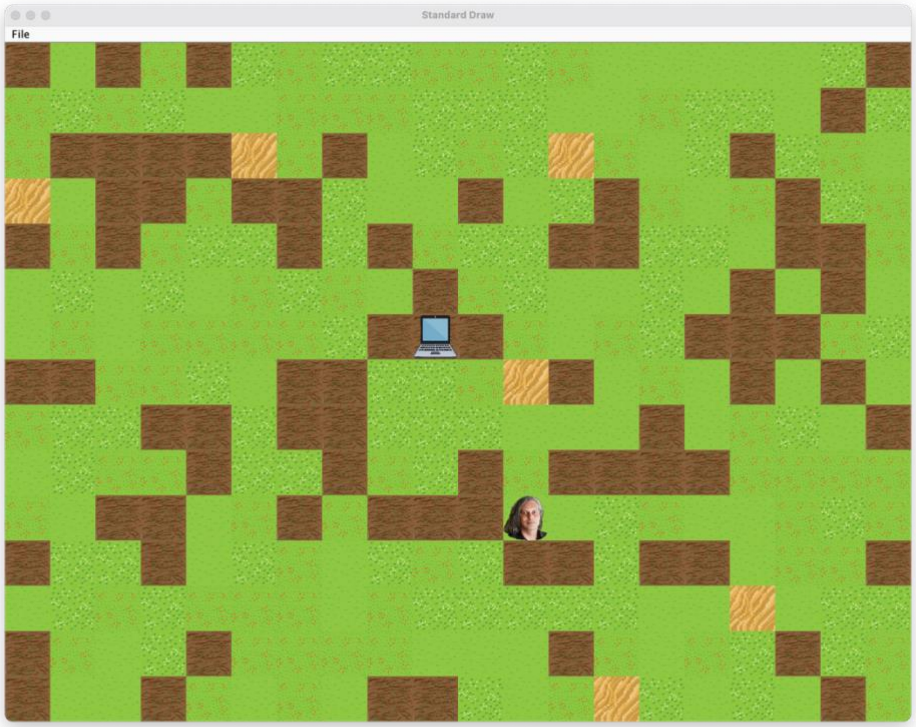


If user clicks to a point where it is possible to reach, program will show the shortest path.



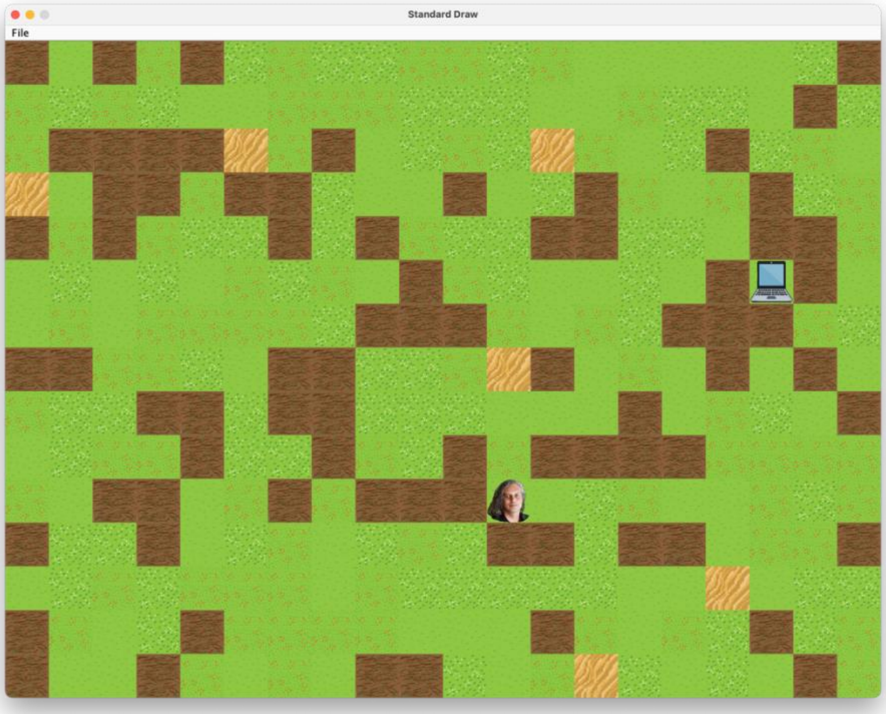
Another example for case 1 is shown here.

Case 2:



If user clicks to an obstacle, program will not draw any shortest path since it is impossible to reach.

Case 3:



If user clicks a point where it is impossible to reach, program will not draw any shortest path.

References and Sources

http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html
http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html
http://theory.stanford.edu/~amitp/GameProgramming/ImplementationNotes.html
http://theory.stanford.edu/~amitp/GameProgramming/MapRepresentations.html
https://en.wikipedia.org/wiki/A*_search_algorithm
https://youtu.be/A60q6dcoCjw?si=FmW4qN2ZWMMCnDPP
https://www.redblobgames.com/pathfinding/a-star/introduction.html