Calvin Le

Senior Project

12/18/20

Technical Report: Shortest Path Visualizer

Abstract

Shortest path algorithms such as Dijkstra's, A* search, greedy best-first search, and many more are limited in terms of visualization. These algorithms can be a difficult process for one to develop because of the complexity and time-consuming tasks required to create a functioning application. Therefore, the interactable shortest path visualizer is aimed to increase student participation by developing an application that display the process in which various shortest path algorithm works. The application is able to display two algorithms: Dijkstra's and A* search in addition to other tools to help aid the user on creating, deleting, and clearing the path workspace.

Introduction

The Shortest Path Visualizer (SPV) is a Python-based application that utilizes GUI and path finding algorithms to display the shortest path from a starting and end point. The application features other tools that help guide the user in creating and deleting paths within the grid. There are two frames that open during launch: the settings menu and grid workspace. The user is able to interact with both windows. The settings menu allows the user with additional tools such as clearing and changing the size of the grid. The grid workspace allows users to set start and end nodes and creating or deleting barriers.

Path-Finding Algorithms

The algorithms used for this application utilizes Dijkstra's and A* search methods. Each method finds the shortest path but functions differently in how they can find the end node.

Dijkstra's Algorithm

The Dijkstra's shortest path first algorithm picks unvisited nodes and calculates the distance between the starting and end node. Starting with the start node, the algorithm selects all neighboring nodes and iterates this process until the end node is found. Since each node has no weight, all nodes must be considered during the path finding process. The shortest path is drawn after the end node is found.

A* Search Algorithm

A* search algorithm functions similarly to Dijkstra's algorithm except with the inclusion of a heuristic function to guide the search. The A* search algorithm only consider paths that is optimal. The algorithm calculates the distance from the start and end node and makes an informed decision in scanning neighboring nodes. The process is repeated until the end node is found and the application can generate the shortest path.

Design Specifications

The application is entirely software-based. The selected language used for the development of this application is Python because they provide packages designed for GUI development and image processing.

Interactable Grid

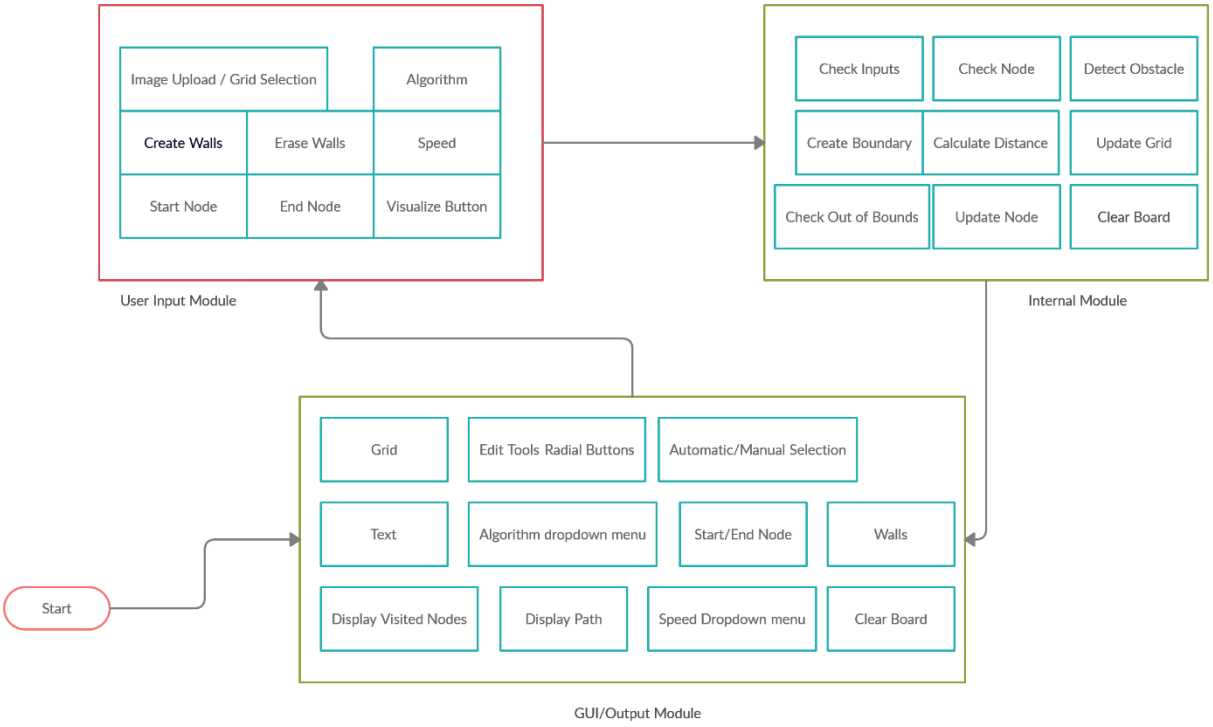
The program will use a grid of nodes to represent the map, each node has up to four traversable edges: up, down, left, right. One node will be the start node and another the target node. Also, a

node can be marked impassable by the user. For the A* search method, each node holds a heuristic value based on its estimated distance from the target. The user will be able to clear and change the size of the grid. The grid window features the grid of nodes visible to the user. The user can interact with the grid using left mouse and right mouse click. If there are no start and node nodes seen on the grid, the user can place the start node using left mouse click anywhere on the grid and set the end node by using the same process. After the start and end node are set on the grid, the user can add barriers by clicking or holding down left mouse button and delete nodes by clicking or holding down right mouse button.

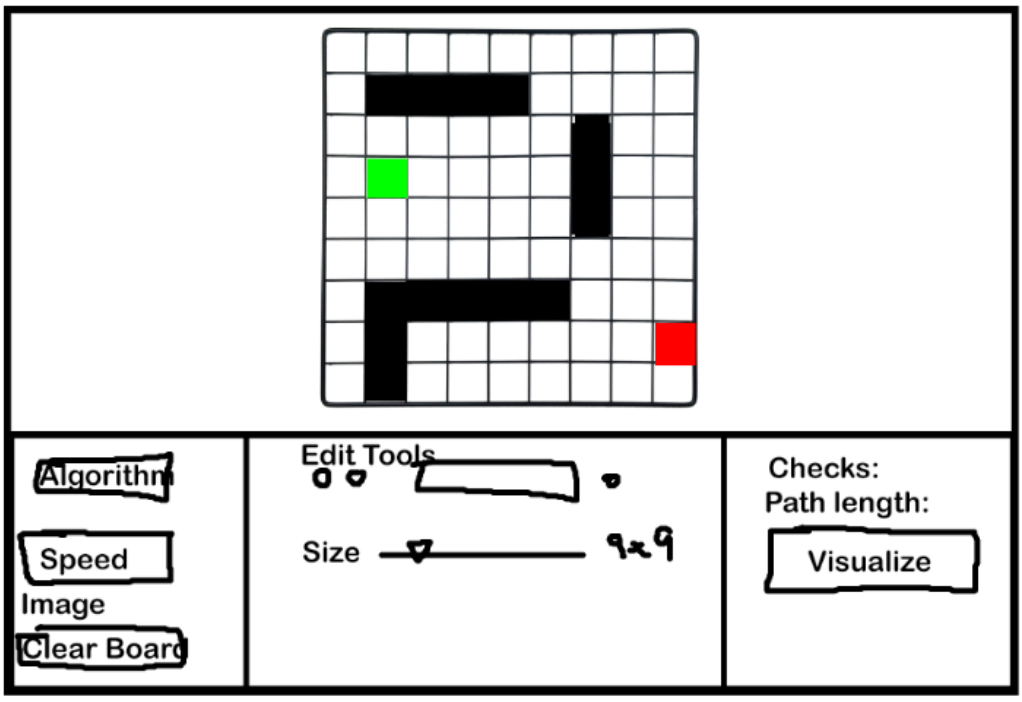
Settings Menu

The applications settings menu provides users with tools needed to interact with the grid. The settings menu includes metrics such as the number of checks and the shortest path distance after visualizing the path. The settings menu also includes a clear grid and change size button to help the user make changes to the grid window. The user is also able to change search algorithms by selecting the dropdown menu and selecting the algorithm of choice. By default, the selected algorithm is A* search. The user can see the algorithmic process by clicking on the visualize button. In addition, the user can select an image of a PNG file and convert it into the grid but this feature is not fully implemented.

Module Interaction Diagram

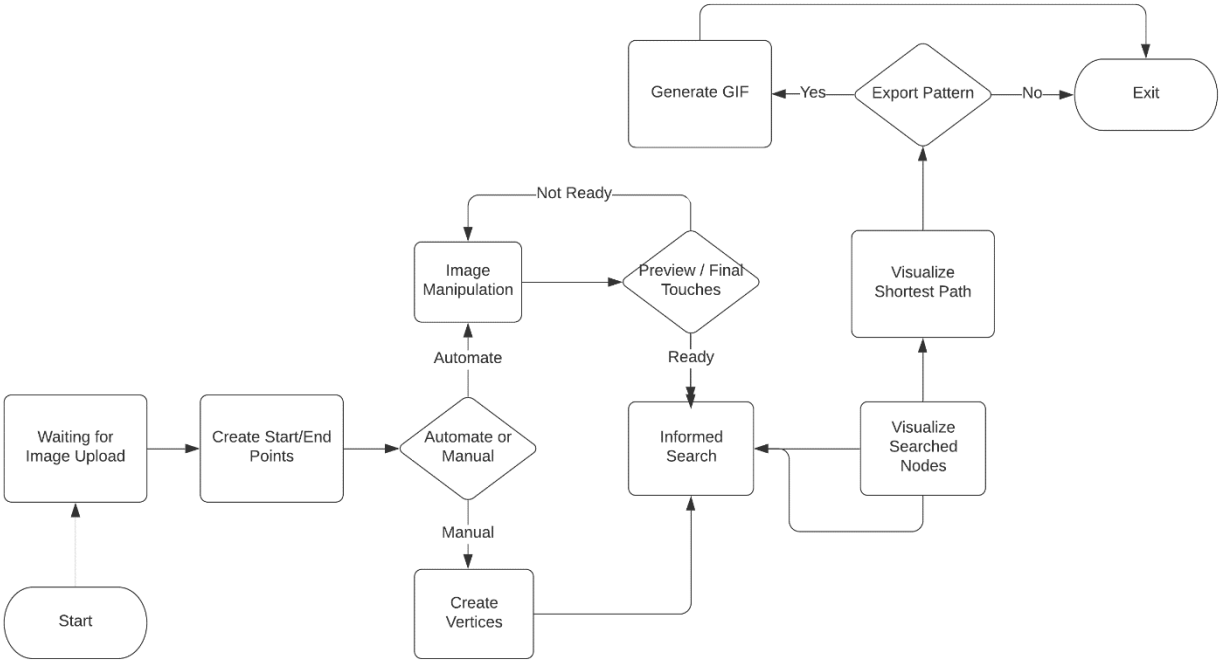


Initial User Interface Design



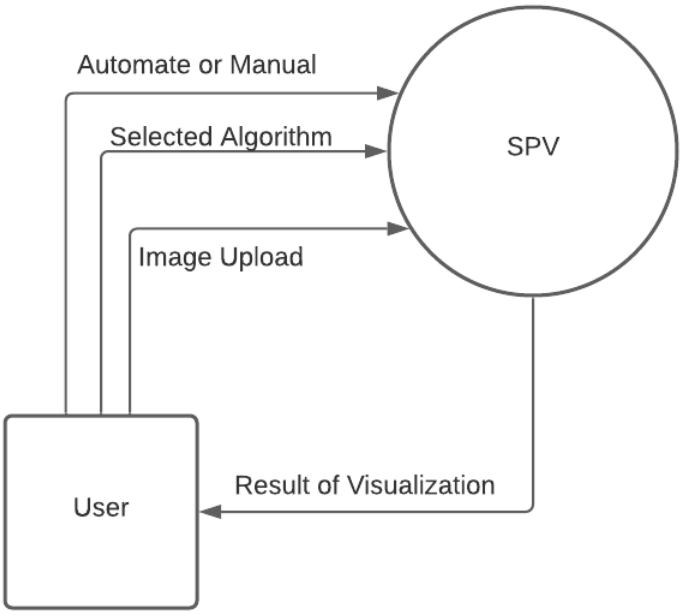
The application was intended to be developed using one window but the decision to split the settings and grid into two separate parts was necessary to allow the user to change the size of the grid and allows development to be modular.

Initial Conceptual Model

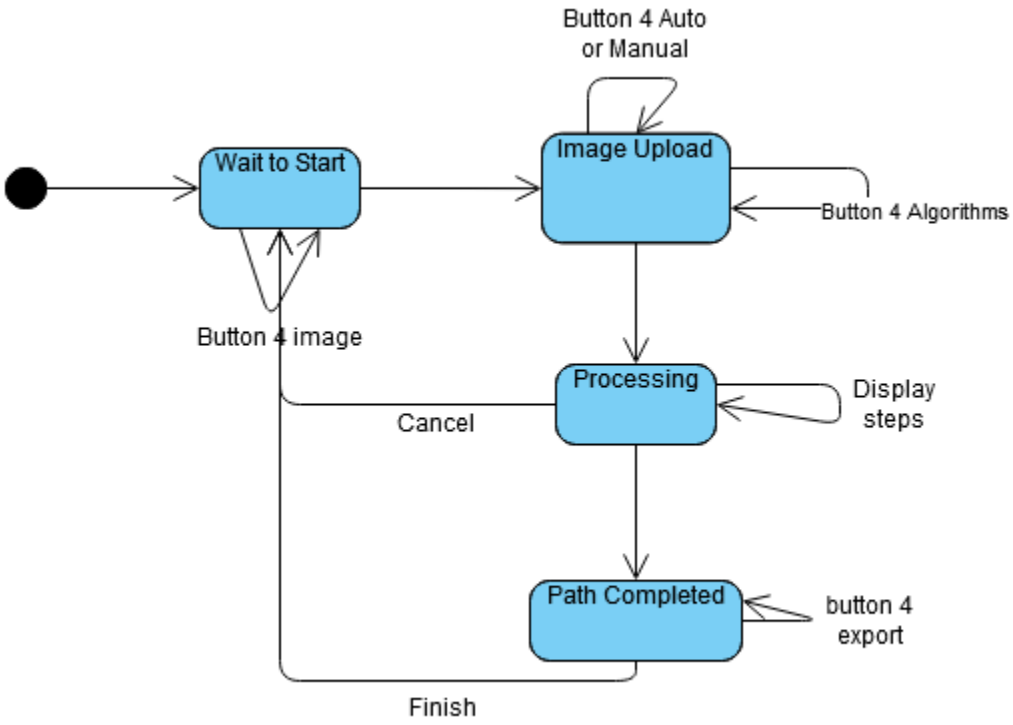


The application was intended to process images but due to time constraints this implementation never reached the final stages of development.

System Context Diagram

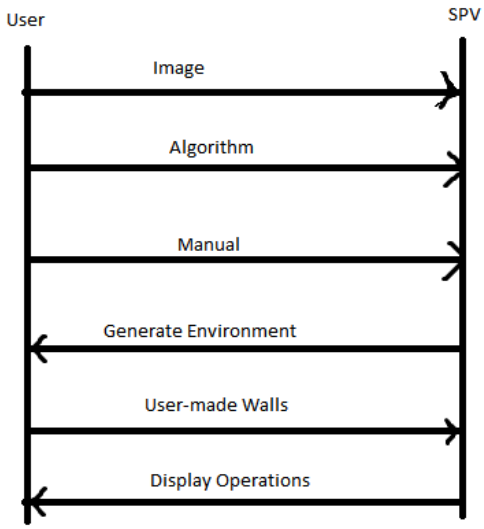


Transition State Diagram

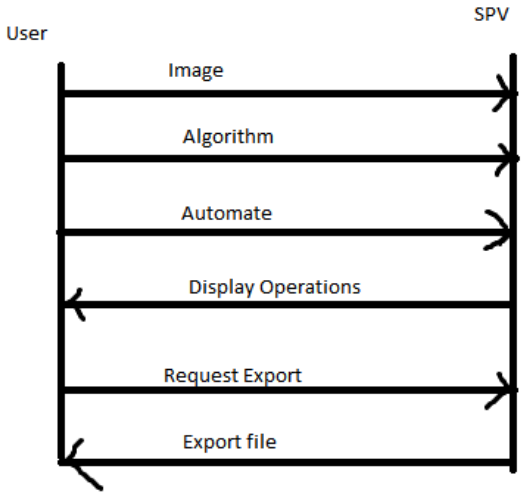


Sequence Diagrams

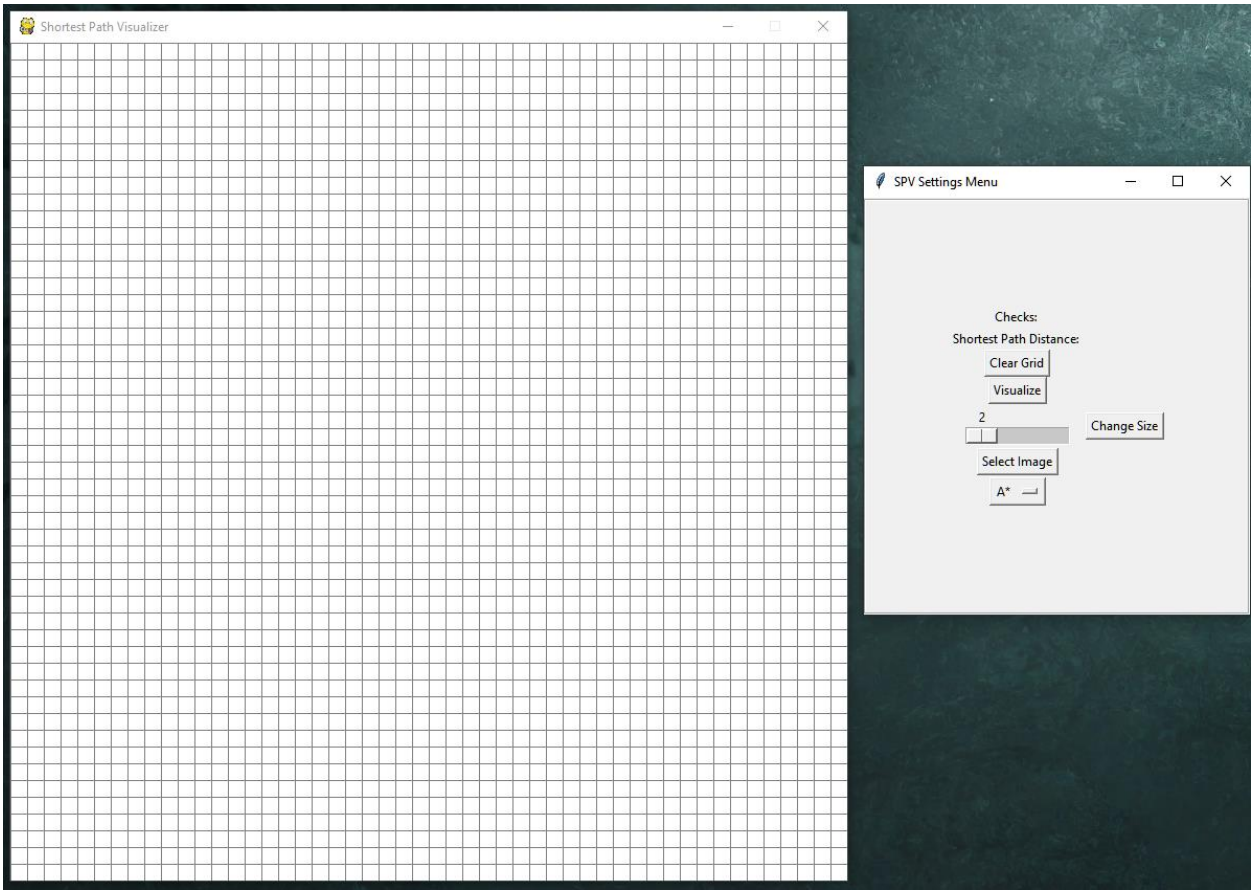
Scenario 1: Manual Selection



Scenario 2: Automatic Selection

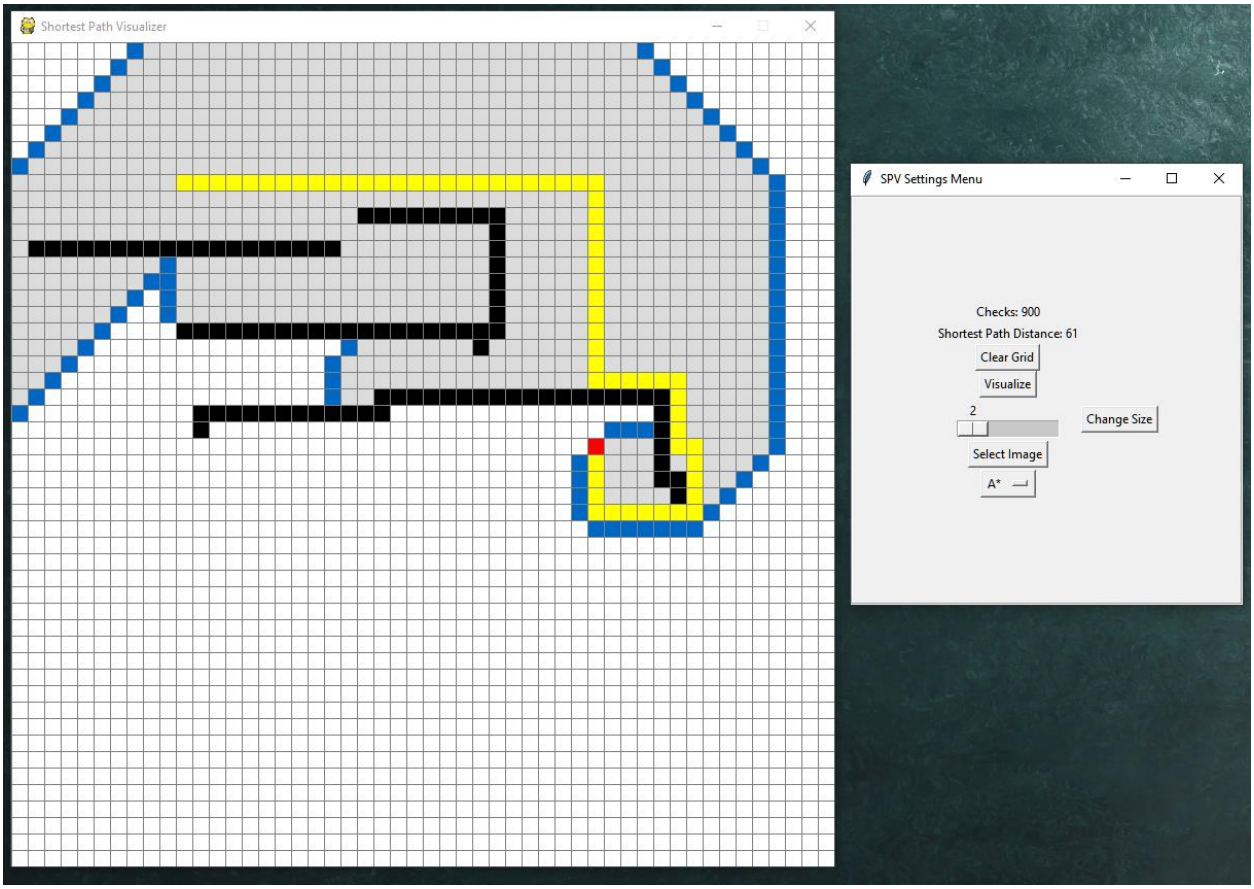


Final UX Design



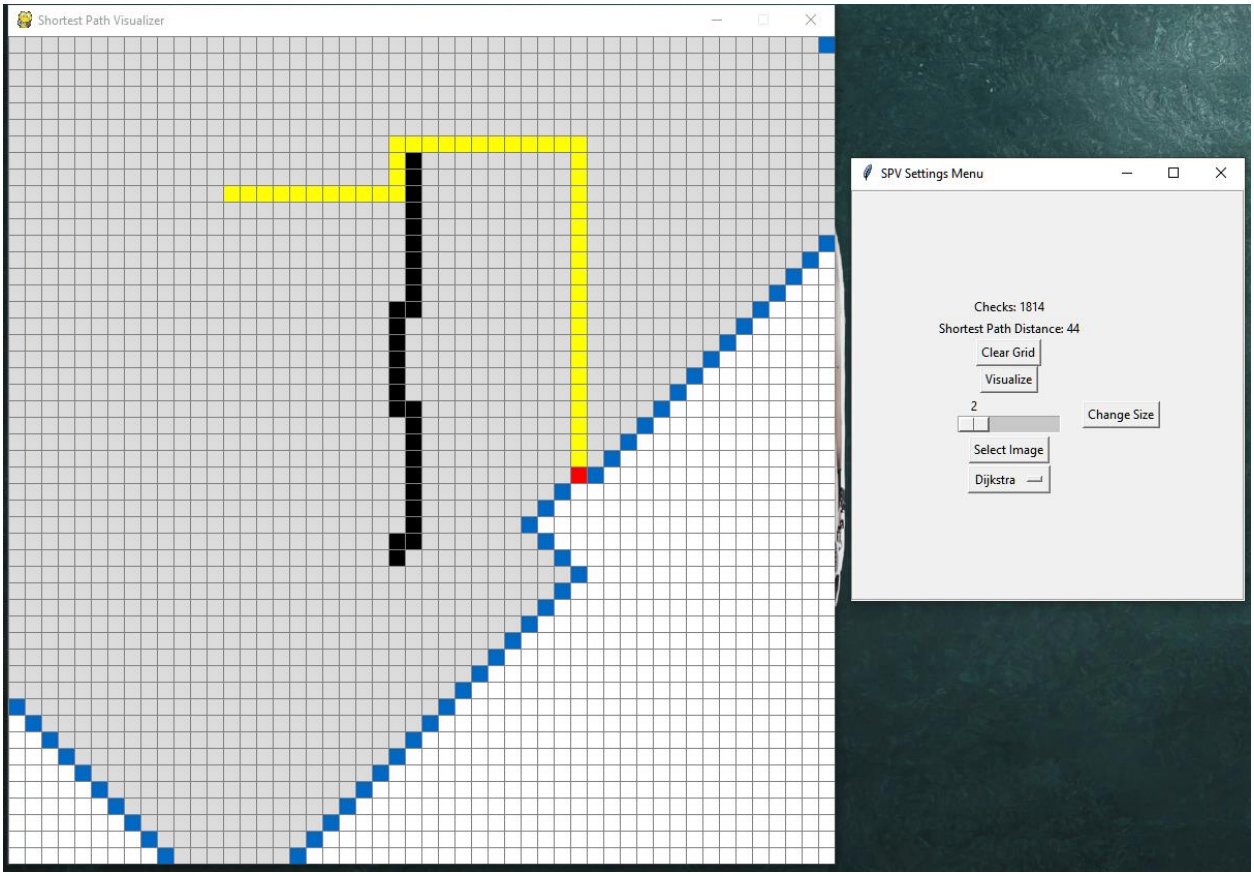
Grid is created using Python's pygame package. The grid is updated constantly when there are changes within the algorithm and changes in settings menu. The settings menu uses Python's tkinter package to display text and buttons. Each button has its own function that interacts with the grid.

Use Case Scenario 1



A use case showing the A* search algorithm. The above image shows the shortest path (yellow) to the end node seen in red. Barriers are made black, grey nodes indicate nodes already considered and blue nodes indicate nodes to be considered.

Use Case Scenario 2



A use case showing the Dijkstra's path-finding algorithm.

Test Cases

User Function Name	User Function Description	Expected Results	Self-Testing Results	Notes
Application Launch	Display the programs GUI for grid and settings	Working resizable panel and buttons	Functional	Final Demo
Setting start and end nodes	The user can click anywhere on the grid to set start and end nodes	Green node = Start node Red Node = End Node	Functional	Final Demo
Creating Barriers	User can create barrios using left mouse click anywhere on the grid	Black node = Barrier	Functional	Final Demo
Deleting Nodes	User can delete colored nodes including start, end and barrier nodes using right click	Deleting nodes will revert the node back to white.	Functional	Final Demo
Clearing Grid	The clearing grid button will erase any changes made on the grid and display a blank grid.	All colored nodes are removed from the grid	Functional	Final Demo
Visualize Button	The visualize button will read in the selected algorithm from the dropdown menu and display the algorithm processes on the grid.	The grid will populate with colors and show pathfinding steps. The algorithm ends when a path is found and drawn.	Functional	Final Demo
Changing Grid Size	The user can change size of grid using the slider and change size button	Upon selecting the size and clicking the change size button, the application	Functional	Final Demo

		should clear the grid and change the number of nodes displayed.		
Select Image	The user can select a png file of a maze of their choice and the application should convert the image into an interactable grid	The application should only read in PNG files and draw the image into the grid.	Not fully functionable, only certain images are able to fully convert.	Final Demo.
Display pathfinding metrics	The application should display the number of checks and the distance of the shortest path in the settings menu.	The checks and shortest path distance label is updated after visualization ends	Functional	Final Demo
Selecting Algorithm	The user should be able to select between two different pathfinding algorithms: A* and Dijkstra's.	A dropdown menu will display the selected algorithm and visualize with the selected algorithm	Functional	Final Demo

Conclusion:

The shortest path visualizer application is successful in displaying multiple pathfinding algorithms and provide users with interactable tools to customize the grid. The application does not however be able to fully render images onto the grid without user modifications. Some bugs are still present in the program such as changing the size of the grid will display stretched out nodes on the edges.

References

Ably, Thaddeus, et al. “Dijkstra's Shortest Path Algorithm.” *Brilliant Math & Science Wiki*, brilliant.org/wiki/dijkstras-short-path-finder/.

“A* Search Algorithm.” *GeeksforGeeks*, 7 Sept. 2018, www.geeksforgeeks.org/a-search-algorithm/.

Sourabh_SinhaCheck out this Author's contributed articles., et al. “Find and Draw Contours Using OpenCV: Python.” *GeeksforGeeks*, 29 Apr. 2019, www.geeksforgeeks.org/find-and-draw-contours-using-opencv-python/.

“Edge Detection.” *Edge Detection – Image Processing with Python*, datacarpentry.org/image-processing/08-edge-detection/index.html.

Mortoray, Edaqa. “Basic Pathfinding Explained With Python.” *Codementor*, www.codementor.io/blog/basic-pathfinding-explained-with-python-5pil8767c1.

johnphilipjones. “Python: Accessing the Coordinate Position of a Mouse Click.” *YouTube*, YouTube, 22 Feb. 2019, www.youtube.com/watch?v=XC4eJQCem_0.

“Tkinter Course - Create Graphic User Interfaces in Python Tutorial.” *YouTube*, YouTube, 19 Nov. 2019, www.youtube.com/watch?v=YXPyB4XeYLA.

mrcordiner. “Game Board with 2D Array / Processing + Python.” *YouTube*, YouTube, 13 Feb. 2015, www.youtube.com/watch?v=nsLTQj-l_18.

“Pathfinding Algorithms.” *YouTube*, YouTube, 5 Dec. 2014, www.youtube.com/watch?v=X3x7BlLgS-4.

user11676515user11676515, et al. “How to Read a Maze from an Image and Convert It to Binary Values in Python.” *Stack Overflow*, 1 Oct. 1968, stackoverflow.com/questions/57610416/how-to-read-a-maze-from-an-image-and-convert-it-to-binary-values-in-python.