[Pathfinder]

Testing and Evaluation Report

Aidan Grafenauer Parker

2148 words not including 407 word beta test feedback and 444 word testing table

Contents

Contents 2

Important Terminology 3

Testing 4

Evaluation 11

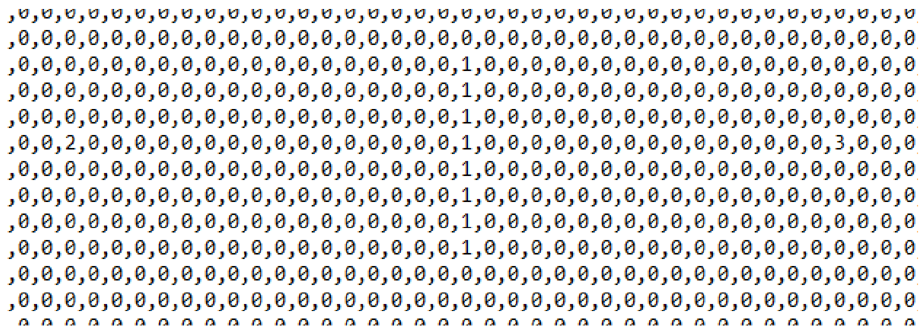
Important Terminology

- *Grid* refers to the array of *cells* that make up the majority of the window.
- A *node* refers to either the start cell or the end cell. A pathfinding algorithm will attempt to find the shortest path between these two nodes.
- A *wall or barrier block* refers to the black cells that cannot be traversed by the pathfinding algorithm.
- A *visited* cell refers to the cells that were explored by the pathfinding algorithm but not determined to be an optimal move.
- The *shortest path* cells are a series of cells that represent the optimal path as determined by the pathfinding algorithm.

Testing

Module Test Data

Primary modules such as the ones included in Module Testing were manually tested with a carefully thought out set of test conditions that encompassed all possible edge cases. Modules were tested manually because the sheer number of possible states that the grid had made it impossible to automatically test every state. Instead, only significant states such as edge cases and common use cases were tested.



```
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

An example of a grid that could be used for testing.

When a module was tested that involved the grid, it was ensured that test inputs would include at least four distinct variations of the following grids:

1. Grids containing only nodes
2. Grids containing nodes and blocked cell
3. Grids containing nodes, visited cells and shortest path cells
4. Grids containing nodes, visited cells, shortest path cells and blocked cells.
5. All of the above having positioned the nodes, blocked cells, visited cells or shortest path cells around the edges of the grid.
6. All of the above having positioned the nodes, blocked cells, visited cells or shortest path cells in the corners of the grid.
7. All of the above having positioned the nodes, blocked cells, visited cells or shortest path cells around the edges of the grid and in the corners of the grid.

Integration Testing

Integration testing was completed during development, most if not all issues were fixed for the final version of the software.

Functionality being tested	**Test Input / How it was tested**	**Expected Output**	**Output matches expectation (%) + notes**
Clearing the grid and Resetting the grid	When testing this functionality, the grid was set up in various ways to check whether clearing / resetting it would result in the correct output. Test grids included all 7 of the states mentioned above.	Grid should be cleared either fully or cleared except for the walls and nodes.	100%
Saving grids	The grid was saved in various states such that the contents of the save file could be later tested. The test grids that were saved included the states mentioned above.	CSV file downloaded in the correct format, 0 for empty cells, 1 for walls, 2 for a start node and 3 for an end node.	100%
Importing Saved grids	The grids that were tested for saving were imported to thoroughly test the capabilities of the importing feature.	CSV file read in and displayed on screen.	43% as described in the *testing during development* section below, the save functionality had an issue which caused imported some saved files to be invalid.

The pathfinding functionality	Each pathfinding algorithm was tested on a variety of grids. To verify the integrity / correctness of the results, the output (ie. the visited cells and shortest path) was directly compared to other simulations using a similar visualisation system. This provides a concrete way of knowing if the pathfinding algorithm produced the correct output. Test grids included the 7 states mentioned above.	Visited cells displayed on screen. Shortest path is found and displayed also. Visited cells and shortest path should be identical (or almost identical) to another similar simulation.	100% as described in the *module testing* section below, the A* algorithm was very slightly flawed in the way it visualised the visited cells but this was not deemed to be a bug, perhaps just a difference in the distance heuristic used or some other difference in the way cells are weighted.
Live pathfinding functionality	This was tested around the same 7 types of grids mentioned above. The live pathfinding was observed when dragged around walls, through areas that could not be traversed etc.	The live pathfinding should find the shortest path in all situations except where there is no valid path in which case the path should simply disappear until one is found again.	100%
Information tab	The information feature was tested with each of the four algorithms. This is easy to test since there are only four possible inputs.	The dropdown should display an information box depending on the algorithm that was selected from the dropdown.	100%

Bug fixes as the result of testing during development.

Testing during development involved an identical process to testing post development. However, there were bugs that required fixing in this stage. The majority of the following issues were located through the formal testing process mentioned above.

The following table is to specifically highlight significant issues that were encountered during development.

Module	**Specific Issue / Implications**	**How the issue was found / fixed.**
save_grid, import_grid	If visited cells or shortest path cells were present on the grid, the save_grid module would save those cells with a Null value. This meant that when the file was read in by the import_grid module, the software returned a runtime error. The issue has since been fixed. If this issue was present in production there would be negative impacts on the users experience as they would not be able to access key functionality of the software.	The series of grids mentioned in the *testing post development* section allowed for this issue to be found. From errors received during said formal testing, a fix was determined quickly and efficiently.
get_neighbors	The get_neighbors module had a bug where the blocked cells would be included in the array of valid neighbors. This would result in the pathfinding algorithms not recognising the walls as being untraversable and simply treating them like they were normal cells. This issue would have been detrimental to a users experience as it would have removed key functionality from the software but has since been fixed in the production build.	This issue was found through formal testing of the pathfinding algorithms. The majority of tests were very obviously deemed incorrect immediately. The error was more difficult to find, various modules that were likely responsible for causing the bug were investigated specifically. This bug took roughly 30 minutes to locate and fix the error.

a_star	This was a complex issue. The algorithm would return an array of visited cells but no valid path was found. When the visited cells were drawn onto the grid, it was apparent that something was very wrong with the algorithm. The algorithm appeared lost and aimless for lack of a more technical word. It would wander about the grid, turning in random directions, sometimes completely away from the goal node. This is obviously a huge issue and could not go un-fixed as it would mean that a very crucial algorithm would not be a part of the software solution.	The issue was located during the development of the module, not requiring formal testing to be found. The module was compared to pseudocode of the algorithm and no discernible difference was found. The source of the bug is still unknown. After roughly a week of programming and approximately 4 attempts at re-writing the A* algorithm based on different pseudocode, a working version was created. This issue remains the most difficult bug created by the software, it is a great success that the issue was fixed.
dijstras	In the early stages of development, the only pathfinding algorithm that had been implemented was dijkstra's algorithm. When a user tried to find a path using this module, the module would return an incorrect visited cells array. The visited cells array would always include the cell in the top left corner of the grid and the visualisation appeared incorrect at times. The implication to the user is not huge as the visualisation was only slightly wrong, however the fact that the top left cell was always visited was noticed during the project proposal and because of this, had to be fixed.	The bug existed in the code base for several weeks of development as it was not deemed to have a high impact on potential users. The bug turned out to be the result of an un-necessary condition being placed on the cells that were accepted into the visited cells array which always evaluates to be true on the top left cell of the grid. The bug was located and fixed after a small amount of trial and error.

Module Testing

Helper functions will not be specifically tested and will instead be determined to be correct if the parent function produces a correct output. Helper functions are listed in italics under the parent function.

Modules were tested with the test data as follows. The *actual output* column was recorded as a percentage of tests that resulted in a correct output out of the total number of tests except when a pathfinding module was tested. Module inputs are based on the above table. **The modules were tested multiple times throughout development however, the values shown currently are reflective of the final version of the software.**

Module Name	**Expected Output**	**Actual Output (% of tests resulting in a correct output)**
clear_grid *disable_btns, get_default_grid, set_grid, remove_mouse_listeners, set_mouse_listeners*	Grid completely reset to default state no matter its current contents.	100%
restart_grid	All visited cells and shortest path cells are removed. Node positions and wall positions are maintained.	100%
import_grid *csv_to_arr, set_grid, remove_mouse_listeners, set_mouse_listeners*	File dialog appears, prompts for a file. If the file is valid then the grid is replaced by the contents of the csv.	100%
save_grid *grid_to_array*	Software prompts for a file name. If a valid file name is entered (not empty) then the file is saved to the downloads folder as a .csv file with the given name.	100%

get_blocked_cells(grid)	Returns a one dimensional array of integers where each integer represents the id of a blocked cell.	100%
get_node_cells(grid)	Returns a one dimensional array with a length of two where the first element is the id of the start node and the second element is the id of the end node.	100%
get_neighbors(blocked_cells) *coords_of, get_neighbors_to_cell*	Returns a two dimensional array where each index represents the id of a non-blocked cell and each corresponding element is a one dimensional array of possible moves from the given cell id (doesn't include walls).	100%
dijstras(neighbors, node_cells)	Return an array of visited cells and an array representing the shortest path that matches the result produced by other simulations such as https://qiao.github.io/PathFinding.js/visual/ when using an identical grid layout.	Shortest path and visited cells were identical to the simulation in all cases.
a_star(neighbors, node_cells)	Same expectation as **dijstras**	Shortest path was identical but the visualisation differed very slightly in some situations. This will require future testing but is not an immediate concern because it has a negligible impact from a users perspective.
best_first_search(neighbors, node_cells)	Same expectation as **dijstras**	Shortest path and visited cells were identical to the simulation in all cases.
breadth_first_search(neighbors, node_cells)	Same expectation as **dijstras**	Shortest path and visited cells were identical to the simulation in all cases.

Testing of whole software solution

The Pathfinder software was tested as a whole throughout the development process as well as post development. The entire software solution was tested using a wide variety of grids as mentioned in the *module test data* section, the result of the pathfinding was compared to other simulations and the results were all but identical. The conclusion of this software testing was completely successful, no issues were found that had any impact on user experience and users ability to learn about pathfinding algorithms.

Evaluation

Production Build vs. Initial Requirements

In a comparison between the current production build and the initial requirements document, it is apparent that the production build has exceeded the requirements in the amount of functionality and the quality of the interface.

	Production Build vs. Initial Requirements
Interface	The user interface is far superior in the production build. The top navigation bar takes up less vertical space while maintaining functionality and usability. There is more space for the grid, meaning that the user has more room to use the software however they wish. The production build of the software has a consistent color scheme and is more responsive to varying screen sizes. The software is more vibrant while also maintaining a high contrast, easy to navigate and read interface.
Primary Functionality	The production build maintained all of the planned primary functionality and even gained more features such as the live pathfinding feature, the information tab and the help button.
Secondary Functionality	Similarly to the primary functionality, the planned secondary functionality has been developed to support all primary functionality.

Feature Evaluation

Refer to the quality assurance criteria in the Term 1 Requirements and Planning Report for the criteria / scores.

	Module Names				
Criteria	clear_grid	reset_grid	import_grid	save_grid	Pathfinding algorithms
Integrity	3/3	3/3	3/3	3/3	2/3 (The A* algorithm produces a visualisation that is not always fully correct, it can be slightly of in some circumstances and therefore cannot receive a 3/3 in this category)
Performance	3/3	3/3	3/3	3/3	3/3 (These could still be optimised further if more abstract data structures were used. Given time constraints it was not considered optimal to include these slight improvements in the production build)
Usability	3/3	3/3	3/3	3/3	3/3
Maintainability	3/3	3/3	3/3	3/3	2/3 (Native javascript is not industry standard for web development therefore it cannot be expected that the software is easily maintained)
Testability	3/3	3/3	3/3	3/3	3/3

Overall Evaluation

The following is an evaluation of the final build of the software compared against the quality criteria set in the requirements and planning report.

Criteria	**Score**	**Notes**
Integrity	3/3	All primary software modules operate at a 3/3 level of integrity therefore the overall software will also operate at this level.
Performance	3/3	The software runs well across a variety of modern browsers. The software will not run exceptionally well on mobile devices due to screen size limitations but is supported on all desktop devices.
Reliability	3/3	Due to the nature of javascript giving a warning where most other languages would cause a runtime error, the software recovers from error quickly and will continue to function optimally irrespective of time.
Usability	3/3	The software has a simple and direct user interface which conforms to modern standards for a high quality user interface.
Maintainability	2/3	Maintainability could have been improved if a javascript framework like React was used.
Testability	3/3	Automatic testing was limited due to the software being web based, generating inputs to various functions would have been too tedious. Regardless, significant manual testing did take place.
Meeting Intended Purpose	3/3	The software certainly meets its intended purpose of teaching users about pathfinding algorithms. Through the rich visualisation as well as the information tab, a potential user is able to gain a deep insight into how these algorithms work without the need to understand code.

Beta Test Feedback

Beta test feedback was received as a way to improve the production build of the software.

Tester Name	Interface Ergonomics	Intuitiveness of Interface	Help Interface	Response Time	Usability	Potential to meet the goal	Any other comments
Luke	10	10	10	Perfect	10	Yes	Communicate more information surrounding the algorithms *during* pathfinding. Add a feature where you can press space and the algorithm will do one iteration etc.
Nathan	10	9	10	Very Acceptable	10	Yep	n/a
Maurice	10	9	10	Perfect, no complaint s	8	Yes, the information tab adds to the learning experience	The visualisation is very nice.
James	8	8	9	Certainly	10	Visualisation is great, so yes it does.	Explain what the various colors on the grid mean in the help menu
Jake	10	10	10	High quality	9	Yes	n/a
Mr Marsh	10	7 - Once you know what to click, the software is easy to use.	8 - the help is helpful but the colour	11. It's too quick.	8 - the software is easy to use.. But how does	5 - it teaches the idea of pathfinding, and identifies some algorithms where the user can interactively	Nooice. But have information about the algorithms in terms of how they work. Maybe a split screen where it

		But if it’s purpose is to educate on the algorithm should it have narration or annotations to describe what the algorithm is doing?	and position make it inconsistent with other interface elements and therefore easily overlooked.		the algorithm work?	explore the algorithms execution… I don’t think it teaches about the algorithm.	visually steps through the algorithm in a simple pseudocode.

All feedback was considered in the production of the final build. Certain comments such as Lukes “Add a feature where you can press space and the algorithm will do one iteration etc.” was not carried over to the production build due to time limitations but could be re-considered in the future. James’ suggestion to “Explain what the various colors on the grid mean in the help menu” was taken and implemented in the final build. Mr Marshes suggestions of “narration or annotations to describe what the algorithm is doing” and “split screen where it visually steps through the algorithm in a simple pseudocode” were not implemented due to time constraints but were excellent suggestions for future iterations of the software. Mr Marsh's suggestions of adjusting the position and color of the help button to make it consistent with other interface elements and slowing down the animation were taken and implemented in the final software build.