PROJECT DOCUMENTATION

PART 1

PROBLEM BACKGROUND

This project is an attempt to understand path finding algorithm through visualization and determine how path finding can be used effectively in Game playing and Robotics. This visualization can be used for education purposes and is useful in determining how the search will be affected if any of the parameters in search is altered. In this project a player moves through a user space and needs to determine which is the best way to get to an object depending on the terrain. I have used A* search for the same. Alternatives for this can be BFS, DFS, DLS, etc.

WHAT IS A*

A* is an informed search algorithm, or a best-first search, meaning that it solves problems by searching among all possible paths to the solution (goal) for the one that incurs the smallest cost (least distance travelled, shortest time, etc.), and among these paths it first considers the ones that *appear* to lead most quickly to the solution. It is formulated in terms of weighted graphs: starting from a specific node of a graph, it constructs a tree of paths starting from that node, expanding paths one step at a time, until one of its paths ends at the predetermined goal node. At each iteration of its main loop, A* needs to determine which of its partial paths to expand into one or more longer paths. It does so based on an estimate of the cost (total weight) still to go to the goal node. Specifically, A* selects the path that minimizes

$$f(n) = g(n) + h(n)$$

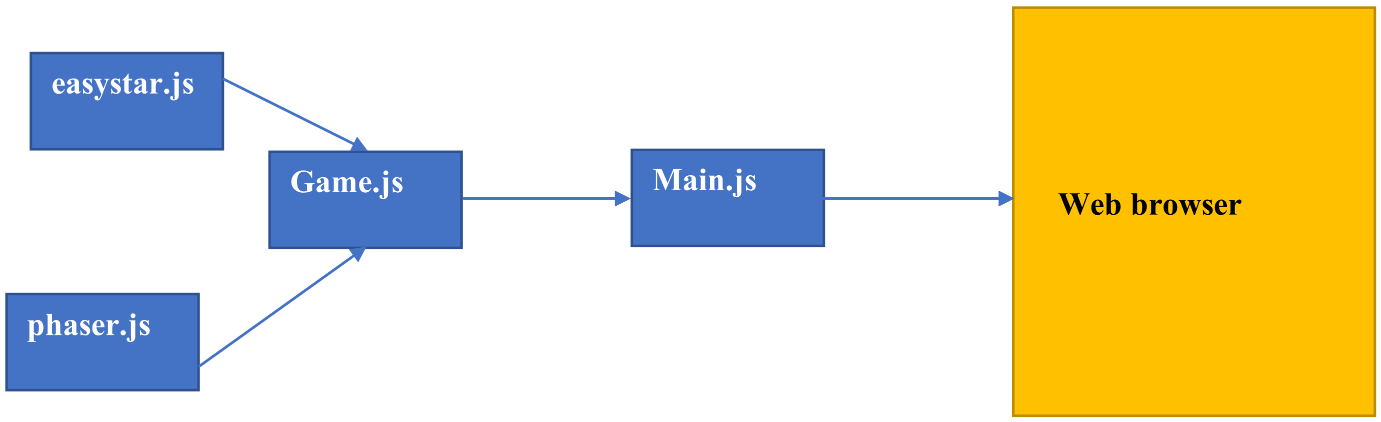
where n is the last node on the path, $g(n)$ is the cost of the path from the start node to n, and $h(n)$ is a heuristic that estimates the cost of the cheapest path from n to the goal. The heuristic is problem-specific. For the algorithm to find the actual shortest path, the heuristic function must be admissible, meaning that it never overestimates the actual cost to get to the nearest goal node.

WHY A*

A* is admissible and considers fewer nodes than any other admissible search algorithm with the same heuristic. This is because A* uses an "optimistic" estimate of the cost of a path through every node that it considers—optimistic in that the true cost of a path through that node to the goal will be at least as great as the estimate. But, critically, as far as A* "knows", that optimistic estimate might be achievable.

PART 2

SOFTWARE ARCHITECTURE



The main.js file is rendered onto the screen. The main file uses the game object as the scene. The game object internally uses easystar.js and phaser,js.

DESIGN CHOICES AND USE CASES

To create the interface a Javascript utility called Tiled was used. I have assigned different properties to each tile such as collision and cost.
For the search, Easystar library was used as a guideline.
For the Pokemon sprites and open source pokeapi was used to get random positions →
https://pokeapi.co/

There is only one usecase. The user clicks on a Pokemon sprite on his screen and is redirected to that position.

The tech stack involves Phaser, npm, nodejs, http-server, tileset, karma, jasmine

TESTING METHODOLOGY

Karma and jasmine were used for unit testing.

ALGORITHM VALIDATION PROCESS

The algorithm can be validated by looking at the GUI on the browser. Each step is taken such that the cost is minimized and the goal is successfully reached. The algorithm was tested for different positions of sprites and players and different obstacles.