Pathfinder – The Technical

# Aim

The goal of the path finding feature is for the user to be able to view all available buildings or rooms (depending on whether the user is on the interior maps or exterior map). The user should also be able to plot a path between any two positions and view the route. The route should be the most optimal and efficient route available.

## Map design

The design of the maps is essential for the Path Finding feature to operate successfully. Each map must be an identical twin of the real environment to be able to give the most accurate and efficient results. Each map has separate layers to accommodate all the buildings, objects, paths and other extras that make up the map. The path finder has its own layer that holds paths and entrances/exits to rooms or building in the shape of rectangles.

Each rectangle representing traversable paths has a unique ID assigned to it with the letter ‘P’ (representing the P in ‘path’) in front of it. The rectangles representing an entrance/exit are assigned a unique ID corresponding to the room or building. As each door should not be considered traversable, each rectangle representing a door must have the letter ‘D’ (representing the D in ‘door’) at the end of the ID.

## JSON data

As each rectangle representing the traversable paths and the doors are unaware of any object surrounding it. This must be manually created to ensure a route is able to be created and that the route is valid. This allows us to design rules to how a route is connected from each path to the next path and restricts the path finder to creating a route that passes through a non-traversable object. Each interior and the exterior map has a JSON file containing all the traversable paths and the doors of the entrances/exits represented as JSON objects.

The use of a JSON file allows us to easily incorporate the contents and their functionality into our code. Each JSON object contains five attribute : value pairs representing the ID of the current path or door and the ID’s of the surrounding paths or doors of the current one (i.e. north, south, east, west). If a path or door does not have anything in a specific direction then the value is “null”.

{

"current": "GULBD",

       "north": "null",

       "south": "null",

       "east": "P40",

       "west": "null"

},

{

"current": "P20",

       "north": "null",

       "south": "LIB-RED",

       "east": "P21",

       "west": "P78"

},

# Path Finding Algorithm

When a user searches for a route between two positions, selecting a building or room as their starting point and doing the same for their destination (depending on if they are searching from the main map or an interior map). The code creates an AJAX call retrieving the correct JSON file corresponding to the current map the user is currently viewing and stores each JSON object as JavaScript class objects and stores them inside a global JavaScript array. After this essential operation is completed the value of each option selected as the starting and destination positions is passed to a method (**findPathToGoal**(start, goal)) inside a class named PathFinder.

The PathFinder class contains three arrays named queue, visitedNodes and paths. The queue array holds all the paths that are yet to be traversed which are the paths surrounding the current node. The visitedNodes array contains all the paths that the algorithm has already ‘visited’. The paths array holds all the current routes found by the algorithm as there are many routes from one point to another that a user can take.

## The main operation

The algorithm uses the queue array to store all the available directions for the current path by calling the **addToQueue**() method. This method adds all available paths for the current path object to the beginning of the queue array. Whilst the paths are being added, the previous value of each path object is set to the current path (only if it hasn’t been set). Once all available paths have been added to the queue, the current path is added to the visited array and the current node is changed to the first path inside the queue. This operation makes the algorithm work in a Breadth First Search (BFS) way. The current node is removed from the queue to stop the algorithm processing the path again.

while (currentNode.containsNode(goal) == false) {

currentNode = nodeObjects[this.getPath(this.queue[0])];

      this.visitedNodes.push(currentNode.getCurrent());

      this.queue.shift();

// Stop looping if currentNode contains goal after updating from queue

      if (currentNode.containsNode(goal) == true) {

       this.createPath(start, currentNode, goal);

             break;

      }

      this.addToQueue(currentNode);

}

The algorithm continues this process of going through each path represented as JSON objects until the current path contains the ‘goal’ building/room. Once the current path has found the ‘goal’ room, it calls a method named **createPath**() which iterates from the current path back to the starting path. The route is generated using the path object method **getPrevious**(), where the path object corresponding to **getPrevious**() is added to an array which represents the route. To ensure this is the most efficient route, the algorithm continues this operation until the queue is empty.

while(currentNode.getCurrent() != start) {

var index = this.visitedNodes.indexOf(currentNode.getPrevious());

      currentNode = nodeObjects[this.getPath(this.visitedNodes[index])];

      path.splice(0, 0, currentNode.getCurrent());

}

# Result

The algorithm returns the most efficient and shortest route by comparing the length of the routes found. The shortest route is returned every time with the interior maps, unfortunately the algorithm only returns the shortest route 80% of the time with the main exterior map. If there was more time, I would like to develop and improve the algorithm to generate the most efficient route 100% of the time. As an additional extra feature, when the user selects their starting and destination point of their route. The corresponding door is clearly coloured in a red colour to help the user locate where they are or where each door is.



