Struct Node

The struct node is formulated to hold all the data for each computational node of the computational graph that motion planning algorithm works on. The x & y coordinates, list of pointers to the neighbours, pointer to the parent node, heuristic and local costs, as well as bool variables to indicate if the node has been visited and whether the node is an obstacle or not.

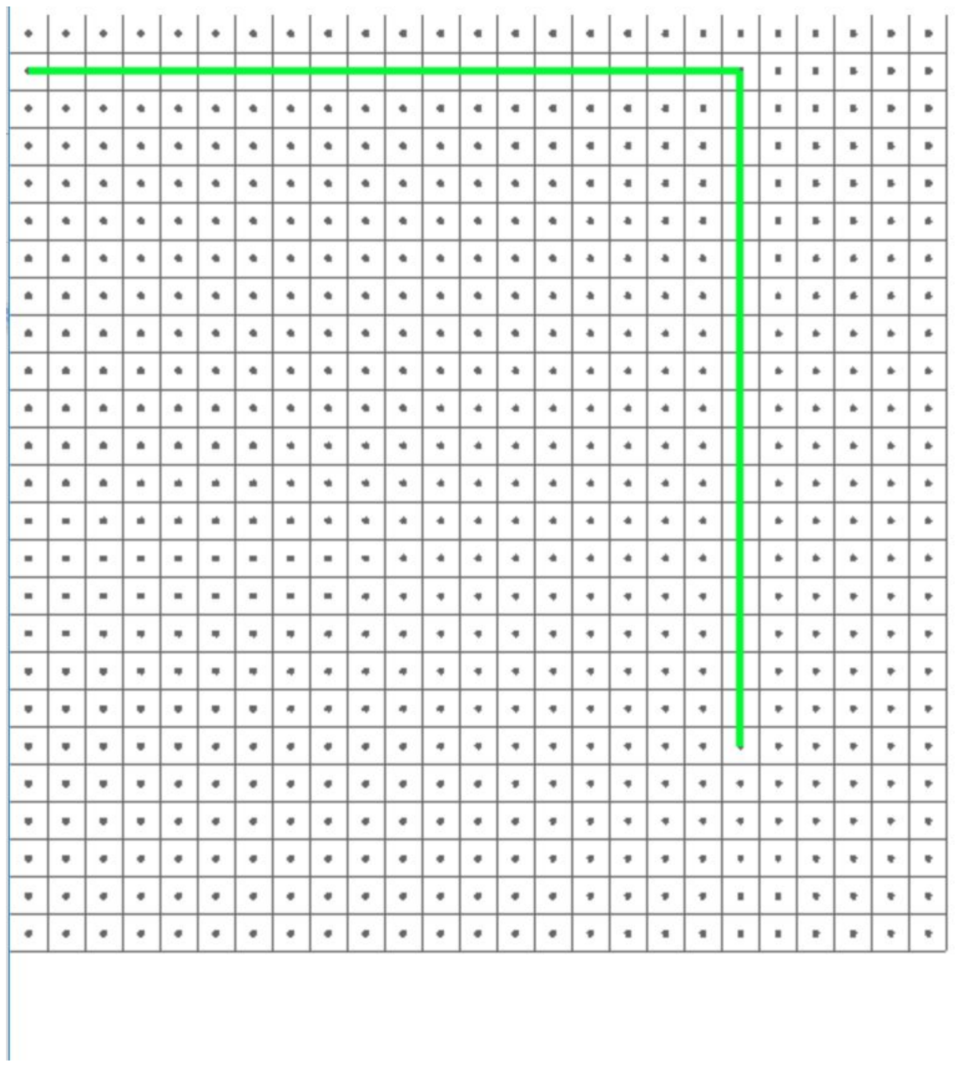
```
struct node
{       //Data structure for each node of the computational graph
   double x, y; // coordinates of node
   // node* neighbors[4]; // list of pointers for neighbours
   node* parent; // pointer to the parent node
   float globalGoal;
   float localGoal;
   bool isObstacle; // whether the node is an obstacle or not
   bool isVisited; // to store if the node is visited or not
   int distance; // to store distance from the start
   vector<node*> vecNeighbors; // list of connections to neighbors
};
```

Graphics Class

The class contains the member functions and variable required for visualization of all the 4 motion planning algorithms. In the next week, function to draw the robot, animate the robot and display the gui buttons as well as instructions would be added. Currently the class contains only 2 member functions, list of x and y coordinates for the computed optimal path.

```
vector<double> pathCoordsX;       //list of x coordinates for the optimal path
vector<double> pathCoordsY;       //list of y coordinates for the optimal path
```

Vector data types is more suitable for these member variables to allow dynamic allocation, since the length of optimal path is highly dynamic.
There are 3 function to draw the background grid-lines and nodes.
The constructor initializes an array of pointers to all the nodes of the grid and initializes the member functions of each node to required values. It contains another block to store the pointers of neighbours of each node is neighbour-list member function. The interfacing between the x and y coordinates of the grid nodes and the 1 dimensional array of pointers to each node warranted a conversion from the coordinates to the index. The **coords_Convert(int cX, int cY, int gridSize, int nX)** function takes the x and y coordinates of each node and returns the index of the respective coordinates required to access it's pointer from the pointer array **nodes**.
The **setPath(double x, double y)** function allows easy interface with graphics class for all the other classes containing the motion planning algorithms and main function.
The draw_Path function draws the optimal path (if any) stored in the path coordinates member variables. Please refer to the image attached below for better understanding.



Dijkstra’s motion planning algorithm:

The Dijkstra's algorithm is just a minor variation of the A* algorithm, the only difference being that the heuristic in the A* motion planning algorithm is set to 0. Since, another teammate is already working on the Dijkstra’s algorithm for the project, I collaborated with him for some functions in that and formulated an additional function that calls the A* function, sets the heuristic to zero and computes the optimal path.

Debugging and testing

The main function creates an instance of the graphics class, calls the respective functions and draws the openGl window to show the visualization. The graphics class and main function that I have formulated, also aid in debugging the component code of other team members.

Please refer to the attached .cpp file for the component code: Comp_Graphs.cpp