159.740 **Intelligent Systems**
Assignment

Robot Path-Planning Algorithms for Fully Dynamic Shortest Path Problems

(Optimal + Incremental + Heuristic Search)

D*Lite and (LPA* or other)

You are allowed to work in a group of 2 members for this assignment.
Deadline for submission: (tentative) 6th of September, 2021

Instructions:
Your task is to write a C++ program that simulates two path-planning algorithms in an 8-connected grid world.
A graphics start-up program is provided to help you plot the optimal path generated and view the details of the variables involved in the computations. This is available in the stream site.

Requirements:

1. You must use the graphics library provided to display the path generated, the values of the variables involved in the computation, and the vertices (states) expanded during the search.

2. For any **special data structures (**e.g. Heap data structure, etc.) that you might want to use in your algorithm implementations, you may utilise/consult codes available in books and websites provided that you cite, document and incorporate them properly with the start-up codes provided.

3. Implement the D* Lite algorithm (Final Version) and (Lifelong Planning A* or other)

4. The search must consider 8 possible directions of movements in the following order: (1) North-West, (2) North, (3) North-East, (4) West, (5) East, (6) South-West, (7) South, and (8) South-East. Generate the predecessor and successor nodes using the same sequence.

1	2	3
4	robot	5
6	7	8

5. Use the **Heap data structure** for implementing the **Priority Queue**. You may use the Standard Template Library (STL) to satisfy this requiement.

6. Your program must be able to execute the algorithms using different parameter settings to record and analyse their performance:

   6.1. Heuristic functions:
   - **Euclidean distance**
   - **Chebyshev distance (Manhattan distance for 8-connected gridworlds)**: Maximum of the absolute value of the differences between the x and y-coordinates of cells

   6.2. Initial gridworld - must be loadable from a text file (this is already implemented in the start-up codes).

   6.3. Changes in the gridworld – specify which cells are unknown to be blocked/unblocked (a facility for blocking/unblocking cells is already provided in the start-up codes).

   6.4. Display the calculated path graphically.

6.5. Test the algorithms on the following test grid worlds provided, namely:

1. Grid_Dstar_journal.map
2. Grid_Lpa_journal.map
3. Grid_Lpa_journal_big.map
4. Grid_big.map
5. Grid_trap.map

6.6. Test the algorithms further by creating 3 big (100x100) user-defined gridworlds. For replanning tasks, specify unknown cells (unknown to be blocked: 9), (unknown to be traversable: 8). Submit all the gridworlds (text files you have created), as part of the assignment. The start-up codes already accept gridworlds of different sizes.

6.7. Performance measures for both the initial and changed gridworlds:

- Maximum queue length
- path length
- number of state expansions
- vertex accesses (to read or change their values)
- actual running time (sec.)
- Report any observed flaws in your implementation, as identified from the experiment runs (specify if there is any test gridworld that it cannot solve)

7. It is up to you to write the main program, and any classes or data structures that you may require.

8. Use the given table structure for tabulating all experiment results. .

9. LPA* Search Behaviour:

INITIAL SEARCH:

- Consider Type ‘9’ cells as UNKNOWN TO BE BLOCKED
  - Apply the FreeSpace Assumption during initial search
- Consider Type ‘8’ cells as UNKNOWN TO BE Traversable
  - Cells of this type should be avoided

REPLANNING: Assume that the robot is able to discover the real status of the unknown cells in the entire gridworld.

- During replanning, automatically change Type ‘9’ cells into Blocked cells
- During replanning, automatically change Type ‘8’ cells into Traversable cells

10. **D*Lite Search Behaviour:**

INITIAL PLANNING:

- Consider Type '9' cells as UNKNOWN TO BE BLOCKED
  - Apply the FreeSpace Assumption during initial search
- Consider Type '8' cells as UNKNOWN TO BE Traversable
  - Cells of this type should be avoided
- Sample sequence of steps:
  - copyDisplayMapToDstarMaze(grid_world, dStar);
  - copyDisplayMapToAStarMaze(grid_world, aStar_maze);
  - dStar->Search();
  - copyDstarMazeToDisplayMap(grid_world, dStar);
  - grid_world.displayPath();

REPLANNING:

- The robot must be able to discover the real status of the unknown cells within its sensory range (radially detect up to a range of one cell away from the robot).
  - Upon detection, automatically change Type '9' cells into Blocked cells
  - Upon detection, automatically change Type '8' cells into Traversable cells

- Sample sequence of steps:
  - dStar->Replan();
  - copyDstarMazeToDisplayMap(grid_world, dStar);
  - grid_world.displayPath();

- **D*Lite** must be able to perform replanning, as the **robot moves towards the target**, and while the world is changing. Assume that the robot's sensors can radially detect up to a range of one cell away from the robot. The simulation must update the Start vertex each time the robot moves, as an indication of the robot's current position. The algorithm is expected to run until the robot eventually reaches the goal.

11. Keyboard handling function: **GetKeyAsyncKeyState()** may record a single key stroke as multiple keystrokes. In order to get around this problem, introduce a delay(200); after executing an action corresponding to a command.

Example:

```
case 109:  //corresponds to F9 key event
    d_star->computeShortestPath();
    endTime = clock();

    cout << "running time : " <<(double)(endTime - startTime) / CLOCKS_PER_SEC *1000 << "ms\n";

    action = -1;
    Sleep(200); //introduce delay here
    break;
```

The start-up codes are using the following keys:

- Ctrl + Space: Initial planning
- Ctrl + Enter: Replanning (for LPA*)

- Ctrl + Enter: Move robot towards the goal and Re-plan (for D*Lite) - perform replanning automatically (if necessary) as the robot moves towards the goal, once the agent encounters unknown obstacles on its way to the goal.

Running the different algorithms:

- You may add another parameter to the command line arguments to specify the algorithm to use (e.g. main.exe algorithm gridworld heuristic). Alternatively, you may submit two separate programs (one program per algorithm).

EXPERIMENTS:

Experiment #1 (Initial and Second Search only, similar to the hand-simulation)

Algorithm: **LPA* or other**
Heuristic: Manhattan-8 distance

	Gridworld	Max. Queue length	Path length	No. of state expansions	Vertex accesses	Actual Running Time (msec.)
	Grid_Lpa_journal.map	Initial search, second search	Initial search, second search	Initial search, second search	Initial search, second search	Initial search, second search

Experiment #2 (Initial and Second Search only, similar to the hand-simulation)

Algorithm: **D*Lite**
Heuristic: Manhattan-8 distance

Note that the table requires details only of the initial and second search results, but the D*Lite algorithm is expected to run until the robot reaches the goal.

	Gridworld	Max. Queue length	Path length	No. of state expansions	Vertex accesses	Actual Running Time (msec.)
	Grid_Dstar_journal.map	Initial search, second search	Initial search, second search	Initial search, second search	Initial search, second search	Initial search, second search

159.740 **Intelligent Systems**
Assignment

Experiment #3 (D*Lite, Search to completion)
Run the algorithm until the agent arrives at the goal position. Simulate the algorithm on different grid worlds, using different heuristic functions, then compare the results.

Algorithm: D*Lite
Heuristic: Chebyshev distance

	Grid world	Max. Q length	Path length	No. of state expansions	Vertex accesses	Actual Running Time (msec.)
1	Grid_Dstar_journal.map					
2	Grid_Lpa_journal.map					
3	Grid_Lpa_journal_big.map					
4	Grid_big.map					
5	Grid_trap.map					
6	User-defined					
7	User-defined					
8	User-defined					

Algorithm: D*Lite
Heuristic: Euclidean distance

	Grid world	Max. Q length	Path length	No. of state expansions	Vertex accesses	Actual Running Time (msec.)
1	Grid_Dstar_journal.map					
2	Grid_Lpa_journal.map					
3	Grid_Lpa_journal_big.map					
4	Grid_big.map					
5	Grid_trap.map					
6	User-defined					
7	User-defined					
8	User-defined					

Experiment #4 (LPA* or other, Search to completion)
Run the algorithm until all unknown cells are discovered
Algorithm: LPA* or another algorithm
Heuristic: Chebyshev distance

	Grid world	Max. Q length	Path length	No. of state expansions	Vertex accesses	Actual Running Time (msec.)
1	Grid_Dstar_journal.map					
2	Grid_Lpa_journal_big.map					
3	Grid_big.map					
4	Grid_trap.map					

5	User-defined					
6	User-defined					
7	User-defined					

Criteria for marking

- Experiments and Documentation – 30%
  - Type-written report in MS-Word format.
  - Include a Table of Contents
  - Specify the data structures used in the different algorithms.
  - Provide a skeleton C++ code for each algorithm (more detailed than the pseudocode of the algorithms provided in class).
  - Using the sample gridworld defined in the D*Lite and LPA* journals (and also used in the tutorials) Include a snapshot of each algorithm run, indicating graphically the path generated and the vertices expanded.
  - Tabulate the results of all experiments performed according to the format given.
  - Discuss briefly the results of all the experiments.
  - User's Guide (simple guide on how to operate the simulation system, short cut keys)
- System implementation – 70%
  - Must compile using g++ 9.2.0
  - Submit the source codes and text files containing the grid worlds tested.

Nothing follows.