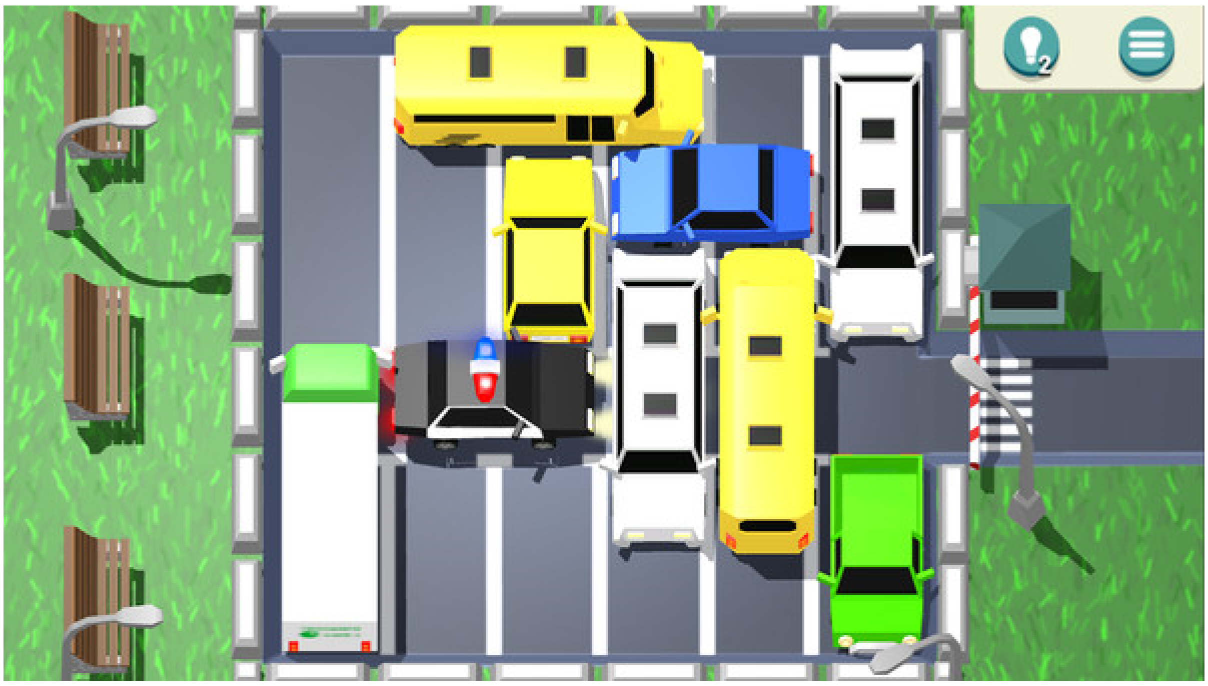
Group 8: Unblock me cars puzzle

Student Name	**Student Id**	**Student Email**
Hồ Minh Khôi	20204917	khoi.hm204917@sis.hust.edu.vn
Hoàng Anh Chung	20204901	chung.ha204901@sis.hust.edu.vn
Lê Trần Thắng	20204926	thang.lt204926@sis.hust.edu.vn
Bùi Mạnh Cường	20204871	cuong.bm204871@sis.hust.edu.vn
Mai Hà Đạt	20200135	dat.mh200135@sis.hust.edu.vn

1, Presentation of the subject

Our task is to write a program to find the way to arrange cars so that it can make way for the emergency car. The puzzle is a 6x6 matrix, the goal is fixed and the initial state of the emergency car must be on the same line with the goal. The initial states for the remaining cars can both be implemented manually and randomly generated. We mainly focus on solvable instances to evaluate our solutions.



The program will have several outputs:

- Step - by - step solution on how to solve the map instance
- Number of steps required to solve
- Time and Space Complexity

2, Description of the problem

2.1, Formulation of the problem for search problem

- *Initial state:* The original position of all cars (either implemented manually or randomly generated)

- *Actions:*

The game consists of multiple cars, but generally there are two types of them

+ Horizontal cars: move left and right

+ Vertical cars: move up and down

We define our transition model to be

S(CurrentState) = (CurrentState, move, NextState)

Here, move is defined as an element of the list of all available moves of CurrentState.

- *Goal test:* The emergency car can exit the matrix

- *Path cost:* There are mainly two ways of measuring path cost: By number of actions, or by number of step that each car moved. However, we choose to measure the number of actions, because in reality, a police officer for example should ask a driver to move 2 steps forward, instead of asking him to move 1 step, and then go back to tell him to move 1 step forward again.

2.2, In reality, there may be a traffic-jam-driving agent

- PEAS formulation:
    + Performance measure: fast, optimizing the number of moves.
    + Environment: Cars around the emergency car, other obstacles.
    + Actuators: Screen monitor of the police, or/and automatic systems that helps the cars move
    + Sensors: camera, sensor to transfer actual image into software.
- Type of environment:
    + Fully observable: the agent's sensor gives it access to the location of all other cars.
    + Static: The environment is unchanged during the time where the agent is making its next decision (all cars stuck)
    + Deterministic: The next state of the environment is completely determined by the current state and the action executed by the agent.
    + Discrete: each car has only 2 ways to move: up and down for a vertical car, left and right for a horizontal car.
- Type of agent: goal-based agent

3, Explanation of the choice of algorithms and their parameterization

3.1, Uninformed Search

a, Breadth-first search: we choose it instead of Depth-First search because DFS can get stuck on an infinite path easily. In DFS, for example, a car moves 1 step forward, then one of its feasible moves is to move 1 step backward. Then if it chooses that move, it will be stuck in an infinite forward-backward loop.

Another property of our Breadth-first search is that we negate all repetitive states. It both saves time and also doesn’t affect the solution (that we will get the lowest number of moves)

b, Iterative deepening search: Both IDS and DLS (Depth limited search) solve the problem of DFS’s completion, but we choose IDS because DLS is fixed in search depth and that can lead to not finding results, while IDS gradually increases the search depth and that helps us find the results of the problem.

3.2, Informed Search

a, A search:* Both greedy best-first search and A* are methods to find approximately optimal solutions with better runtime. Even so, greedy best-first search only searches for the states with the smallest heuristic value among the states, so it can get stuck in loops. Meanwhile, A* calculates both the number of steps moved from the initial state and the heuristic value of a state, thereby avoiding loops.

4, Implementing the algorithms

4.1, Breadth-First Search

In this algorithm, we use the fringe as a queue. We will expand the first node of the queue and every new node will be put at the back of the queue. Because all of the states of the problem are fully observable, we can save all the visited nodes in memory.

4.2, Iterative Deepening Search

In this algorithm, we will iterate over the depth limit. For each depth limit, we will use the DFS to check whether we could reach the goal given that every path length could not exceed the limit.If the goal is reached, we will return the depth limit as the solution.

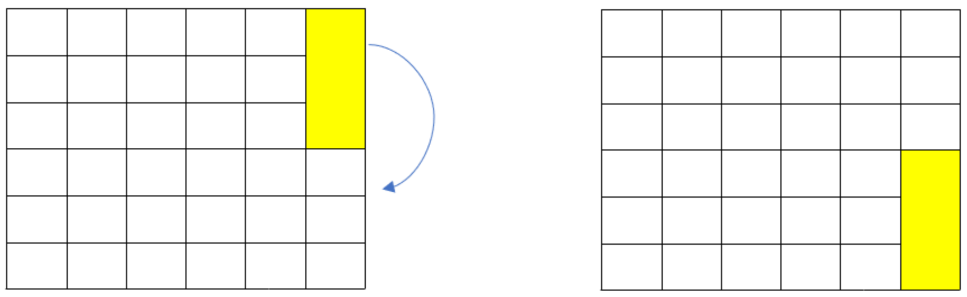
4.3, A* Search.

For the A* search, a priority queue is used as the fringe for implementing the algorithm, which will return the smallest priority value node when we use the dequeue method. Thus, we can expand the best heuristic node in the fringe.

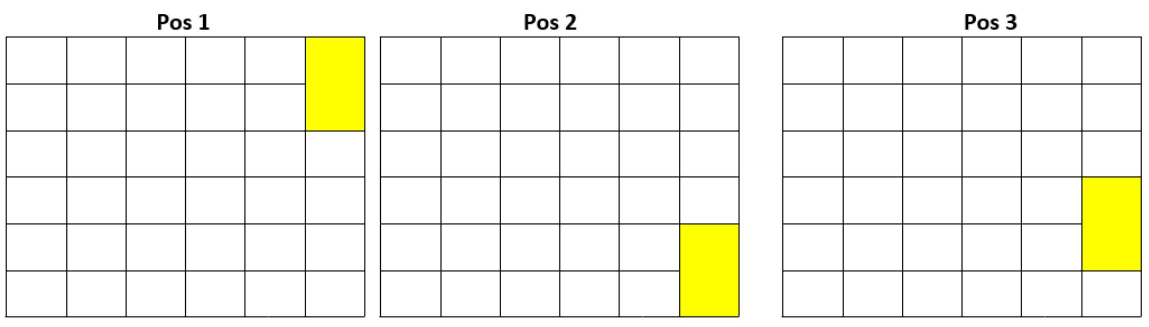
So far, we have come up with 3 heuristic functions:

+ *Heuristics 1:* This is probably a method which everyone solving the map has thought about. As we need to make a way for the emergency car to go to the exit, all of the vertical car blocks must not block the way to the exit, which is the row 3 in the matrix. Therefore, we will count all cars blocking the exit row as our heuristic.

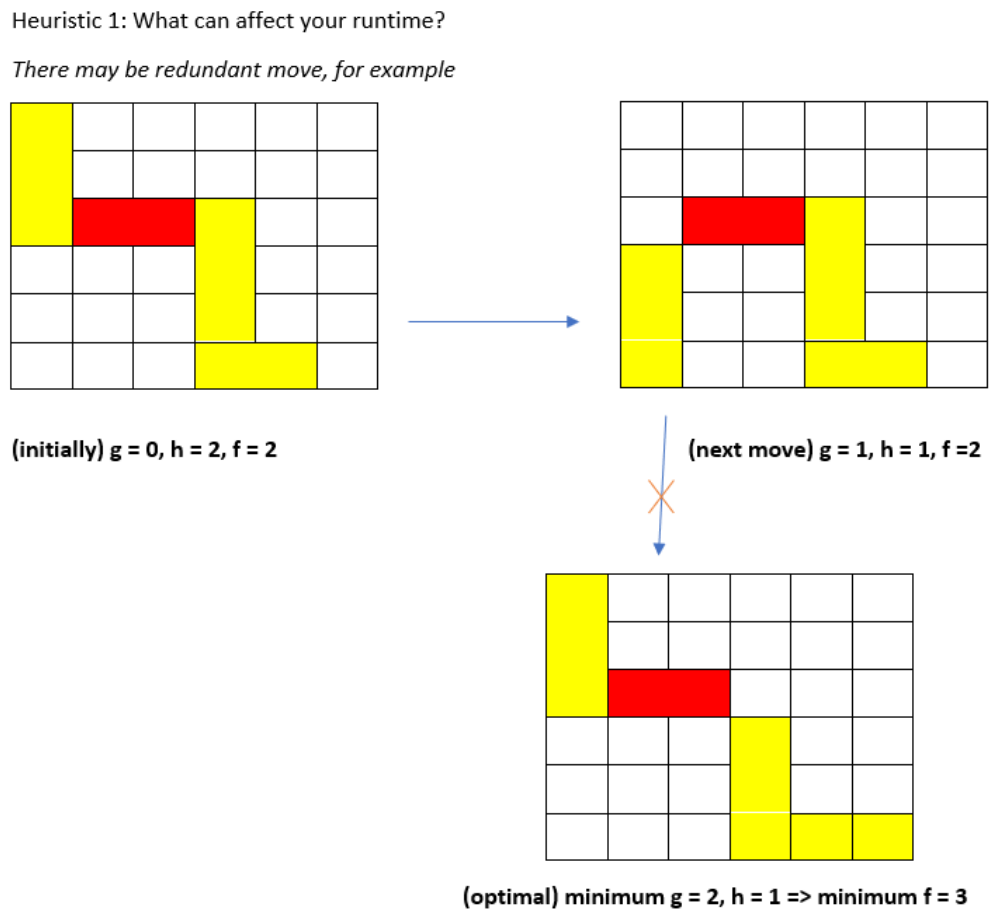
Heuristic 1.1: If a vertical block is of length 3, then its final position should be at the bottom



Heuristic 1.2: If a vertical block is of length 2, then there may be 3 final positions for them



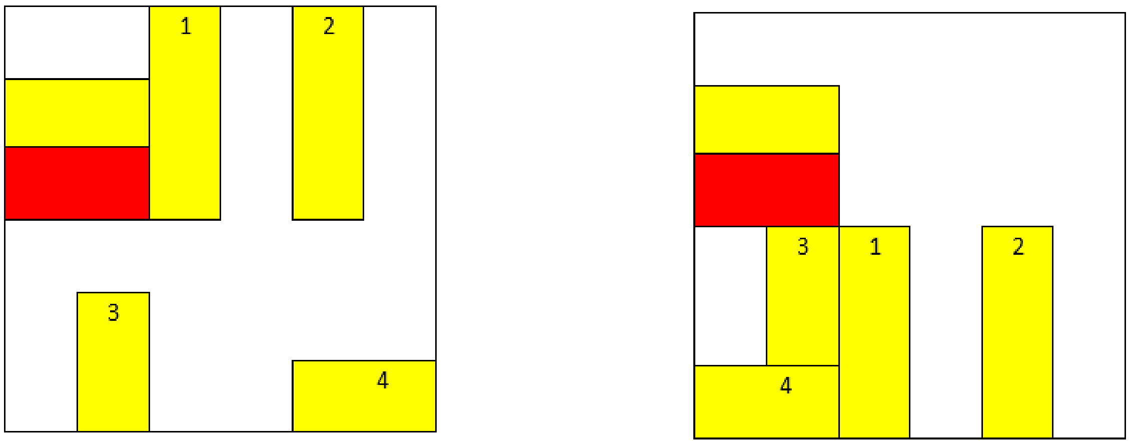
This heuristic can lead to some redundant moves as shown below, but we can still get the solution.


Heuristic 1: What can affect your runtime?
There may be redundant move, for example
(initially) g = 0, h = 2, f = 2
(next move) g = 1, h = 1, f =2
(optimal) minimum g = 2, h = 1 => minimum f = 3

+*Heuristics 2:* If we know the final state of the map, we can check if a car block's position is similar to its at the final state or not. The heuristic is calculated based on the misplaced block of a state . To get the final state of an initial map, we run BFS on it first to get and store the optimal final state.

Heuristic 2: Number of misplaced blocks, compared to goal state (generated by BFS)

Here f(state) = 4, as there are 4 blocks at the wrong position

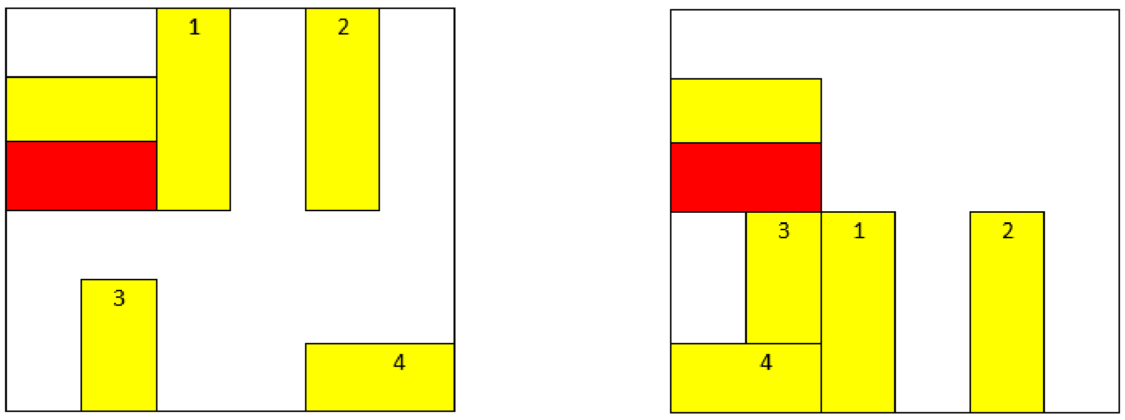


+*Heuristics 3:* Almost the same as heuristics 2, we still need to know the final state and check every car block to find if it is misplaced. But instead of counting the number of misplaced blocks, we count the Manhattan distance of them from the current state to the final state as a heuristic. Although this heuristic makes the evaluation function mathematically incorrect. (Because we define the cost of moving each car equals 1 but the heuristic varies depending on the position of the car map). However, we can easily fix the source code to change the cost of each move, and thus yield different results. However, from the result we've obtained (which will be shown later), the mathematically-incorrect cost of moving a car doesn't affect the solution that much, and sometimes instead provides many better solutions.

Heuristic 3: Manhattan distance between current state and goal state (generated by BFS)

Here Manhattan (1) = 3, Manhattan (2) = 3, Manhattan (3) = 1, Manhattan (4) = 4

Therefore f = 3+3+1+4 = 11

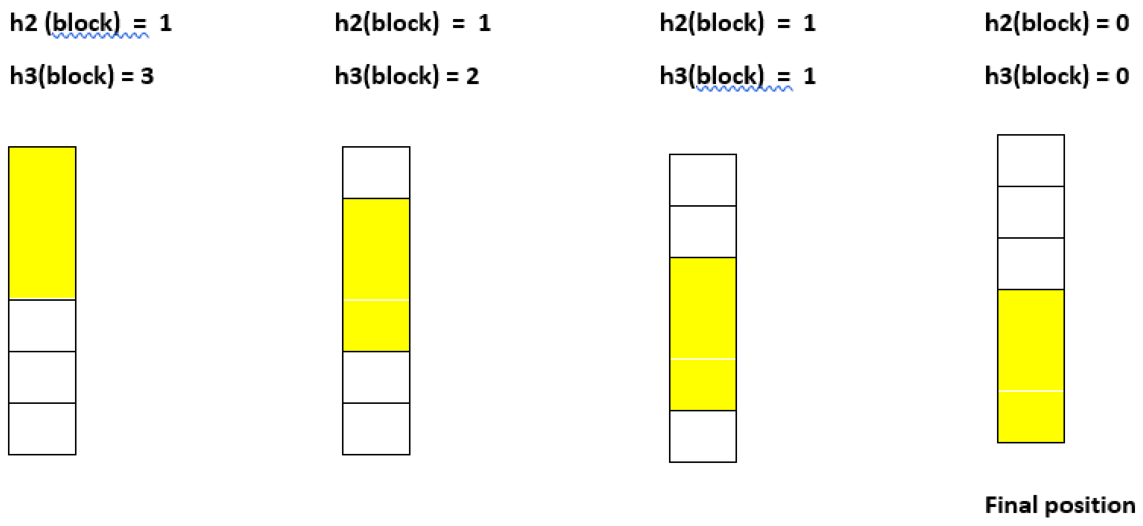


Comparison of heuristics:

- In fact, in some of our test cases the A* search returns nearly-optimal results because we suspended all repeated states. We think that it may take less time running and still return a good solution with only 1 or 2 steps more than BFS.
- However, if we accept repeated states, the A* search will reach an optimal solution.
- *The problem with heuristic 1* is mentioned above, so we can conclude that heuristic 1 is **not admissible**. However, we still can use it for most cases. In those cases heuristic 1 is still admissible. In conclusion, A* here is **not admissible.**
- Heuristic 2 is **admissible** because each car not in its final position must be moved at least once. Thus the true cost will always be greater than heuristic value at any state.
- Since for each move, the heuristic value of heuristic 1 and heuristic 2 can only be reduced by at most 1, we obtain the inequality:

h(CurrentState)<= c(CurrentState,move,NextState)+h(NextState)
Therefore both heuristics 1 and 2 are **consistent.**

- Heuristic 3 may not be mathematically correct, but it weights the unexpanded nodes differently from heuristic 2, thus reducing time and space complexity.



Comparison figure of heuristic 2 and heuristic 3

- On the same computer, heuristic 2 runs twice faster than heuristic 1 for the hardest map instance (input 44). Also, heuristic 3 runs faster than heuristic 2 for about 4 to 5 times.

5, Comparing the result of the algorithm

5.1, Providing quantitative performance indicators

We will run all the inputs with each algorithm to get the maximum nodes in the fringe and also running time to approximately compare the time complexity and space complexity among them. However, we must keep in mind that the results may heavily depend on the input. Not only the input size(number of car blocks), but also their positions.

Note: We will compare IDS(Not deleting repetitive nodes) only in terms of running time, since its memory is mainly about recursive stacks which is likely to be equal to the depth it is expanding.

First, we compare some particular inputs ranging from easy to difficult levels, which are input 11, input 30, input 44.

	Input 11	Input 30	Input 44
Maximum (in term of time)	*A* search with heuristic 1:* 0.06231 sec	*IDS(Not deleting repetitive nodes):* 1.66654 sec	*IDS(Deleting repetitive nodes):* 3246.44 sec
Minimum (in term of time)	*IDS(Not deleting repetitive nodes):* 0.00674 sec	*A* search with heuristic 3:* 0.02277 sec	*A* search with heuristic 3:* 11.5491 sec
Maximum (in term of space)	*A* search with heuristic 1:* 92 nodes	*BFS:* 1270 nodes	*BFS:* 40920 nodes
Minimum (in term of space)	*IDS(Deleting repetitive nodes):* 16 nodes	*A* search with heuristic 3:* 40 nodes	*A* search with heuristic 3:* 7102 nodes

Table 1: Extremum of time and space complexity for 3 different inputs

Then, we compare all algorithms with BFS by:

+ Run all the inputs for each algorithm and then store the running time and the maximum nodes.

+ For each input, we take the percentage of other algorithms to BFS in terms of running time and maximum nodes.

+ We take the average value of all the percentages for all algorithms and compare them.

	BFS	IDS(Not Deleting Repetitive nodes)	IDS(Deleting Repetitive nodes)	A* with heuristic 1	A* with heuristic 2	A* with heuristic 3
% time compared to BFS (easy maps from 11-30)	100%	108.7%	62%	71%	16.48%	16.62%
%space compared to BFS (easy maps from 11-30)	100%	only recursive stacks	36%	86%	29.1%	28.55%
% time compared to BFS (medium maps from 31-40)	100%	runtime is too high	1181%	94.78%	57.88%	40.78%
% space compared to BFS (medium maps from 31-40)	100%		70.8%	88.77%	60.61%	49.47%
% time compared to BFS (hard maps from 41-45)	100%		3558%	93%	46.9%	20.27%
%space compared to BFS (hard maps from 41-45)	100%		64.09%	92.6%	56.5%	27.5%

Table 2: Comparison of time and space complexity for 3 kinds of input states

5.2, Explaining the result

5.2.1, Time complexity

In most maps, the A* search with heuristic 3 is fastest, but in easier maps like map 11, the IDS (Not deleting repetitive nodes) is more effective (about 2 times less than A* with heuristic 3). The reason is in this case, IDS can save time by not comparing duplicate states and not calculating heuristic functions.

In harder maps, both IDS with not deleting repetitive nodes or deleting repetitive nodes are the slowest methods. IDS with deleting repetitive nodes is about 80 times more than A* with heuristic 3 and IDS with not deleting repetitive nodes is about 300 times more than A* with heuristic 3. It is because the branching factor of each node is so high that recursive stacks would be heavy.

In general, when we compare the average time of all methods with BFS, we can see that A* search runs quite faster than BFS and IDS runs very slower than BFS.

The reason is: IDS repeats all the work of the previous depth in each depth, so the time complexity is so big. In the BFS we use, we remove duplicate nodes so the time complexity is smaller. A* are the best methods because it searches based on the heuristic and leads to the optimal solution first.

5.2.2, Space complexity

Except for the small map like map 11, the BFS algorithm will mostly have the largest space complexity compared to other algorithms, because it trades off space to keep lower time complexity. The other methods, such as IDS in many cases don't need to expand as many nodes as BFS and with each new depth-limit, IDS removes memory from the old depth-limit.

A* search has smaller space complexity than BFS because these methods use heuristics and can reopen closed nodes.

6, Conclusion and possible extensions

6.1, Conclusion.

- Unblock me is not just a game, it also has practical applications. It can be used to train self-driving cars to reduce traffic jams. When the infrastructure has not been improved, this is also a solution for one of the biggest problems in the big cities.
- In general, breadth-first search always finds a optimal solution.
- However, as complexity of the map raises, A* search should be used to reduce computational time

6.2, Possible extensions.

If we have more time, we can make a GUI for this problem to describe the problem and the solution in a more understandable way.

We can add some fixed blocks 1x1 to make the problem closer to reality (In fact, on the road there are not only moving vehicles, but also stationary vehicles, points where road repair takes place, etc.)

To apply this problem into real life driving problems, we can use the openCV library to identify cars in the crossroad and map them into blocks for solving.

7, List of task

Task	Subtask	Contributors
Proposing the subject	Choosing the subject	Ho Minh Khoi
	Topic submission preparation	Le Tran Thang
	Heuristic proposal	Hoang Anh Chung (50%) Ho Minh Khoi (50%)
Programming	Map and Block library	Ho Minh Khoi (50%) Mai Ha Dat (50%)
	class State	Hoang Anh Chung
	Input handling	Mai Ha Dat
	Breadth first search	Hoang Anh Chung (80%) Ho Minh Khoi (20%)
	Iterative Deepening search - not closing nodes	Ho Minh Khoi
	Iterative Deepening search - closing nodes	Le Tran Thang
	Priority Queue class for A* search	Hoang Anh Chung
	A* search	Hoang Anh Chung (60%) Bui Manh Cuong(40%)
	Appendix A. Generating input	Hoang Anh Chung (50%) Bui Manh Cuong (50%)
	Merging codes on Notebooks	Ho Minh Khoi (50%) Le Tran Thang (50%)
Items for running the algorithm on notebooks	Input files and heuristic output	Mai Ha Dat (60%) Ho Minh Khoi (10%) Hoang Anh Chung (30%)
	Input images, heuristic explanation	Ho Minh Khoi
Video, pdf file of running algorithms		Le Tran Thang (50%) Ho Minh Khoi (50%)
Report	Comparison of heuristics	Ho Minh Khoi (60%) Hoang Anh Chung(40%)
	Analytics of result and the remaining components of the report	Mai Ha Dat (40%) Bui Manh Cuong (40%) Ho Minh Khoi (20%)
Slides		Mai Ha Dat

8, List of bibliographic references

1, Stuart J. Russell and Peter Norvig, Artificial Intelligence : A Modern Approach ,Third Edition

2, Muriel Visani, Slides of Introduction to Artificial Intelligence Course