Project Report

There are 3 abstract files in the generic package; Node,State and General Search:

1. The **Search Problem** which is the search problem ADT is a that class contains list of operators of the search problem, a node representing the initial state of Ethan, a list of the state space and finally a state representing the goal state. It also contains all the abstract methods needed for solving the general search problem such as generalSearchProblem method, expandNodes method and methods representing each of the search strategies.
2. The **Node** which is the search-tree node ADT is a class that contains a state representing the node's state, the parent node of this node, the operator used to reach the state of this node, the node's depth, pathCost , heuristic and costAndHeuristic to be used in uniform cost search, greedy search and A* search respectively. It also contains abstract methods equals to compare two nodes together and toString to display the node.
3. The **State** class is a search-tree state ADT that contains an object Position that contains x and y of the state, a boolean variable isSubmarine to know if this state is a submarine or not, a boolean variable hasAgent to know if the state has an agent or not, a variable agentsLeftSoFar to indicate the number of agents left in each state, a list of agents (where an agent is an object that contains position and health value of the agent) that has not been carried yet.

The MissionImpossible class is a subclass from the General Search class. It has static values representing each of the information extracted (Ethan position, submarine position, agents position, agents health, truck size and the grade size) from the string grid. It also contain a 2D grid array containing the nodes representing the agents, Ethan and the submarine. It also has a variable that keeps track of the number of expanded nodes. It has getters and setters for all variables inherited from the general search class. It contains all the methods inherited from the general search class.

The main functions implemented are:
1. random(min, max): generates a random number in the range(min --> max)
checkIfTaken(Position[], Position): checks if the position is taken previously
InterpretGrid(String grid): extracts information from the grid
gridTogrid2D(): converts the grid string to a grid of nodes, indicating each node with it's attribute whether it's Ethan, an Agent, or the Submarine.
2. statePresent(ArrayList<MissionImpossibleNode> states, MissionImpossibleNode s): Checks whether a state, represented with a

node, is present in the given array of states, represented with an array of nodes. This is a part of how the checks of the repeated states is handled

3. isParent(MissionImpossibleNode n1,MissionImpossibleNode n2): Checks whether a given node, is the parent of the other
4. goalTest(MissionImpossibleNode node): Checks whether a given node is a goal node. A goal node is a node that contains the submarine, and it's state indicates that there are no agents left to rescue.
5. solve(String grid, String strategy, boolean visualize): The main function of the project. Given a search strategy, and a search problem represented with a String outputted from the genGrid() function… outputs the solution to this problem in the form of a path between the start and the goal nodes. It uses the method generalSearchProblem to pass it the strategy and the truck size.
6. generalSearchProblem (String strategy,int truckSize,): The second main function of the code, where given a strategy of the 6 strategies and a truck size, and applies the strategy on the problem.
7. expandNode(MissionImpossibleNode node, int agentsCollectedSoFar,int truckSize,String strategy): expands a certain node to its children "operators" of all 6 types of them.. up, down, left, right, carry, and drop. And returns the set of children nodes at the end.
8. iterativeSearch(int truckSize): The method concerned with the Iterative Deepening Search Strategy.
9. depthSearch(int truckSize): The method concerned with the Depth First Search Strategy. It calls the generalSearch Problem
10. breadthFirst(int truckSize): The method concerned with Breadth First Search Strategy. It calls the generalSearch Problem
11. uniformCostSearch(int truckSize): The method concerned with Uniform Cost Search Strategy. It calls the generalSearch Problem
12. greedy1(int truckSize): The method concerned with Greedy Search Strategy, using the first heuristic function. It calls the generalSearch Problem.
13. greedy2(int truckSize): The method concerned with Greedy Search Strategy, using the second heuristic function. It is called inside of the generalSearch Problem.
14. as1(int truckSize): The method concerned with A* Search Strategy, using the first heuristic function. It calls the generalSearch Problem.
15. as2(int truckSize): The method concerned with A* Search Strategy, using the second heuristic function. It calls the generalSearch Problem.

The implementation of the search algorithms was done using the generalSearchProblem method as described in the lecture. The generalSearchProblem creates a queue and a list of visited nodes to store the nodes visited. The method loops on the queue of nodes until it is empty.For each it checks if it passes the goal test or not and if it did it returns this node. If the node didn't pass the goal test the method calls expandNode method to expand the nodes to its children but this step is done if the node is not a repeated one. When the agent carries and drops it resets the list of visited nodes since the state has changed.

The heuristics and the path cost functions are implemented as follows:

1. pathCost here we calculate the cost by adding the damage that each agent on the board will take multiplied by a factor in order to make tha path with the highest deaths have higher cost. This factor is 0.5 to the power of ceil[((100 - current health of the agent)/10)].So the agents that have health in the range 90>=h>100 for example will be damage taken * 0.5*(1) and agents that have health in the range 50>=h>60 will be damage taken * 0.5(5). So this formula will give higher weight to the agents with higher health. The damage is calculated as 2 the manhattan distance between the node and the agent.

2. getHeuristic1 here we calculate the heuristic the same as the path cost but the difference is when calculating the damage instead of using manhattan distance, euclidean distance is used. So at anytime the heuristic will always be less than or equal to the path cost in the case that the manhattan distance is equal to the euclidean distance. So it is admissible.

3. getHeuristic2 calculates the heuristic the same way as the getHeuristic1 but the difference is instead of taking into consideration all the agents and adding their damage*factor to the heuristic, here we take into consideration X number of agents where X = truckSize. So for example if the truckSize is 3 we will only consider 3 agents in the heuristic function and the heuristic would be calculated as getHeuristic1 so at any time the getHeuristic2 would be admissible as the maximum of truckSize would be the number of agents available on the grid so would be the same as getHeuristic1 which is admissible.

Two working examples:

1. Running Iterative Deepening on the following grid:

Input :
```
10,10;6,3;4,8;9,1,2,4,4,0,3,9,6,4,3,4,0,5,1,6,1,9;97,49,25,17,94,3,96,35,98;3
```

Output :
```
right,carry,right,right,right,right,right,down,left,left,left,left,left,left,left,left,left,do
wn,right,right,right,right,right,right,right,right,right,down,left,left,left,left,left,left,l
eft,left,carry,right,right,right,right,right,right,right,right,up,left,left,left,left,left,left,
left,left,left,up,right,right,right,right,right,right,right,right,right,up,left,left,left,left,l
eft,left,left,left,left,up,right,right,right,right,right,right,right,right,right,up,left,drop
,right,left,left,left,left,left,left,left,left,left,carry,right,right,right,right,right,right,rig
ht,right,drop,right,left,left,left,left,left,left,left,left,left,up,right,right,right,right,carr
y,right,right,right,right,right,carry,left,left,left,left,left,left,left,left,left,down,right,ri
ght,right,right,right,right,right,right,drop,right,left,left,left,left,left,left,left,left,left,
up,right,right,right,right,right,right,right,right,right,up,left,left,left,left,left,carry,rig
ht,right,right,right,right,down,left,left,left,left,left,left,left,left,left,down,right,right,
right,right,right,right,right,right,drop,right,left,left,left,left,left,left,left,left,left,up,ri
ght,right,right,right,right,right,right,right,right,up,left,left,left,left,left,left,left,left,l
eft,up,right,right,right,right,right,right,carry,right,right,right,carry,left,left,left,left,l
eft,left,left,left,left,up,right,right,right,right,right,carry,right,right,right,right,down,
left,left,left,down,right,right,right,down,left,down,drop;3;99,99,99,99,100,99,10
0,99,100;60665
```

2. Runny Greedy with heuristic function 2:

Input :
```
10,10;6,3;4,8;9,1,2,4,4,0,3,9,6,4,3,4,0,5,1,6,1,9;97,49,25,17,94,3,96,35,98;3
```

Output :
```
right,carry,right,right,up,left,left,up,up,carry,right,up,left,carry,right,right,right,
down,right,down,drop,left,left,left,left,up,up,right,right,right,right,down,right,
carry,left,down,drop,up,up,left,left,left,left,left,up,right,right,right,carry,left,left,
down,right,down,left,left,up,up,left,up,right,right,right,carry,down,left,down,
right,down,left,left,up,up,left,up,left,down,down,down,down,left,carry,right,right
,right,right,right,right,right,right,drop,up,up,left,left,left,left,left,up,right,right,
right,up,left,left,left,left,down,down,down,down,right,right,right,right,down,right
,down,right,down,right,down,left,left,up,left,left,left,left,left,left,left,down,right,
down,carry,left,up,up,right,right,up,up,up,right,up,right,up,right,down,down,
left,down,right,right,right,up,right,drop,up,up,left,left,left,left,left,up,right,right,
right,up,right,right,down,right,carry,left,down,down,down,drop;2;99,83,99,93,9
8,29,100,99,100;452
```

	Completeness	Optimality	RAM Usage	CPU Utilization	Number of Expanded Nodes
IDS	Yes	No	High RAM Usage. And it grows extremely High in the cases of the grid being of Big sizes.	Very High CPU utilization in terms of time complexity, and it might take more than 30 minutes to run some tests.	60665
GR2	Yes	Yes	Low	Low	452