Introduction to Artificial Intelligence, Winter Term 2022
Project Report of Project 1: Coast Guard

By: Omar Gamal

• A brief discussion of the problem.
The problem is about a rescue boat that is the responsibility of a coast guard force. The members of this force goes into the sea to rescue the sinking ships. During the rescue, all the survivors and the black box of the ship should be brought back.If the ship sinks before rescuing it, the blackbox should be rescued and retrieved before it becomes damaged. Each time step the rescue boat takes in its way to the sinking ship, the sinking ship loses one passenger and the blackbox gaines one damage point per time step from the ship being wrecked. The time step is counted increasingly every time an action is performed.
The goal is reached when there are no living passengers still not rescued, no damaged boxes not yet retrieved, and the rescue boat has no passengers on board. Best case scenario is to save as many people as possible and bring back the highest number of undamaged blackboxes.
The navigation area of the sea is an m × n grid of cells where 5 ≤ m, n ≤ 15. The grid elements follow some restrictions and different search algorithms can solve this problem.

• A discussion of your implementation of the search-tree node ADT.
I implemented the search tree node very closely to what was mentioned in the lecture in the sense that I declared the node's attributes to be a state which represents the location of the node, the parent which is the location of the parent node, the path cost which indicates the path cost to reach this node and heuristic cost1 which is the estimated black box health when the rescue boat arrives, and heuristic cost2 which represents the estimated deaths that will occur until the rescue boat arrives and action which is the action performed to reach the next node which might be close to the operator variable in Russell and Norvig's implementation however operator was the previous action that caused the current node to be reached. An equals method was implemented to facilitate comparing the current node and goal node in the strategy class.

• A discussion of your implementation of the search problem ADT.
The search problem takes an input of the world with all the elements interacting with each other and takes an input of the strategy in which the user chooses which strategy to apply to the given problem which ultimately affects the result of the problem.

• A discussion of your implementation of the CoastGuard problem.
I implemented the solution mainly in 2 classes, CoastGuard.java & World.java which have the logic of the game. I used the OOP to represent the items used in the game by classes such as the Ship, the Station, the Rescue Boat and the location. I also represented the values that will be fixed in ENUMs for easier handling such as the actions that will be taken by each ship, the different algorithms and the different states of the mission.

• A description of the main functions you implemented.
The solution consists of the following classes and functions:

- `CoastGuard.java`
  - `solve(..)`:
    - The solve function parses the information from the test cases and passes them to the world class

        - It also handle which Algorithm to run based on its parameter `search_algo_str` and map it to the defined enum value corresponding to each Algorithm then pass it to the World class object's constructor while its declaration
        - It declares an object from the World class then call its functions generate and solve
        - It returns the String value of the result in this format "`takenactions_list;numberOfSunkPassengers;numberOfSavedBoxes;nodes`" which corresponds to the format stated in the task "plan;deaths;retrieved;nodes"
- `World.java`
    - `World(..):` The world constructor handles the declaration of the necessary variables and initialises them, its parameters are:
        - *`search_algo: String stating which search method to use (eg bfs, dfs, ids)`*
        - *`visualize_mode: boolean to decide to visualise or not`*
        - `debugMode:` *`used for debugging`*
    - `generate(..)` The world generate function handles the creation of the grid, ships list and stations with the parameters provided from the test cases. It also calls inside it the 4 following functions declared in the same class.
    - `genGrid(..):` it constructs and object from the Grid class with the m,n cell values
    - `genRescueBoat(..)`: it constructs an object from the RescueBoat class with the grid_object, the coastGuard capacity and its x & y position that are passed as a parameters
    - `genShips(..):` it constructs number of ships objects from the Ship class and adds it to the ships_objectlist based on the number of ships and their location that are passed as parameters
    - `genStations(..):` it constructs number of station objects from the Station Class and adds it to the `ships_objectlist` based on the number of stations and their location that are passed as parameters
    - `solve(..):`
        - The world solve function handles the solving of the search problem and returns the solution to the test case provided
        - It has a while loop that will continue working as long as the mission is active by calling `isMissionActive()`
        - Inside the loop, it keeps in rescue mode if there are still passengers/boxes on ship and there is still room on the coast guard and calling the `setActiveShip()`. Otherwise, it will set the mission state to dropping and call `setActiveStation()`
        - Then it calls the `takeMotionAction()` to move the ship to the rescue position
        - Then it calls `takeRetrieveDropAction()` to either retrieve or drop
        - Finally, it calls the `updateWorld()` which updates all the variables that hold the different numbers needed for the logic (ex. Number of saved passengers, number of saved boxes,..etc)
    - `isMissionActive():`
        - The mission is considered active when the `totalNumberOfShipsPassengers>0 or totalNumberOfShipsBoxes>0 or rescueboat_object.getNumberOfPassengers()>0`

        - Otherwise, the mission is considered inactive and MissionState is set to DONE
    -
-
- `Algorithm.java`: An enum class that holds the different algorithms whether it is BFS, DFS, IDS, Greedy, A* or none.
- `Action.java`: An enum class that holds the different actions taken by the ships whether it is up, down, left, right, drop, retrieve or none.
- `MissionState.java`: An enum class that holds the different states of the mission, whether it is rescue, dropping or done.
- `Pair.java` : represents the x,y of the location
- `RescueBoat.java` : represents the rescue boat and its attributes which includes the capacity, the black box, the location,..etc
- `Ship.java` : represents the ship and its attributes which includes the number of its passengers, ship status, black box life,..etc
- `Station.java` : represents the station and its attributes which includes the passengers, the black box, the location
- `Display.java:this class is responsible for the creation of the GUI for the problem so that we can visualise the grid and the components interacting, the GUI was implemented using javax.swing`
- `Grid.java:represenets the grid with hashmap as the cells to minimise search time since it it o(1) also each cell in the grid is easily represented by the hashmap and the objects on the grid are placed as a String where location is the key that gets the string of the object placed in that location.this class is responsible for maintaining the grid such as the adding objects and removing objects.`
- `ManCtrl.java:this class is responsible for moving the objects manually strictly to facilitate debugging and allow a more in-depth visualisation of what is the problem at hand.`
- `Node.java:represnts the basic building for the search algorithms as each cell in the grid is represented as a node that the algorithm then acts on.`
- `Strategy.java:represents the different algorithms that can be taken to solve the coast guard problem which includes the algorithm, start node and goal node….etc. It also includes 2 classes which are used as the comparators for Heuristic functions 1 and 2 to be added in the priority queue based on the comparator value.`

• A discussion of how you implemented the various search algorithms.

- **I implemented the BFS algorithm by using a queue in which I enqueued a node and then check if it is the goal node then return all the nodes expanded however if it is not, all the adjacent cells are enqueued in the queue to ensure first in first out (FIFO) technique is used and the repeat the whole process until a goal node is expanded, if the algorithm failed the method returns false.**

- **I implemented the DFS algorithm by using a stack in which I pushed the start node and checking if it is the goal node or not, if so all the expanded nodes are returned and if not the adjacent cells are then pushed into the stack t ensure that last in first out**

(LIFO) technique is used so that the depth is explored before the other operations that can be performed.

- **I implemented the IDS algorithm by using a stack in which I pushed the start node and checking if it is the goal node or not, if so all the expanded nodes are returned and if not the adjacent cells are then pushed into the stack t ensure that last in first out (LIFO) technique is used so that the depth is explored.The depth is checked at each cycle to ensure that the algorithm won't run forever and that all the operations at the same depth is executed first and thus the algorithm proves its characteristic of being complete**

- **I implemented the first greedy algorithm by enqueuing the current location as the start node in a priority queue and then checking the current location refers to the goal node then return the expanded nodes however if not, then all the adjacent cells are enqueued in the priority queue and sorted based on the heuristic cost which in this case is the estimated blackbox health when the rescue boat has reached the goal node, the blackbox health is returned from the blackbox heuristic function. This way I can ensure that nodes with lower heuristic cost are expanded first, the whole process is repeated until the goal node is reached, if the algorithm fails the method returns false. In test e5 for example the rescue boat prioritised the ships with the lowest estimated death, however in test a5 the rescue boat prioritised the closest ship. And on the second trip from his way back from the station the algorithm also favoured cell 1 and 2 to save more people than the previously prioritised 0 and 3 location ship.**

- **I implemented the second greedy algorithm by enqueuing the current location as the start node in a priority queue and then checking the current location refers to the goal node then return the expanded nodes however if not, then all the adjacent cells are enqueued in the priority queue and sorted based on the heuristic cost which in this case is the estimated deaths to occur until the rescue boat reaches the goal node, the estimated deaths is returned from the deaths heuristic function. This way I can ensure that nodes with lower heuristic cost are expanded first, the whole process is repeated until the goal node is reached, if the algorithm fails the method returns false.**

- **I implemented the first Astar algorithm by enqueuing the current location as the start node in a priority queue and then checking the current location refers to the goal node then return the expanded nodes however if not, then all the adjacent cells are enqueued in the priority queue and sorted based on the heuristic cost which in this case is the estimated blackbox health + the cumulative path cost when the rescue boat has reached the goal node, the blackbox health is returned from the blackbox heuristic function. This way I can ensure that nodes with lower heuristic cost + path cost are expanded first, the whole process is repeated until the goal node is reached, if the algorithm fails the method returns false.**

- **I implemented the second A-Star algorithm by enqueuing the current location as the start node in a priority queue and then checking the current location refers to the goal node then return the expanded nodes however if not, then all the adjacent cells are enqueued in the priority queue and sorted based on the heuristic cost which in this case is the estimated deaths to occur+the cumulative path cost until the rescue boat reaches the goal node, the estimated deaths is returned from the deaths heuristic function. This way I can ensure that nodes with lower heuristic cost+path cost are**

expanded first, the whole process is repeated until the goal node is reached, if the algorithm fails the method returns false.

- **For the sake of comparing a star and greedy , grid5 was selected as a benchmark due to its relative complexity the following table compares the number of sunk passengers according to each algorithm as well as explored nodes and saved boxes. Neither DFS or IDS converged to a solution**

	Sunk passengers	**Saved boxes**	**Explored nodes**
BFS	**33**	**4**	**81**
GR1	**33**	**4**	**48**
GR2	**33**	**4**	**48**
Astar1	**33**	**4**	**54**
Astar2	**33**	**4**	**54**

• A discussion of the heuristic functions you employed and, in the case of greedy or A*, an argument for their admissibility.
The heuristic functions I implemented are :

- blackBoxHeuristicFunc:
  This function take the current location as a node and then calculates how many steps will be taken by calculating the absolute value of the distance in x-coordinate plus the absolute value of the distance in the y-coordinate in order to reach the goal node and as mentioned in the project description every time step or step taken, the health of the black box decreases by one and thus it calculates the estimated health of the black box when the rescue boat arrives and then the heuristic cost is calculated by removing the estimated health of the black box from the maximum health of the black box possible and thus representing the heuristic cost in an admissible way since no overestimation has occurred as every loss of health points is accurately calculated, in the case that the estimated health box will be 0 or less than 0 it doesn't make sense to visit this wreck and thus the heuristic cost is set to 1000 to indicate to the rescue boat that this action shouldn’t be performed.

- deathsHeuristicFunc:This function take the current location as a node and then calculates how many steps will be taken by calculating the absolute value of the distance in x-coordinate plus the absolute value of the distance in the y-coordinate in order to reach the goal node and as mentioned in the project description every time step or step taken,a passenger dies and thus it calculates the estimated deaths when the rescue boat arrives and then the heuristic cost is the same the number of deaths and thus representing the heuristic cost in an admissible way since no overestimation has occurred as every death is accurately calculated.

I used some built-in functions and supporting classes to help in the program implementation such as:

- `StringJoiner:` to concatenate the values returned in `solve()` function in this format "plan;deaths;retrieved;nodes"

I also implemented some helper functions such as:

- `convertListToString(List<Action> takenactions_list):` to convert the given list of items into one concatenated String
- `public double dist(pair startLocation,pair endLocation:` Closest distance function to get the ship or station, that are closest to the coast guard.

• A comparison of the performance of the different algorithms implemented in terms of completeness, optimality, RAM usage, CPU utilisation, and the number of expanded nodes. You should comment on the differences in the RAM usage, CPU utilisation, and the number of expanded nodes between the implemented search strategies.

Algorithm	Completeness	Optimality	RAM usage	CPU utilisation	# of Expanded notes
DFS	Not Completed	Not Optimal	Most	Most	2nd Most
BFS	Completed	Not Optimal	Most	Most	Most
IDS	Completed	Not Optimal	2nd Lowest	2nd Lowest	3rd Most
Greedy	Completed	Optimal	2nd Lowest	2nd Lowest	2nd Lowest
Astar	Completed	Optimal	Lowest	Lowest	Lowest

• Proper citation of any sources you might have consulted in the course of completing the project.

During the development of the project, I mainly depended on the project description document as the starting point to understand the problem, the requirements, the delivery package and the algorithms I should use to solve it. I have also used some public youtube channels, in addition to the lecture notes to apply the BFS, DFS, IDS, Greedy and AStar algorithms on grids, then I implemented them.I have relied on this stackoverflow post "https://stackoverflow.com/questions/15421708/how-to-draw-grid-using-swing-class-java-and-detect-mouse-position-when-click-and" as a boilerplate code for the display code of the grid and the manual control code. However they were heavily modified by me to suit the purpose of the project.

• If you use code available in library or internet references, make sure you fully explain how the code works and be ready for an oral discussion of your work.

I did not use any open source code in the course of the development of this project.

• If your program does not run, your report should include a discussion of what you think the problem is and any suggestions you might have for solving it.

Everything worked fine.

Notes:

DFS was the highest cpu.
BFS was the highest memory
Greedy was the most optimised
Atsar better than bf and df
Ids better the df due to being complete
For some grids there are out of bounds locations for ships coast guards or stations.

-