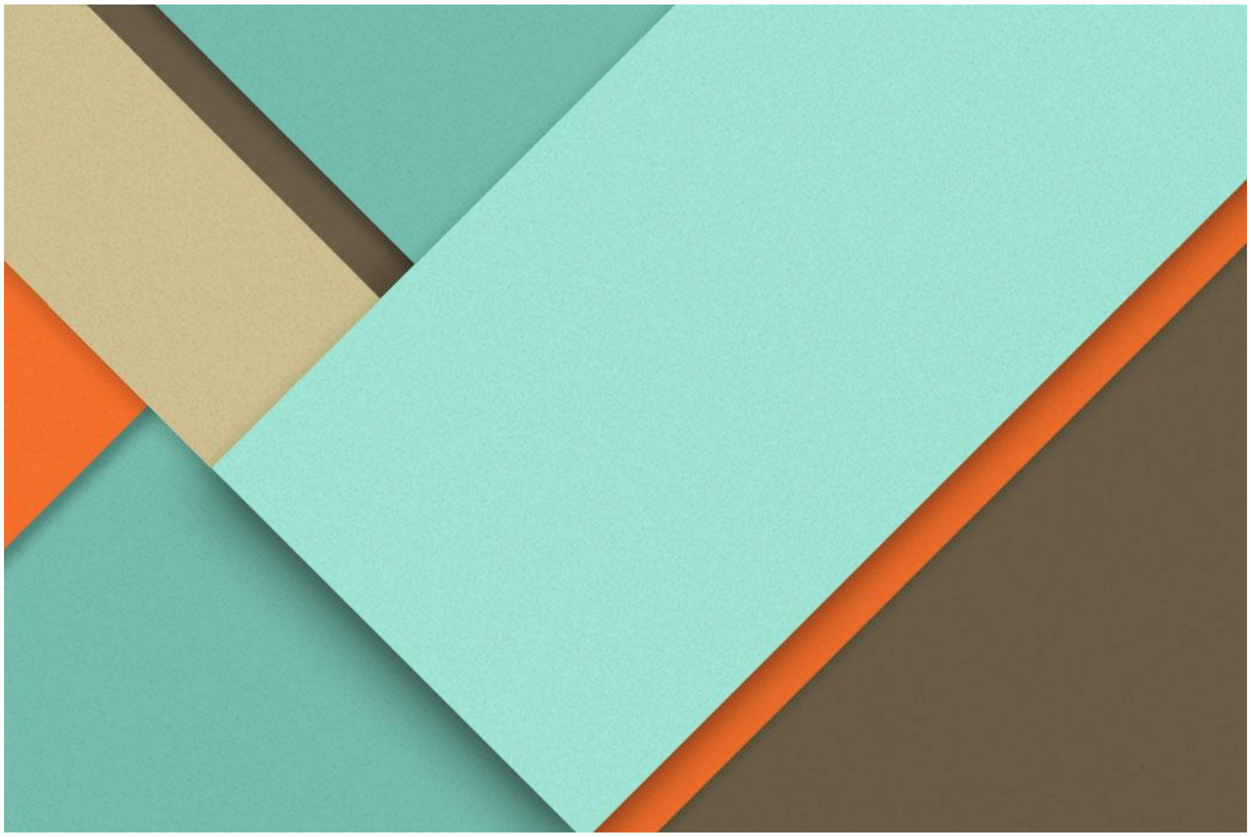

AI Project 1 Report

16.12.2022

Shahd Koura	46-1105
Amira Hossam ElDin	46-1073
Mohannad Osama	46-2021

Discussion of The Problem

The project represents a search optimization problem, where we want to search for sinking ships and rescue as many passengers on these ships before they die or "Time Out". Moreover, each ship has a box that we want to retrieve. However, Rescuing the passengers is of a higher priority. Our goal is to rescue and collect all what can be saved before it's too late (passengers and black boxes). We will go through multiple search strategies to know which finds the optimal solution to our search problem. Our agent will need to drop off all passengers to a station inorder to be able to rescue the rest of the passengers on the other ships. Our agent will have either failed or succeeded according to how many passengers were dropped off before all is lost.

The Search-Tree Node ADT

Our node is a class that contains multiple variables that helps us track the expansion sequence of each node that will be discovered. The node includes the parent node, state in which we are in, depth of the node in the tree and finally the cost of each node. The cost will be used in greedy search and A* search strategies later on.

Search Problem ADT

We created a cell class in which we use to create our grid. In each cell we have boolean markers to mark if this cell has a station or a ship in it so that we can compare their location to the location of the agent. We keep track of all the actions that lead to this cell/node in the search problem. The actions are represented as an arraylist of strings in which we concatenate the operators that lead to this node. We then print this arraylist as our final result whenever we reach our goal test.

CoastGuard problem

In order for the Agent to be able to search for the ship and stations on its path we have created a class for the agent, ship , agent-state, cell. Agent class keeps track of the position of the agent on the grid and how many passengers are saved. Ship has its location, box-state - which is an enum that tells us the state of the box- and other variables. Agent state is the class we use to create a new node for each expansion sequence. Lastly the cell class is the base unit of our grid. We have boolean variables that we use to check if that cell has a station or a ship. The main class “Coast Guard” keeps track of all ships and stations using an array list of type ships in case of ships and objects in case of stations.

We create a new node for every path taken to keep the track of our variables in each state. We modeled our problem so that whenever the agent finds a ship, it picks up the passengers until the capacity of the agent is full. If all the passengers are picked up then we can collect the box on the ship. Whenever the agent finds a station we drop off the passengers.

Main Functions Implemented and The How

I. Solve

The Solve method takes the grid as a string and the search strategy that will be used in the problem. We switch on the strategy and accordingly we call its function. This function returns the path taken, total deaths and No. of expansion nodes

II. BFS

BFS creates a queue that whenever an action is possible for the agent it is enqueued to be further explored. We keep on dequeuing the states and enqueuing till we reach our goal test.

III. DFS

DFS is pretty much the same as BFS with the core difference of instead of a queue we push the nodes in the stack.

IV. ID

Iterative deepening utilizes the depth in the nodes so that whenever the current depth is equal or more than the depth , it resets the depth by popping all the nodes in the stack and restarting the search problem again and so on until we reach our goal.

V. Greedy

The greedy search algorithm is a priority queue based search which enqueues its nodes based on a heuristic function. We have 2 greedy methods one for each heuristic

VI. A*

The A* search algorithm is a priority queue based search which enqueues its nodes based on a heuristic function. We have 2 A* methods one for each heuristic

VII. Damage

The damage function acts as the timer in our problem, in which for every movement the agent does all the passengers on all ships get decremented by 1. If a ship has no more passengers “becomes a wreck” a timer for the boxes begins and when that timer reaches 100 the box is lost.

Heuristic Functions

We have created 2 admissible heuristic functions for both the greedy.

1. Based on number of ships available (is not a wreck)

We want to take the path which will lead to the most saved passengers and thus whenever a ship has zero passengers it becomes a wreck either from saving them or them dying so we want to choose the path with the most available ships in order to rescue the most passengers.

2. Based on count c, where c = sigma(ships.getPassengers / agent.capacity)

This heuristic will assign a cost to each node so that the path taken is optimal. Because if a ship has more passengers on it than what the agent can carry this will cause more death to occur than going for a ship in which we can rescue all the remaining passengers with the agent’s capacity. This way ensures that we will never overestimate the cost as we are looping on the ships in each node to calculate our cost.

Performance Comparison

BFS: BFS uses a larger amount of memory because it expands all children of a vertex and keeps them in memory. It stores the pointers to a level's child nodes while searching each level to remember where it should go when it reaches a leaf node.

DFS: The DFS generally needs less memory as **it only has to keep track of the nodes in a chain from the top to the bottom.**

ID: an iterative graph searching strategy that takes advantage of the completeness of the Breadth-First Search (BFS) strategy but uses much less memory in each iteration

Greedy:

Astar: