A* algorithm flow:

Iterative part of A* algorithm initializes open(boolean matrix) and closed(Priority queue) sets. Then puts the start Node to the open set. Then it starts to take the elements with minimal f value, while Priority Queue is not empty. f = g + h where g is the length of path to the current cell, h is the heuristic length of the path from this cell to target. On each step A* marks current Node as present in the closed set. Visits all its valid neighbors if they are not in the closed set, calculates g and f values and adds these neighbors to the open set and performs all the checks.

If the neighbor cell is our target then we save the value of kraken alive flag from the last cell in the path set flag that any path is found and return. If the neighbor cell danger level is 1(the only one enemy perception zone or one enemy) we check whether we killed kraken and if the kraken is killed then we can check whether it is Kraken or not. If it is Kraken or his perception zone and Kraken is killed, we can visit this cell and add it to the open set. If we have rum but kraken is not killed we can check whether this cell is on diagonal to the Kraken and if it is we can save that Kraken is killed for this cell. If the next cell is just safe we can check whether this cell is Tortuga or not, and if it is, we can set the value of this neighbor cell haveRum to true. On each iteration all the neighbors save the parent from where we came from and save the parents flags(krakenAlive and haveRum).

The driver part of this algorithm executes A* for two different paths (through Tortuga and simply without Tortuga straight to the goal) and looks for the shortest one, if any is found. Otherwise, it just returns the empty list because the path does not exist.

Backtracking algorithm flow:

Recursive part of backtracking algorithm is quite similar to the A* algorithm. However, there is a row of dissimilarities. For example, we have two recursive methods, the first one checks if any path exists. The second one is executed after the success of the first one because it is not safe and can lead to infinite recursion in case of absence of the path to the goal.

Backtracking algorithm in my case is an improved DFS algorithm that performs additional task-based checks and uses heuristic functions to check if the current branch length is + heuristic

function result is larger than the length of shortest path, and if it is, it simply backtracks. Moreover, we save the minimal value for each visited cell and use it to backtrack from other branches that have larger or equal length of path + length of the heuristic path to the target and we backtrack if the length of the current path + heuristic function result exceeds the 24 because 24 is maximal length of the path for given by task conditions (This number was found by exhaustion of all the possible maps and execution of A* algorithm on them). Backtracking recursive part marks current node as visited visits all the unvisited neighbors and performs the same checks as for the A* (checks on Kraken etc.). However, if we find a goal we do not return, the algorithm just updates the best found path and marks that any path is found. Moreover, flags (haveRum and krakenAlive) are being sent to recursive calls and used further inside the called function execution.

The driver part of an algorithm tries to find the direct way from start to target and if it exists, looks for the shortest path and saves it. Then it tries to find the way indirectly(through Tortuga) it also checks if the way to Tortuga exists and way from Tortuga to target exists, looks for the shortest paths for both these segments and saves them. At the end of the day, it compares the lengths of both found paths (if any exists) and returns the shortest one.

PEAS:

Performance measure:

The shortest path is found correctly and programs execute quickly.

Environment:

9 by 9 map with obstacles.

Environment properties:

1. Partially observable (limited by spyglass)
2. Multiple Agent (other agents are Kraken and Davy Jones)
3. Deterministic (we can determine the next state of the environment)
4. Sequential (current decisions have consequences on future actions)

5. Static (because all the object keep their position while main agent is thinking)
6. Known (because we know all the rules of the environment)
7. Discrete (because we have step-by-step movement)
Actuators:
Ship, rum throwers.
Sensors:
Spyglass and compass.

Statistical analysis:

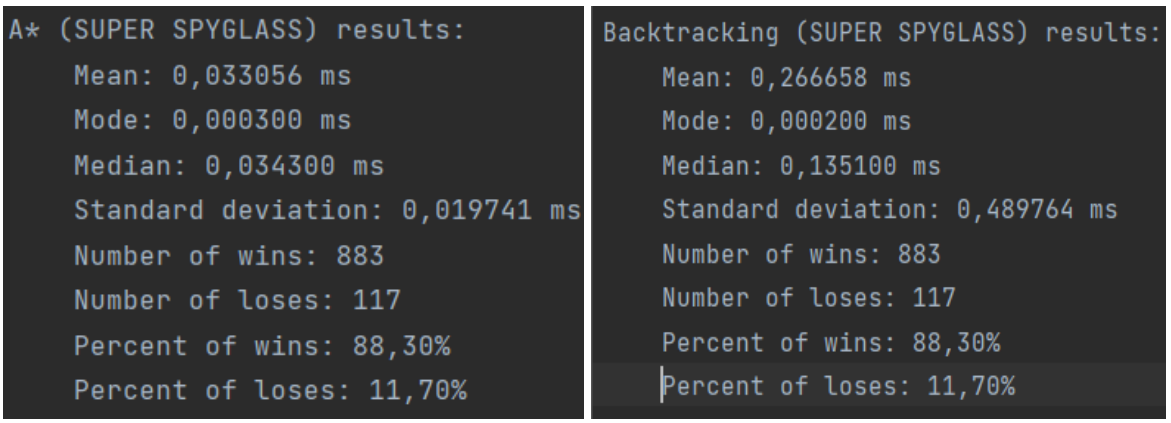
- Backtracking (variant 1) compared to A* (variant 1):

```
A* (SPYGLASS) results:
    Mean: 0,039795 ms
    Mode: 0,000200 ms
    Median: 0,027400 ms
    Standard deviation: 0,099558 ms
    Number of wins: 883
    Number of loses: 117
    Percent of wins: 88,30%
    Percent of loses: 11,70%
```

```
Backtracking (SPYGLASS) results:
    Mean: 0,423855 ms
    Mode: 0,000200 ms
    Median: 0,213250 ms
    Standard deviation: 0,577845 ms
    Number of wins: 883
    Number of loses: 117
    Percent of wins: 88,30%
    Percent of loses: 11,70%
```

According to these results A* works significantly faster than the backtracking algorithm (mean of first one is ten times less than mean of the second one). The median for the A* not ten times smaller than the median of backtracking (bit less than ten times), that says that a* starts its execution a bit slower than backtracking (allocates memory for it). For backtracking the standard deviation is quite big so we can say that it has a lot of cases where it executes two long or terminates very fast. Mode is the same for both algorithms so we can say that in most of the cases (cases when we have no path) both algorithms terminate quickly.

- Backtracking (variant 2) compared to A* (variant 2)



Here we can see an artificial improvement for the A* because Super spyglass doesn’t affect the efficiency anyhow. We