Report.md

Description of using algorithms:

- backtracking:
  - Description: naive implementation of backtracking algorithm without any heuristics. Slow, not optimised algorithm, that is not recommended using for traversing graphs.
  - How backtracking works: from each point it chooses some point in the neighborhood, adds it into visited path variable and then picks another neighbor and then traverse from it. After finding the neighboor checks that after stepping on that field the actor won't become infected. Also, I store variable Resistance that holds information about picking up mask or become vaccinated.
  - Handling infected & healing cells: information about resistance of COVID-19 is stored in a special variable called Resistance and it's value is updated after each step
  - Pseudocode:

```
traverse(From):
       if actor != home:
               pick a Neighboor of actor,
               make sure that after stepping of this field actor won't become infected,
               if actor picks an mask and visited doctor ->
                       update Resistance variable,
               append Neighboor to visited path variable,
               traverse(Neighboor).
       print: Found path
```

  - Execution time - extremely long, appr. up to 7 min on 9 by 9 field.
  - Comments. Due to great difference in execution time not recommended for usage
- dijkstra:
  - Description: this is optimised implementation of backtracking algo. This is fast algorithm, thus is ideally works for finding paths in 2 or 3 D game fields.
  - How algorithm works: as a global variable (fact) I store a set of all visited paths from initial position to some vertex and min cost, needed to traverse there. As it is known, dijkstra works only on weighted graphs, so I converted game field map into weighed graph, where we can traverse from point to its neighbor. At each step we traverse from point to its neighboor, calculate cost to travel there and if cost is less that cost from all visited paths fact, updates value there.
  - Pseudocode:

```
traverse(From, CurrentCost):
       if From != Home:
               pick a Neighboor of actor,
               checks that Neighboor is not visited earlier,
               checks that actor won't became abfor stepping on this field
               append to the set of paths tuple (Neighboor, CurrentCost+D),
               if there already exist path containing Neighboor:
                       if Cost < CurrentCost + D:
                               drop tuple (Neighboor, NeighboorCost)
               traverse(Neighboor, Cost)
       else:
               print: "End of algo!"
```

  - Execution time: approx. 250ms. for 9 by 9 field.
  - Comments: this is example of good algorithm that efficiently founds the path from initial position to home. This algo is more preferable rather than by using naive backtracking.

I tested programs on n=5 examples.

Statistical analysis:

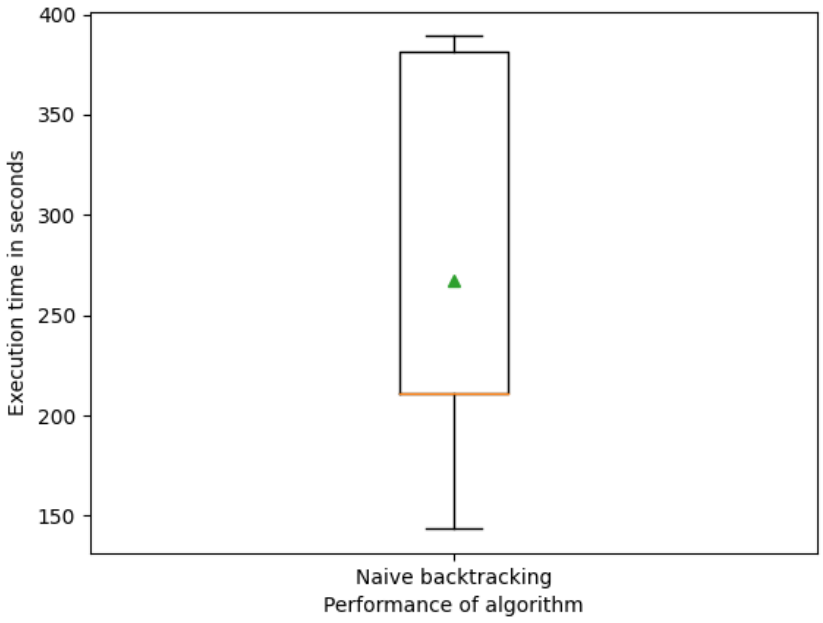
- backtracking
  - range = 1
    - mean = 266.386
    - std = 99.500
- dijkstra:
  - range = 1:
    - mean = 0.391
    - std = 0.049

Testing my program on range = 1 or range = 2 I did not notice the difference:

- backtracking:
  - this algorithm checks all paths that exists on the field and chooses within them the minimum path - so this algo must traverse through each unvisited point and find the minimum path.
- dijkstra:
  - this algorithm should also traverse through each point to fling the minimum distance within init position and any other point, thus, the algorithm ought to visit any point and check each (minimum) path from init to home position.

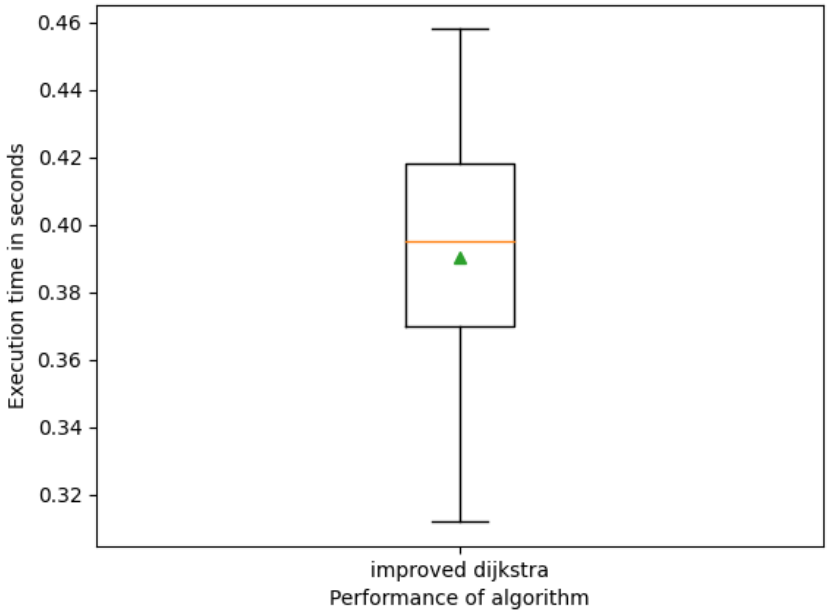
Box plots:

- backtracking



{width="300", height="300"}

- dijkstra



{width="300", height="300"}

```
Values:
        - back(1):
                * [211.377, 381.588, 143.448, 389.14, 211.377] s

        - dijkstra(1):
                * [0.370, 0.395, 0.458, 0.418, 0.312] s
```

Calculating p-values:

- back(1) vs dijkstra(1)
  - p_value = 0.0007 <- The most significant p value, so we could discard null hypothesis that means of execution time are the same for both algo. is the same.

PEAS model of actor:

- agent type:
  - actor (player)
- performance measure:
  - found a minimum path from actor to home
- actuators:
  - making (moves) on the game field
  - ability to collect a mask or be vaccinated
- environment:
  - field map with COVID-19, mask and doctor zones
- sensors:
  - information about current position (relative position to the game field boarders)
  - distance to home field
  - information about infected cells when staying close to them

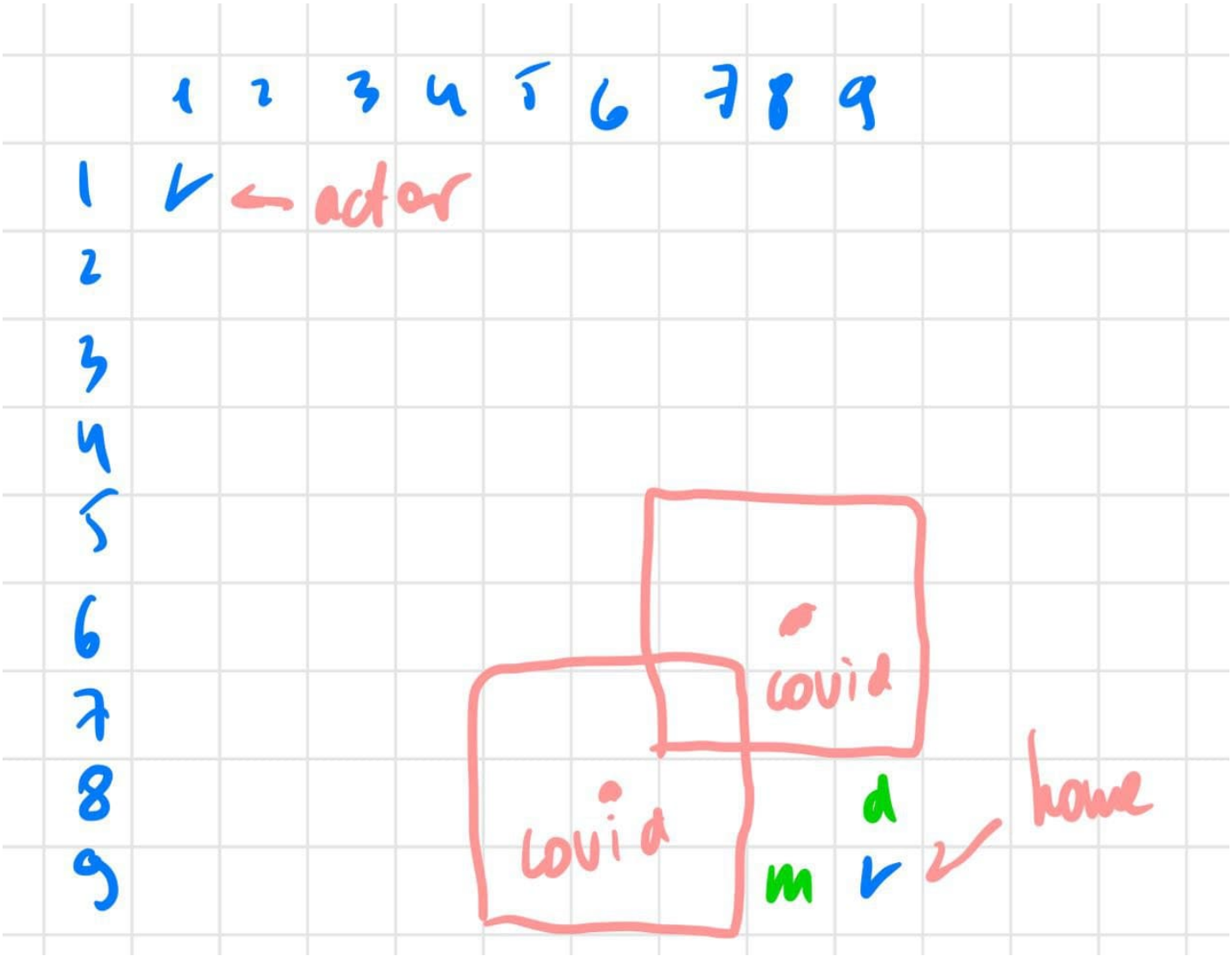
Environment is:

- partially observable
  - actor does not have information about all cells in the map

- deterministic
  - game does not contain any random elements - any change in-game field is fully controlled by the player move
- Single Agent
  - there is no other player, only one agent is playing
- sequential
  - all moves made by AI algorithms will affect future steps
- discrete
  - there are states in the environment (positions of figures), in this game, there are a finite set of steps, that players can perform
- known
  - there are a complete set of rules of how all figures can move
- static:
  - only the actions of an agent modify game field. There is no other processes are operating on it.

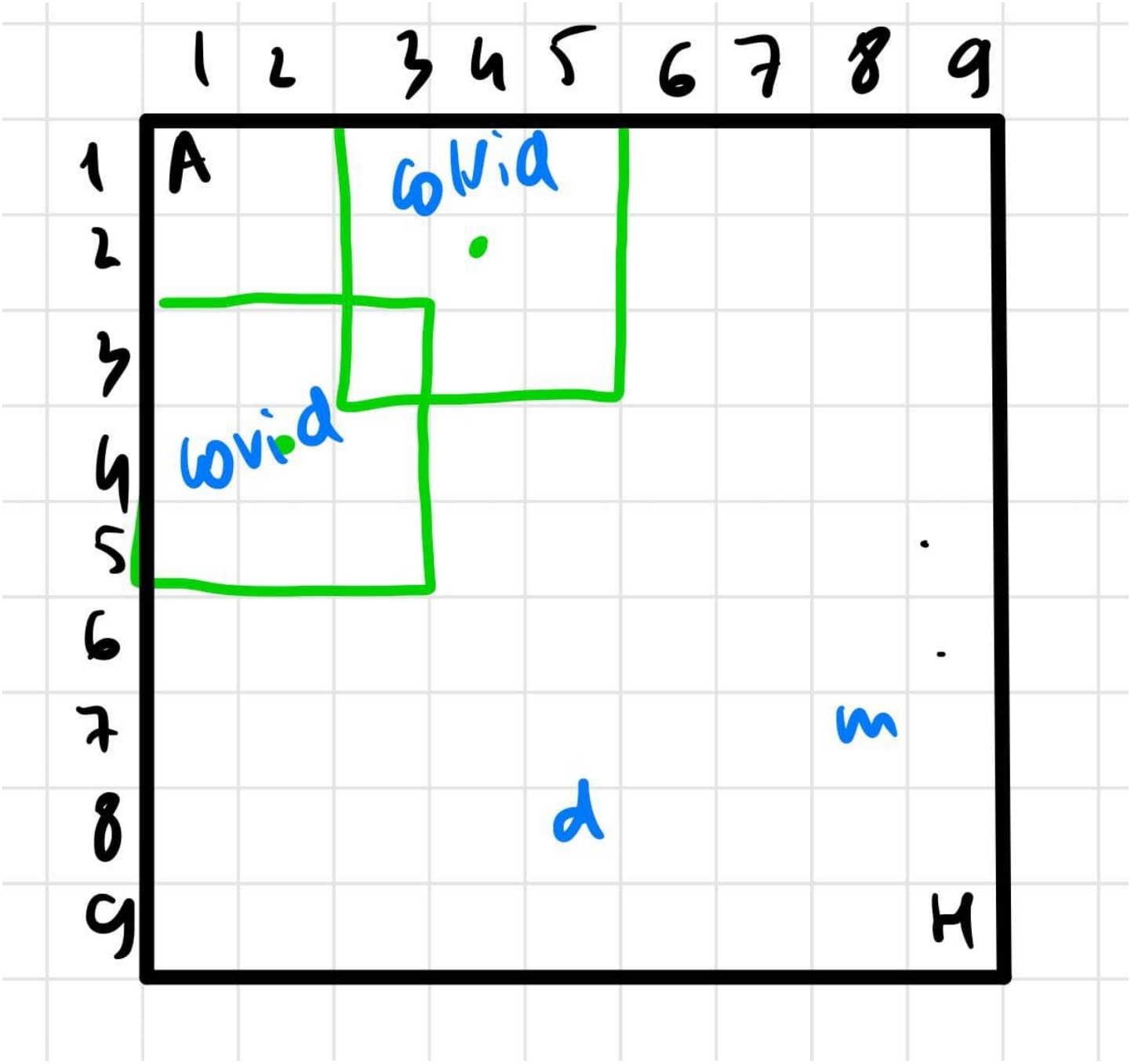
Impossible paths:

- Map 1 (doc, mask, home is surrounded by COVID-19), tested on both algorithms (execution time on backtracking is ~ 12 min.)



{width="300", height="300"}

- Map 2 (actor is surrounded by COVID-19), tested of both algo (execution time < 1s)



{width="300", height="300"}