Introduction to Artificial intelligence Assignment 1. Humans vs. Orcs Rugby

Innopolis University, Spring semester

Lev Svalov
BS-2018, group 5.

March 2020

Contents

1 Introductory setup

1.1 Prolog setup

I have implemented my project using the SWI-Prolog with the following version:
SWI-Prolog version 8.0.3 for x86 64-darwin
(*copypasted from the terminal, just to be sure that project will work on your machine.*)

1.2 Project structure

The project consists 3 files

- **input.pl** - the file with input predicates within the declared format.
- **map.pl** - auxiliary file with map predicates.
- **main.pl** - the core file where all rules are defined.

To **run** the project you need to do the following:

- Open **swipl** in the terminal.
- Load knowledge base - '[**main**].'
- Run the rule - '**start.**'

2 Description of implementation concepts

2.1 General assumptions and ideas of my implementation

- The map is drawn by 3 defined rules **draw, draw2 and draw3** for better visual understanding, but I need to notice that it has a little ~~minus~~ feature that it is drawn on vertical axis symmetrically in different way, like (0,0) is left-upper angle, but needed to be left-bottom. That's totally ok, as the drawing is for visuality.
- The current cell of the player is storing in the predicate **current(X,Y)**
- About the input. The input itself contains in **'input.pl'**, but during the execution, the information about where touchdown/orcs/humans are located is converted in the format: **coordinate(X,Y,[Coondition])**.As you can see in '**map.pl**' condition of each cell is initially set up as *free*, it changes when **'input.pl'** have been read.
  The optimality of such format is fairly arguable, but it works and this is just a subjective point of implementation. (**"This Is the Way"** ©The Mandalorian)
- The direction of the action*(pass or move)* is determined by value of variable with name*(in most of cases)* **Direction**
  There can be the following values:

- **1** - pass/move to the **left**.
- **2** - pass/move **up**.
- **3** - pass/move to the **right**.
- **4** - pass/move **down**.
  Next values are for diagonals and possible **only** for pass action.
- **5** - pass **left-up**.
- **6** - pass **right-up**.
- **7** - pass **right-down**.
- **8** - pass **left-down**.

2.2 Description of random method implementation

My implementation of random method is based on the following concepts:

- The main rule - '**random search**' is attempting rule '**random algo**' 100 times and finds successful one with the shortest length.
- '**random algo**' is loop rule that has 2 implementations:
  first one checks whether the current cell is touchdown or not.
  second one does the action, but before it randomly decides what kind of action to do: **pass***(if pass was not attempted already in this round)* or **move**
- As player have reached the touchdown point during the particular round, it calculates the length of the path and compares it with current minimum length, if it is shorter, it stores the path into the predicate '**min path**' and this action for 100 times.
- As all 100 attempts are proceed, it prints the minimum path and the length of the path.

2.3 Description of backtracking method implementation

My implementation of backtracking method is based on the following concepts:

- I am looking for **the first** successful backtrack solution, **not** the optimal one. For this, I use in-built backtrack predicate **findnsols(n,...)**. It is analog of **findall(...)** predicate, but that finds at most n solutions, in my case, $\mathbf{n = 1}$.
- As a **goal** of backtrack - player needs to become on touchdown position, as **templates** I provide listing of all possible actions that can be applied to move from one cell to another.
- As player have reached touchdown position, it saves the path and calculates its length using predicate **save path**. After that, it prints in a common format for all 3 methods.

2.4 Description of heuristic function method implementation

- The heuristic function method is based on the proposed subjective concept of implementation of heuristic function.
- The idea of the heuristic function is in the following:
  Each cell of the map has the **rank**, that indicates the amount of unknown information agent(player) can gain if it will move to this cell.
  In particular: **rank** of the cell is **number** of neighbour cells(with **no** respect to diagonals) that have not been visited yet by the agent. This number is in the interval - [0,4]. For example, for angle of the map, the maximum rank = 2, as it has only 2 neighbours. For cells that are near border, but not in angle, maximum rank = 3. For others, = 4.
- In the very beginning of the program, during the drawing of the map, the predicate **rank** is initiated for each cell with maximum rank value for the corresponding cell. When algorithm is processing, after the player has visited some cell, for this cell and all 4 its neighbours, the rank decreases by 1.
- **The important notion** is that: since the calculations of the rank for each cell are time-consuming enough, therefore the execution time of the method is expected to be a bit bigger in comparison with the **Backtracking** method.
- The algorithm is greedy one and the main idea is that: the player decides to move to the adjacent cell with maximum rank. And this is proceeding until it reaches the touchdown position.

3 Part 1. The analysis of results

3.1 Explanation of decisions on developed maps

To analyze methods properly, we need to generate maps that are different on the several criteria: The generated maps of my project can be specified by the following points:

- **The density of the map.** How bushy player/orcs are located to each other? Does there exist many empty holes*(where player can uselessly walk)* on the map or not? This kind of questions is present on this criterion.
- **The distance between start and touchdown position.** How far player is located to the touchdown on OX axis as well as on OY?
- **The popularity of obstacles(orcs) and teammates.** With this criteria we understand is there needed pass to get the shortest path or not? If majority cell of the map with orcs, probability of successful pass is not high enough, e.t.c.

According to the criteria, 5 sample-maps have been generated, also we have one from the description of the assignment.
Next, I will analyze results(number of actions and execution time) on the each map.

3.2 Analyzing results

Notion: All results easily can be tested on your own(there will be exact solution path and results)
All maps are stored in **input.pl** *(commented, for better understanding).*
Here, **only** result of my particular try.

3.2.1 Default map

```
Hey! Here the map:

S _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _
H O _ _ _ _ _ _ _ _
_ T _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _
```

The results are:

- **Random** - Actions: 26 , Time: 0.008 sec.
- **Backtracking** - Actions: 22 , Time: 0.001 sec.
- **Heuristic** - Actions: 18 , Time: 0.002 sec.

As we can see **Backtracking** is fastest, but **Heuristic** is better in terms of number of actions. It can be explained by the fact that the map is sparse one, have a lot of empty space and **random** method is almost always worse than 2 other methods.

3.2.2 Map 1

```
Hey! Here the map:

S _ _ _ _ _ _ _ _ _
_ _ O _ _ _ H _ _ _
_ _ _ _ _ O _ _ _ _
_ _ H _ _ H _ _ _ _
_ _ _ _ H _ _ _ _ _
_ _ _ _ _ O _ _ _ _
_ _ _ _ _ _ _ _ _ O
_ _ _ _ O _ _ O _ T
_ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _
```

The results are:

- **Random** - Actions: 151 , Time: 0.020 sec.
- **Backtracking** - Actions: 46 , Time: 0.002 sec.
- **Heuristic** - Actions: 31 , Time: 0.004 sec.

As we can see **Backtracking** and **Heuristic** are almost the same in terms of time, but **Heuristic** has less amount of actions. The map is dense and the touchdown is located quite far away from the starting point, so that random has quite bigger number of actions and execution time is longer as well.

3.2.3 Map 2

```
Hey! Here the map:

S _ _ _ _ _ _ _ _ _
_ _ _ O _ _ _ _ _ _
_ O _ _ _ _ _ _ _ _
_ _ O _ _ O _ _ _ _
_ _ _ _ O _ _ O _ _
_ _ _ _ O _ _ _ _ _
_ _ O _ H _ O H O _
_ _ _ O _ _ _ _ _ _
_ _ _ _ O _ T _ _ H
_ _ H _ _ _ _ O _
```

The results are:

- **Random** - Actions: 7 , Time: 0.008 sec.
- **Backtracking** - Actions: 58 , Time: 0.002 sec.
- **Heuristic** - Actions: 31 , Time: 0.004 sec.

As you can see , several diagonal with potentially successful passes are present on the map, that is why **Random** has the least number of actions, but, nevertheless, because of non-optimality of method, it took more time than other methods did. **Backtracking** and **Heuristic** are almost the same in terms of time again. **Heuristic** has less amount of actions, but on this map where distance between starting point and touchdown is bigger, the difference between these 2 becomes bigger as well.

3.2.4 Map 3

The results are:

- **Random** - Actions: 84 , Time: 0.008 sec.
- **Backtracking** - Actions: 72 , Time: 0.003 sec.

```
Hey! Here the map:

 S _ _ _ _ _ _ _ _ _
 _ _ H O _ _ _ _ _ _
 _ O _ _ _ _ _ _ _ _
 _ _ H _ _ _ _ _ _ _
 _ _ _ _ _ _ _ _ _ _
 _ _ _ _ _ _ _ _ _ _
 _ _ _ _ _ _ _ _ _ _
 _ _ _ _ _ _ _ _ _ _
 T _ _ _ _ _ _ _ _ _
 _ _ _ _ _ _ _ _ _ _
```

- **Heuristic** - Actions: 37 , Time: 0.004 sec.

On this sparse map, without possibility of getting advantage of passes(for long distance), **Random** has the worst results as for number of actions as for execution time. **Backtracking** and **Heuristic** have the same execution time, but for sparse map **Heuristic** method finds the shorter path to the touchdown point.

3.2.5 Map 4

```
Hey! Here the map:

 S _ H _ _ _ _ _ _ _
 _ _ _ _ _ _ _ _ _ _
 _ T H _ _ _ _ _ _ _
 _ _ _ _ _ _ _ _ _ _
 _ _ _ _ _ _ _ _ _ _
 _ _ _ _ _ _ _ _ _ _
 _ _ _ _ _ _ _ _ _ _
 _ _ _ _ _ _ _ _ _ _
 _ _ _ _ _ _ _ _ _ _
 _ _ _ _ _ _ _ _ _ _
```

The results are:

- **Random** - Actions: 3 , Time: 0.018 sec.
- **Backtracking** - Actions: 21 , Time: 0.001 sec.
- **Heuristic** - Actions: 81 , Time: 0.010 sec.

In this map, touchdown position is near start point, so **Random** can try a lot variants and finds the best ones(*So, one of the best is taken on this try)*, but in terms of time, it is always worse, than **Backtracking** and **Heuristic** methods. However, the correlation between **Backtracking** and **Heuristic** is different here. As touchdown is near to the start, **Backtracking** has done better as for amount of actions, as for execution time.

3.2.6 Map 5

```
Hey! Here the map:
[
  S _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _
  _ _ _ _ _ _ _ _ _ _
  _ _ _ 0 0 0 _ _ _ _
  _ _ _ 0 T 0 _ _ _ _
  _ _ _ 0 _ 0 _ _ _ _
  _ _ _ 0 _ _ _ _ _ _
  _ _ _ _ 0 _ _ _ _ _
[ _ _ _ _ _ _ _ _ _ _
```

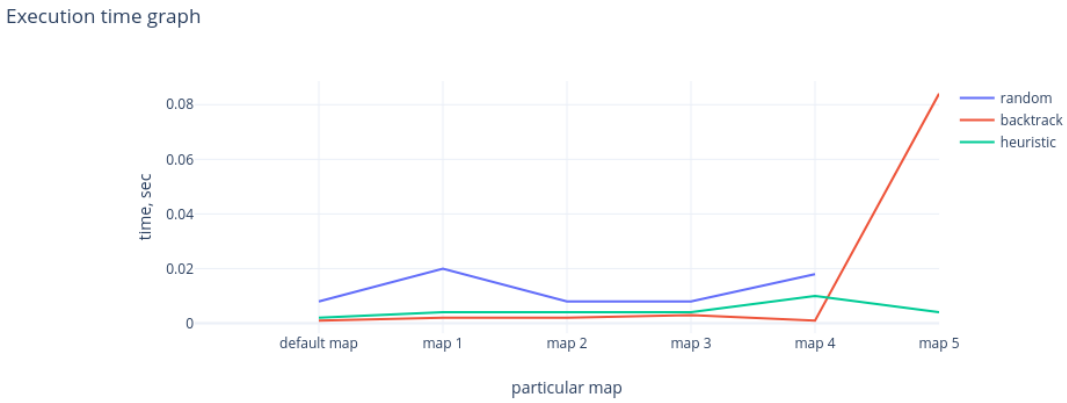
The results are:

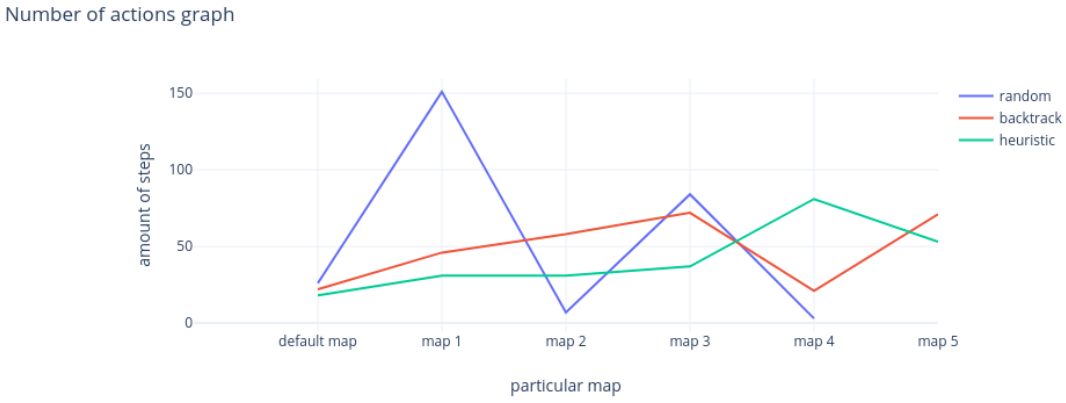
- **Random** - All 100 attempts have failed
- **Backtracking** - Actions: 71 , Time: 0.084 sec.
- **Heuristic** - Actions: 53 , Time: 0.004 sec.

The map is sparse enough, there is no chances to pass successfully, so all 100 attempts of **Random** search have failed. Results of **Backtracking** and **Heuristic** methods are different as well. Because of empty holes on the map, **Heuristic** has done better, numbers of actions are comparable, but in terms of execution time, **Heuristic** method is much better for this try.

3.3 Visualisation of the obtained information

3.3.1 Graphs for developed maps

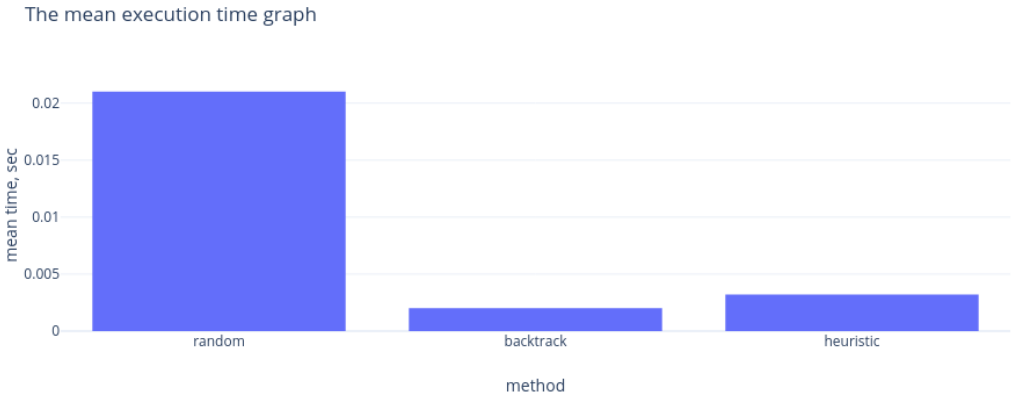


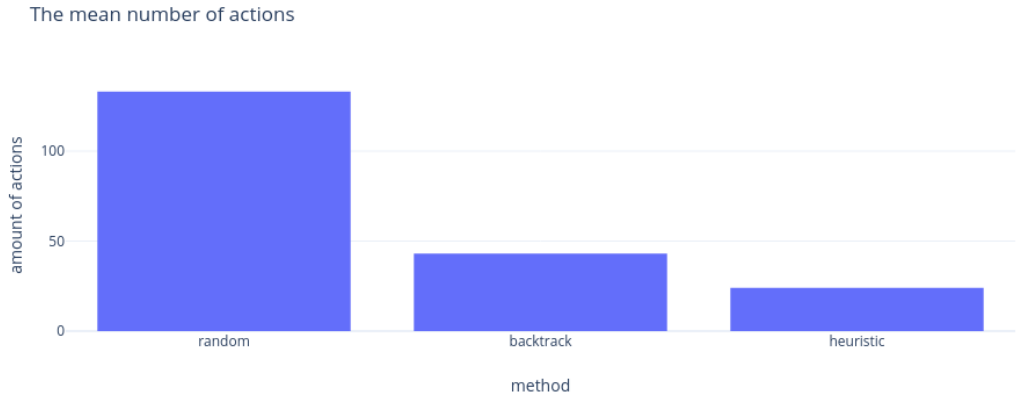


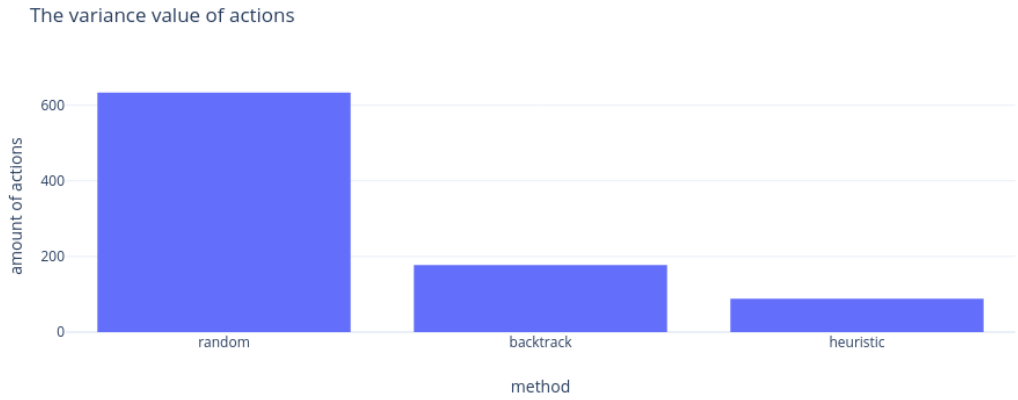
There are 2 graphs with execution time and number of actions respectively, for each of described map for every single method.
As all random method 100 attempts have failed on the 5 map, the graph is only till the 4 map.
The results are analyzed for the particular map above, at **3.2.** Now, let's see the average and variance values for every method.

3.3.2 Mean and Variance graphs

For calculating average statistics, I've tested methods on 30 different *(sparse and dense, small-size and big-size,etc)* maps. Also, I need to notice that, the **random** method have completely failed for 11 times out of 30. So, we have only **64** percent of successful attempts.
After that, I've calculated the mean values and variance values. They are represented as bar-diagrams below.







3.4 Conclusion

From analyzing results we can come up with the following ideas:

- In average, **Backtracking** method is the fastest, **Heuristic** method is best in terms of amount of actions and close to **Backtracking** in terms of execution time.
- **Random** method is the suitable for sparse maps where touchdown is quite close to starting point or it is suitable for tiny maps, not 10x10.
- **Random** method is also good for maps where big popularity of teammates(humans) and less obstacles(orcs). It gains big impact and gets less number of actions.
- **Heuristic** method works well for dense maps, but when touchdown is near to the starting point, then **Backtracking** works better.
- **Random** method values are the most far away from the mean value, it means that the values of the different attempts can be very contrasting to each other. However, **Heuristic** and **Backtracking** values are not so far and it means that methods have some intervals which majority of results belong to.

4 Part 2. 2-cells vision

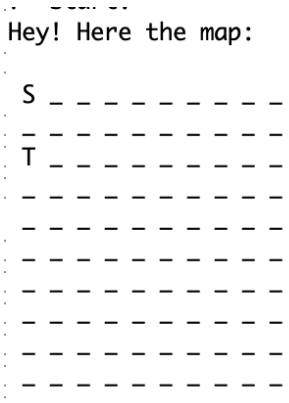
There is an explanation how vision not for 1 cell long, but for 2 cells can change performance of each method.

4.1 Impact on Random method

I consider that **there is no impact** on **Random** method at all. As all actions are chosen randomly, we are not looking to adjacent cells even at the current option with 1 cell vision. So, nothing will change and performance*(number of actions and execution time)* **remains the same**.

4.2 Impact on Backtracking and Heuristic function methods

I state that with 2-cells vision, **the problem can be solved using the methods in a more efficient way.** To explain it properly, I provide example: how it is solved with the current option and how it could be solved with the ability to determine the condition of the cell that is 2 units away from the cell where the player is standing.
Let's consider the obvious example: My current implementation of **Backtracking** and



Heuristic methods proved the following results:

- **Backtracking** - Actions: 20 , Time: 0.001 sec.
- **Heuristic** - Actions: 74 , Time: 0.010 sec.

But with 2-cell vision it can be solved with **2** actions! There significant changes of algorithms are needed for always looking for touchdown point, that can be very close. As we once found it in 2-cells long, there should be proceeded additional algorithm with another logic such that: it keeps the found touchdown coordinate and tries variants that possibly can direct to the touchdown. (kind of warmer-colder game is presenting when we've determined touchdown in 2 cells).
All in all, this option is quite powerful and is able to improve **Backtracking** and **Heuristic** methods significantly.

5 Part 3. Hard to solve and impossible maps

5.1 Hard to solve map

Hard to solve map means the map that takes more actions to be done for reaching the touchdown point. So, here we are with such one:

```
Hey! Here the map:

S _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _
_ _ _ _ _ T _ _ _ _
```

Looks pretty straightforward and easy, isn't? Nevertheless, agent has a lot of options(there is no obstacles, etc) for action, since it takes quite big number of actions for each method:

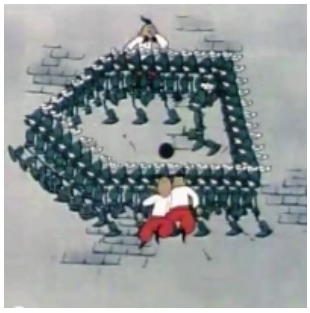
- **Random** - All 100 attempts have failed
- **Backtracking** - Actions: 94 , Time: 0.004 sec.
- **Heuristic** - Actions: 50 , Time: 0.005 sec.

5.2 Impossible map

With impossible map, we are **not** able to reach touchdown by some reasons.
So, here we go: Player will **never** reach the touchdown point.

```
Hey! Here the map:

S _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _
_ _ _ 0 0 0 _ _ _ _
_ _ _ 0 T 0 _ _ _ _
_ _ _ 0 0 0 _ _ _ _
_ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _
```



By the way, this strategy was taken from Soviet Union's cartoon called "Kazaki is playing football"

6 Summary

According to the results of the tests on 3 implemented methods, I have come up with the following ideas:

- As it was expected, **Backtracking** and **Heuristic** methods are good enough in terms of performance. The comparison between them is depends on the subjective ideas of implementation. How exactly they were implemented and which concepts were proposed.
  In my case, **Backtracking** mostly was a little bit faster in execution time, but **Heuristic** method was better in amount of steps for reaching the touchdown.

- A pass action is definitely not a good option for **Random** method, since for sparse maps, for big enough maps, for maps where are not many teammates - it fails. Fails very frequently. Because of this fact, it becomes untestable in many cases.

- if I needed to decide with which method to survive on a desert island, I would choose **Heuristic** function method because of its reliability.