Introduction to AI (IAI)

Assignment 1. Innopolis University, Spring 2020

Name: Hany Hamed
Group Number: BS18-06
Professor: Joseph Alexander Brown
TA: Nikita Lozhnikov

Contents

IAI, Innopolis University, Spring 2020
Author: Hany Hamed

Part0: Agent Description

This section includes a description of the Agent in PEAP form and the assumptions and the terminologies that have been used.

The term episode and round is used interchangeably in that document.

Episode means the sequence of actions that the agent takes in order to get fail or success in aspect of the achieving the goal.

The agent is the program/algorithm that gives the solution.

The term configuration here is used as the initial positions of the humans and orcs and touchdown (snapshot for the objects in the environment initially).

All the units in the figures are in msec (Y-axis).

PEAP

- Performance Measure:

  The cost that the agent takes in order to achieve the goal. For each action, we have its own cost.

- Environment:

  The environment is originally a playground which is in our program is a grid/map 10x10.

  It has several objects in that map:

  - Orcs ['o']
  - Humans ['h']
  - Walls/Boarders (Extra rows and columns for the map)[̂'w' (not explicitly given from the input but generated according to the map size parameter in the code)]
  - Place of the touch down (End point) ['t']
  - Place of the start point ['s']

- Actions:

  - Move (Up, Down, Right, and Left), it is available if there is no human, no orc, no wall, or not a visited cell in the target cell.
  - Hand-off (All the Directions) (Move to the cell and give the ball to the human inside it) It is available if there is a human inside the target cell and the cell is in the neighbourhood the current cell. The first move is not considered in the cost calculation neither the giving the ball step but the next movement from the human with the ball is considered in the cost calculation.

  - Long Pass (All the directions). This is only applicable once in the whole episode. It is available if the direction has a human and not intercepted by an orc.

- Sensors:

  The agent can see only n step ahead according to the parameter that is set in the code. In the code it is put with 1. Step ahead means that can see the cell ahead in the specific direction.

Notes:

- The map is assumed to be always a square that has n rows and n columns
- In Random Search Algorithm, the number of the maximum steps is a parameter and it is by default is set to 50. The counter of calculating the steps is considering all the actions not only the movements in the counter, which means that the agent will do 50 step regardless it is a move, hand-off or long pass.
- In Random Search Algorithm, the number of the episodes that is supposed to run is specified in the parameters and it is set by default to 10. Which means that the algorithm will try number of episodes and get statistics that how many of them failed and how many of them succeed to achieve the goal.
- In printing the path in the backtracking, the path is printed inversely. from the end point to the start point to show the concept of the recursive calls. Also, in the printing, the cells if it is visited, it will be presented in the path twice as in and out from that cell.
- The source codes structure consists of 5 files:
  - main.pl: it has the main algorithms for the tasks.
  - utils.pl: it has some general auxiliary rules that has been written to support the main.pl file (e.g. printing a list in prolog, get the length of the list, get the number of occurrences for a specific pattern in a list)
  - counters.pl: it has rules and definitions of counters, these has been created by a python script that generate this code according to the name of the counter.
  - input.pl: input file to the code.
  - ouput.txt: output file that is generated from the code.
- Running and Testing the code: It has been tested by swipl
  - Run Backtracking Search algorithm:
    "swipl -s main.pl -t mainBacktrack."

  - Run Random Search algorithm:
    "swipl -s main.pl -t mainRandomSearch."
  - Run A* Search algorithm:
    "swipl -s main.pl -t mainAStarSearch."

- The parameters are defined in the beginning of the main.pl file
  - "mapSize"
  - "maxStepsRS" is for the maximum number of steps in the episode in Random Search
  - "numEpisodesRs" defines the number of episodes that run by the Random Search
  - "numStepsAhead" is related to how many cells that the agent can see in advance, it is more related to Heuristic algorithm (A*) as it is not useful at all for Backtracking neither the Random Search.
- To make an action, we ran a validation on it, to ensure if it is possible or not and get a fact (flag). Flag value is 0 if the action is valid, 1 if the cell is Border, 2 if the cell has orc, 3 if it is visited before, 4 if it is not valid pass (Valid pass: is possible pass when this is the first pass and the only one)
- Hand-off can be done multiple times not limited to be once in round, however, the long pass is limited to be once in the episode/round.
- In A* with one step ahead, only moves and hand-offs are available, long pass is not allowed as the agent does not see the next cells as the step ahead is default by one, if the step ahead increased the long pass can be used and can decrease the cost and the its heuristic cost will be the Manhattan distance between the place that the ball is going to be passed to and the goal and its value will be with negative and be normalized to be in the range [-1,0] as the goal heuristic cost is -1 (it is discussed later in part 1.3).

Part1.1: Backtracking Search Algorithm

- **Algorithm Description**

  The algorithm is mainly about exploring the whole search tree in a recursively DFS way as if it is reached the leaf of the solution (no other action to be made, is recursively return to the previous action and check another action instead).

  In the source code, the backtracking is implemented with adding a small optimization for pruning the non-optimal paths that exceeds the current optimal path length which is initialized with the n^2 where n is the map size as the upper-bound for the path length as the path is impossible to exceeds the number of the cells in the map as it is not allowed to visit a cell that has been visited before. When the path length exceeds this optimal, it is pruning as we already have a better optimal solution, in this way, we will be able to get the most optimal solution.

Also, there is another optimization that is known but it is not implemented here, which is incrementally deepening for Backtracking, which starts with 1 as maximum depth for the searching tree and whenever a solution is found, it is known that it is the optimal as any other solution will be found will be either equal or longer which in that case will not be optimal.

Also, if there is multiple optimal paths, they are being saved and being output.

In the implemented algorithm, the order of trying the actions is as following: try moving up, down, right, left, and then passing by the all directions in order. Sometimes, this order can find a configuration, that will make it fast and constant performance with different map sizes with the same configuration

- **Reflection**

  It can explore all the possible solutions by bruteforce all the solutions in the search tree and recursively back to the error and correct it. This made it very slow in big sized maps.

Part1.2: Random Search Algorithm

- **Algorithm Description**

  This algorithm explore the space of paths (Solutions and not Solutions) by choosing randomly an action in each step. In the end the sequence of the actions that are taken are not necessarily to be a solution.

  It is totally a stochastic method that is not a good way for finding a solution, however, it can be considered a way to generate a solution for a problem (Monte-Carlo Methods) to be used further to improve that solution from not optimal to an optimal solution by optimization methods (Hill-Climber, Simulated Annealing, Augmented Random Search in Reinforcement Learning [Source1, Source2]).

  The code is running number of episodes and each episode is maximized by specific number of steps.

  Choosing the action is based on having a random variable from a uniform distribution and the actions are having also equal probability to be made. The random variable is between 0 and 1 and this range is divided by 12 (number of possible actions) to have the step size between the ranges that the actions are allowed to be done.

  Also, the cells can be visited multiple times.

- **Reflection**

  The results is totally stochastic and cannot be determined, but it can provide fast way to get a solution in small scale problems, but with large search space it might be impossible to find a solution.

Part1.3: Heuristic Search Algorithms

- **Algorithm Description**

  The algorithm that has been implemented is A*.

  A* requires a prior-knowledge from the environment to have its heuristic cost (e.g. the Manhattan distance between the current and the goal cells) and it is fast algorithm and it gives the optimal solution as long as the heuristic cost/function is lower bound for the solution, if it overestimate the cost, it will give a local optimal solution, if it is lower bound, it will give an optimal solution and will be faster if the heuristic cost is closer to the real estimated cost for the solution to solve the problem.

  However, in our case we do not know the place of the goal, that's why the heuristic is not useful in that case, and it can be set to zero and the cost of the step is based on the cell that the human is moved to, if it is possible move it costs 2, if it is possible hand-off it costs 1, if it is long pass it costs 1 and all these costs is summed up with the value of the parent cell to that cell. The value of each cell is being calculated incrementally according to the number of the observable cells (number of steps ahead).

  In another analogy for the cells, to think about it as a graph and the transition (Edges) between the cells (Nodes) only costs 1 and the rest of the cost is the heuristic value which it differs according to what inside the cell if it is human as hand-off pass the cost is just 0 and if orc or wall it is excluded from the open list as having really big cost, if it is empty the cost will be 1 and if it the touch down the cost will be -2 to make the value of cell is less and explore as soon as possible and one of the optimization that is used to finish faster is if the touchdown is in the open list, explore it directly, if it is in the closed list, finish the search and the agent have reached to the touchdown with the optimal path and then just recursively iterate on the parents on the visited cells starting from the touchdown.

  Also, If we know the place of the touch down, the heuristic will be the Manhattan distance.

  Some notes about Hill-Climber Algorithm, this algorithm requires in the beginning to get a solution that is not an optimal and try to optimize in the set of successful solutions to get an optimal solution either locally or globally, however, this algorithm is very hard and power consuming to be implemented in that problem as it is high dimensional problem that the search space for optimization will be A^{num_steps} where the number of steps is the number of actions that is taken to reach the goal and this number is needed to be improved, and A is the set of possible actions, the number of steps in the initial path could be huge.

- **Reflection**

  It gets the optimal solution and faster than the backtrack. If the touchdown place is known, it will be faster by defining better heuristic costs relative with the distance left to reach the touchdown.

Part1.4: Further Algorithms To Be Implemented

- Q-Table

  This algorithm is one of the Reinforcement Learning Algorithms that is very basic and based on Bellman optimality equations, I have used it before in simple problems and it can be applied here in that problem but it will be much easier not to be implemented in Prolog, I had the intention to implement it in Prolog but did not have time for it.

- Evolutionary Hill-Climber

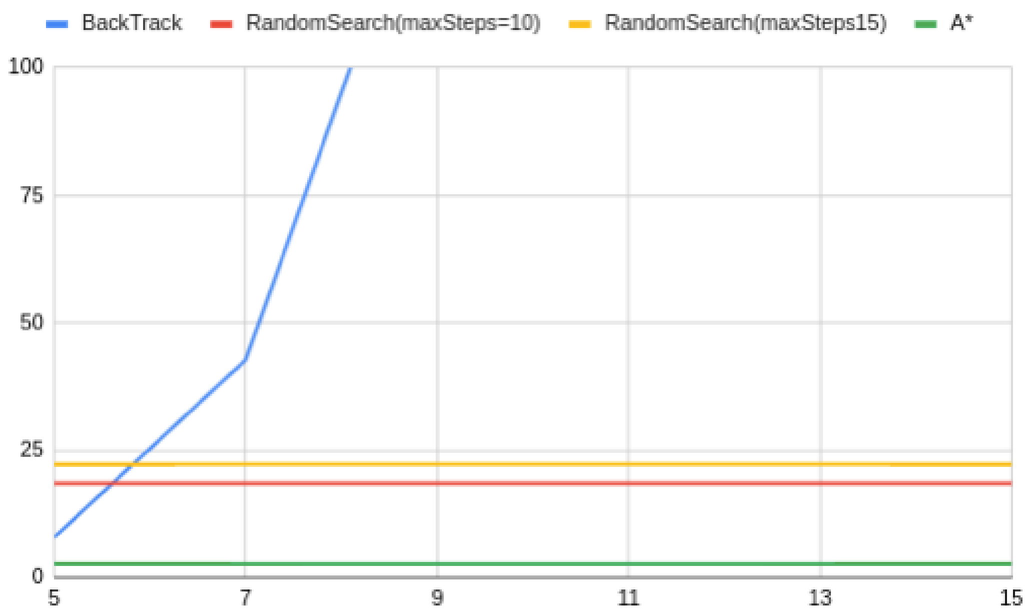
  This algorithm is based on Hill-Climber optimization algorithm with an evolutionary optimization that make generations of optimization parameters for a neural network that output the correct action according to the state of the world and select the best from them. This algorithm, I got known about it from Behavioural & Cognitive Robotics by Prof. Stefano Nolfi

- Multi Agent systems

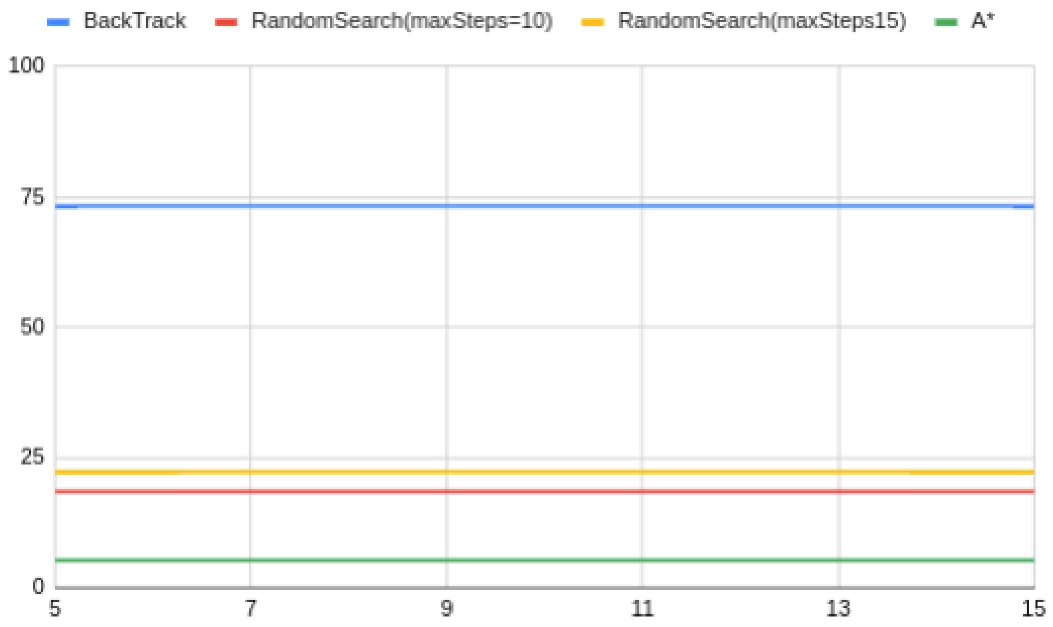
  The environment can be discussed further in the point of view of multiple agents that share the knowledge that they had with each other.

Part2: Agent Modifications & Results

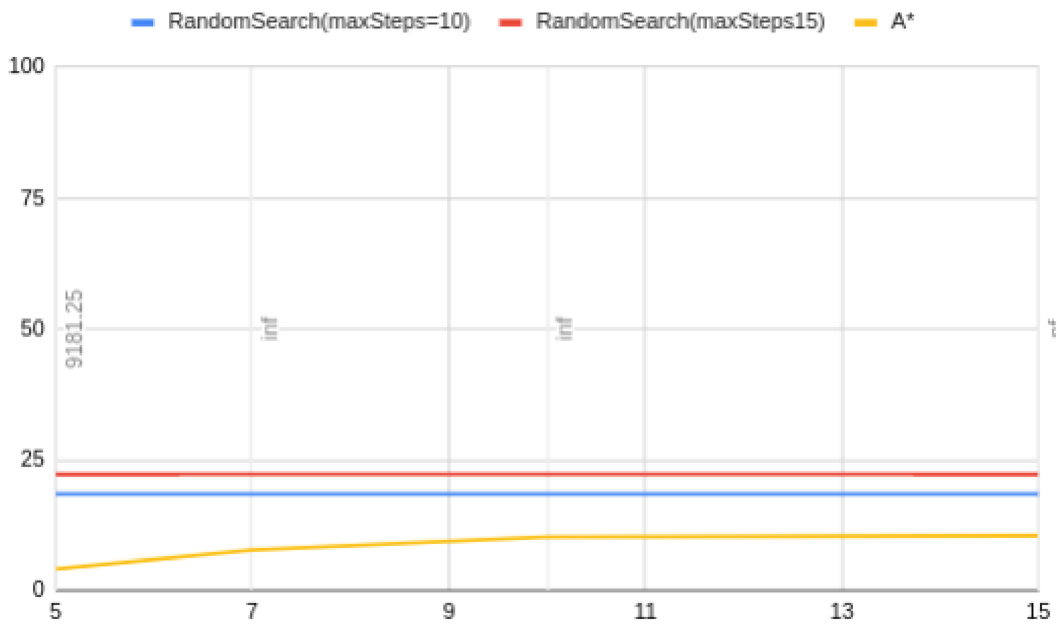
- If map size increased, Random Search will most probably will not get a solution and Backtrack search will take much time to get the optimal answer, but the A* algorithm will be faster and more suitable in that case.
- If the number of steps ahead changed, it will affect on A* algorithm to reach the touchdown faster as it will put the touch down earlier than in case it is only one step ahead.
- Variation of the number of steps ahead is implemented especially for the A* algorithm by a parameter in the beginning of the file to provide calculating of the cells ahead and converge faster to the solution by exploring the touchdown earlier and it will be more useful with better heuristic if the agent has knowledge about the place of the touchdown.



(Figure: 1) Comparison of changing the map size for map(1) (inputs/input_easy1.p)



(Figure: 2) Comparison of changing the map size for map(2) (inputs/input_easy2.pl)



(Figure: 3) Comparison of changing the map size for map(2) (inputs/input_hard.pl)

In all the figures, the random search is always constant as it is only variable by the maximum number of the steps per episode, and the number of the episodes per iteration, however, the random search as it was mentioned before it is totally stochastic and it can lose from the first step by stepping to an orc cell.

Moreover, A* performance does not change by the map size but it change by the configuration of the map, and still it is the fastest between them all.

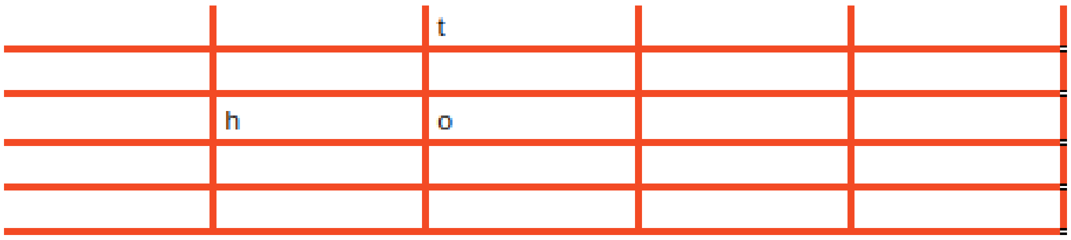
In Figure 2, Backtrack is constant as the configuration of the map and the order of trying the movement conform to find the solution from the leftmost deep level of the search tree as it is actually not doing backtracking as it finds the solution directly, thus the changing of the map size does not affect the performance.

In Figure 3, the performance of it, was really bad it, it started with 9000 msec and with larger maps it was not terminating in 2 minutes.

Part3: Easy & Hard Map Configurations/Arrangements

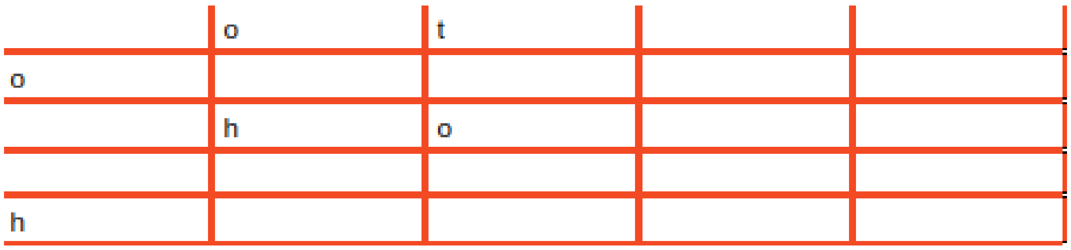
I have added the tested configurations in folder called ”inputs”

- Easy configuration of the playground, one orc and touchdown is relatively close to the human. (inputs/input_easy1.pl)



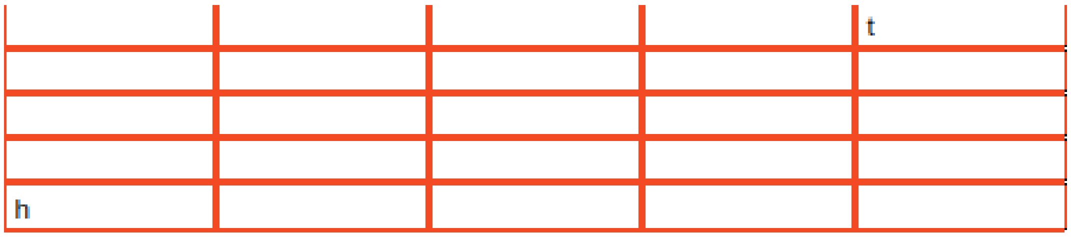
(Figure: 4) Description of part of (inputs/input_easy1.pl) map

- The same as last one but an extra human and more orcs, to test the algorithms are working with different cases (inputs/input_easy2.pl)



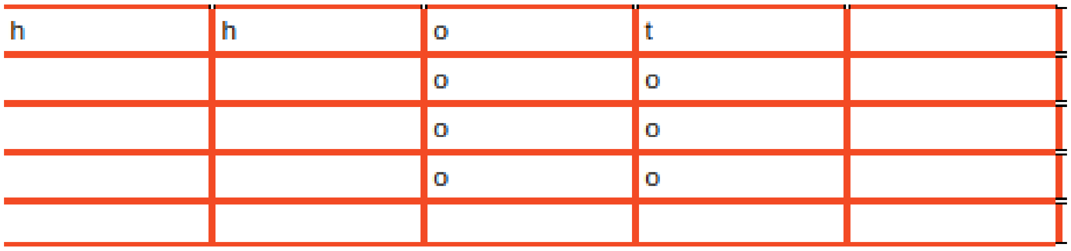
(Figure: 5) Description of part of (inputs/input_easy2.pl) map

- Only one human in the staring point (1,1) and the touch down is on the end corner of the playground (n,n), A* will work as BFS search algorithm in that case. (inputs/input_med.pl)



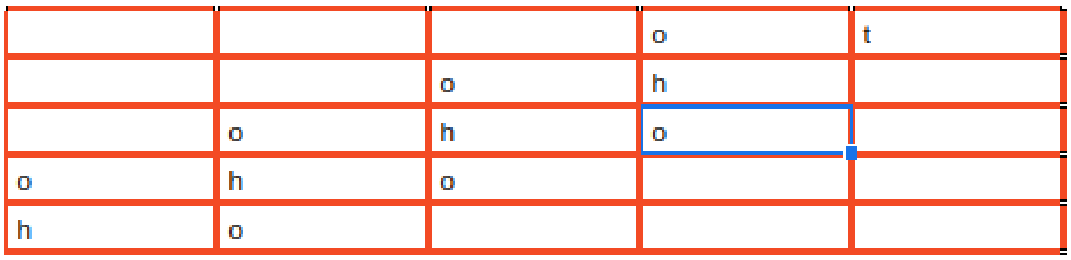
(Figure: 6) Description of part of (inputs/input_med.pl) map

- One tunnel is available for humans from their own zone to the touch down as the orcs in the most of the map. (inputs/input_hard.pl)



(Figure: 7) Description of part of (inputs/input_hard.pl) map

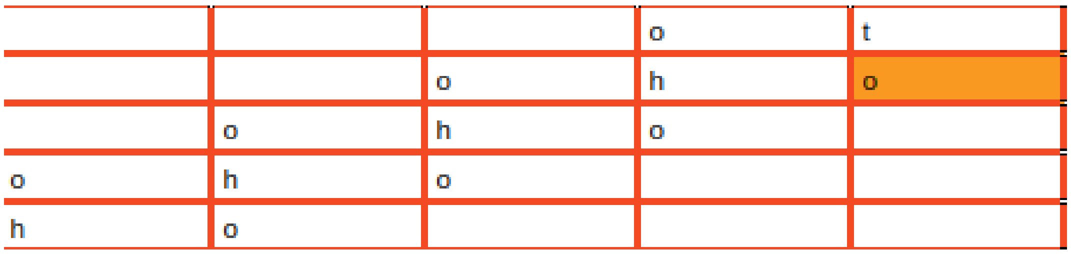
- Map full of humans in the diagonal in which the ball is allowed to be handed-off in any number of times and orcs around them except down the touchdown to be able to move and go to the touchdown. (inputs/input_special1.pl)



			o	t
		o	h	
	o	h	o	
o	h	o		
h	o			

(Figure: 8) Description of part of (inputs/input_special1.pl) map

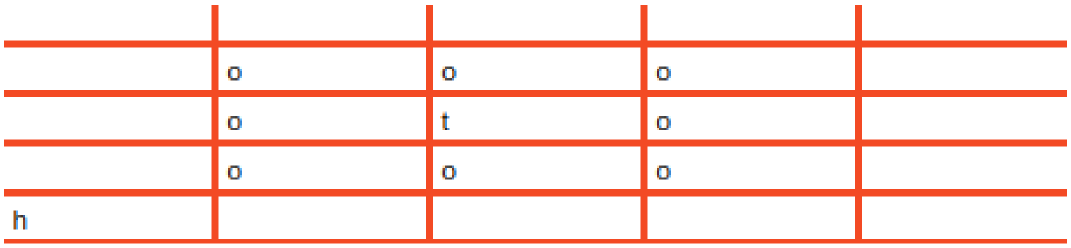
- A modification to (inputs/input_special1.pl) to have an orc that blocks the only way to the touchdown which make it impossible to reach the touchdown. (inputs/input_special2.pl)



			o	t
		o	h	o
	o	h	o	
o	h	o		
h	o			

(Figure: 9) Description of part of (inputs/input_special2.pl) map

- One of the hard configurations that it is nearly impossible to be solved that the touchdown is surrounded with orcs from all the possible directions. (inputs/input_impossible.pl)



	o	o	o	
	o	t	o	
	o	o	o	
h				

(Figure: 10) Description of part of (inputs/input_impossible.pl) map

+Acknowledgments & References

- Khaled Ismael: For discussing about the different algorithms and the some optimization techniques for A* for example BFS-zero-one and discussing

about the heuristic cost and some discussion about the optimality of finding a solution for A*. Discussing about the idea of sharing the knowledge between the human players to create a Multi Agent system where the agent is related to the humen.

References

(Stackoverflow, Prolog Documentation, online Presentations, ...et) It is cited within the codes.