# Introduction to Artificial Intelligence

## Assignment 1. Humans vs. Orcs Rugby

**Report**

**Khastiev Shamil, BS18-05**

March 2020

# Contents

# 1. Part 1

## 1.1.Overview

I have implemented 3 algorithms(Random search, Backtracking, and Heuristic search) for solving the Humans v. Orcs Rugby problem. Implementations placed in three files:

- random.pl
- backtracking.pl
- heuristic.pl.

**To run the program** execute the following commands:
swipl algorithm.pl (instead ‘algorithm’ put random or backtracking or heuristic )
letsplay(input.pl). (instead ‘input.pl’ you can put any input file in .pl format)

**Input** should be in the following format(instead off X, Y put coordinates):

- h(X,Y). Human coordinate
- o(X, Y). Org coordinate
- t(X, Y). Touchdown coordinate

Sample input from one of my tests already in input.pl file and some tests are in tests.txt

## 1.2. Brief description of 3 implementations

Common aspects

Each algorithm has the main function (letsplay). Calling this function you can start executing the whole solution. Also, there are 8 functions responsible for moving and passing.

1. Random search

In this algorithm, movement is randomly chosen. If movement is illegal, then the program chooses another direction to move. After doing a moving program prints the current position. If the program will attempt more than 100 moves to find a touchdown it will return false and inform about the reason failing.

2. Backtracking

The main part of this implementation is dfs function that recursively calls a moving function and if the ball reaches a dead-end algorithm will go back and try another movement. After finding a touchdown program still goes back and trying to find a shorter solution. So, after finishing execution it prints the shortest path to a touchdown or false, which means there are no solutions for this much.

3. Heuristic search

In this implementation first function tries to visit all neighbors of the current cell, then the algorithm repeats the same function for the next cell until finds the solution.
There are some constraints for choosing adjacent cell:

- cell can be reached via possible moves;
- a new cell is in map boundaries
- cell was not visited before in the current path

## 1.3. Comparison between implementations

For testing and comparing search algorithms, I used 30 random maps, on every map I ran each algorithm 10 times(for random search sometimes more than 10, till it find a path). This 30 maps contains maps with different properties:

- The population of the map. A number of orcs and humans.
- Size of the map
- The distance between the initial point and touchdown

Discussion wich maps are hard to solve for each algorithm in section 3.1
After collecting data from tests I have got next average results:

| Algorithm | Average time (sec) | Mean num of moves | Fail percentage |
|---|---|---|---|
| Random | 0.001 | 75 | 84% |
| Backtracking | 0,8 | 13 | 0% |
| Heuristic | 0.3 | 13 | 0% |

Backtracking and heuristic search have the same number of moves because they are searching for the shortest path to the touchdown. Also in my tests were only maps that have at least one solution that is why for backtracking and heuristic search algorithms fail percentage is 0.

## 1.4. Conclusion

From tests and analyses of results of the test, I have found some points:

- Random search is the fastest algorithm but has a low probability of success
- Random search is good for big maps with less number of orcs and humans
- Backtracking and heuristic search are good for finding the shortest path

- Backtracking better heuristic search for cases then touchdown locates faraway from an initial point.

## 2. Part 2

In this part, we will look haw switching from one-cell vision to two-cell vision will influence to performance of searching algorithms.

### 2.1 Two-cells vision random search

I did not find any effect on any of the performance indicators of the algorithm. The number of cells that we see in the random search algorithm has no effect because the movements are randomly selected. The randomness leads to the fact that even being in the vicinity of the endpoint, the player can go in the opposite direction and the algorithm finishes execution only after it gets to the touchdown.

### 2.2 Two-cells vision backtracking and heuristic search

Two-cells vision in backtracking and heuristic search will give some performance gain. To substantiate my point of view, I will give examples of maps and my statistics collected from experiments.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| * | * | * | * | * | * | * | * |
| * | | | | | | | * |
| * | | | | | | | * |
| * | | | | | | | * |
| * | | T | | | | | |
| *” | “ | | | | | | |
| *” | “ | “ | “ | | | | |
| i* ” | “ | “ | “ | | | | |

For example in the case then touchdown locates in 3rd-row backtrack algorithm will start searching with way shown by ‘*’ and heuristic search will go by way shown by ‘ “ ’. In both cases having two-cells vision, touchdown can be found much faster. For finding the shortest path with one-cells in following example vision backtracking and heuristic search need 0.009 and 0.005 seconds correspondently. But with two-cells visions, they need only 0.001 seconds both.

## 3. Part 3

In this section, we will look at various maps that can cause difficulties for searching algorithms.

### 3.1. Some interesting(hard to solve for some algorithms) maps

In below maps, the following notations are used:

- H - human
- o - org
- T- touchdown
- i - initial point
- . - shortest path

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | | | | | | | |
| | | H | | | | | |
| | | | | H | | | |
| | o | o | o | o | | | |
| | o | T | . | . | . | .H | |
| | o | o | o | o | | | |
| i. | . | . | . | . | | | |

For solving this map backtracking method needs 4 seconds and heuristic search needs only 0.005 seconds, because of the order in which the backtracking search tries moving directions. By the way, a random search on this map showed an average result and found a solution for the 6th time.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **H** | | | | | | | |
| | | | | | | **T** | |
| | **O** | | | | | | |
| | | | | | | | |
| | **O** | | | | | | |
| | | | **O** | | | **O** | |
| | **H** | | | | | | |
| **i** | | | | | | | |

For solving this map heuristic search needs near 3,5 seconds, because it first checks ways near the current position and does not go the maze depth. To solve this map backtracking algorithm needs 0.061 seconds and random search 0.002 seconds.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | **O** | | | | **O** | **O** | **O** |
| | | **O** | | **O** | **O** | **T** | |
| | | | **O** | **O** | | **O** | |
| | | | | **O** | **O** | | |
| | **O** | **O** | **O** | | | **O** | |
| | | | | | | | |
| **i** | | | | | | | |

To solve this map random search algorithm needed 21 attempts, this happened because this map has a lot of ‘bottlenecks’ and if the ball goes to the ‘bottle’ without a touchdown it’s really hard to go to another ‘bottle’.

## 3.2. Impossible maps

There are a lot of impossible maps, but all of them have a common thing, that touchdown and initial point are in a different parts of a map separated by orcs between each other.
For example:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | **T** | | | | | |
| | | | | | | | |
| **o** | **o** | | | | | | |
| **i** | **o** | | | | | | |