AI assignment 1 – Humans vs Orcs Rugby. Report

Rufina Talalaeva, BS18-01

Input:
Input is defined inside of all my programs by hands.

1. World size:
   The size of world is given by predicate *world(x,y).* , where x=y, x ∈ [0, 19] by task.
2. Agent coordinates:
   The coordinates of agent is given by predicate *agent(x,y).* , where x=y=0 by task.
3. Touchdown coordinates:
   Coordinates of touchdowns are given by predicate *listOfTouchdowns([]).*
   Example of touchdown in x = 3, y = 2: *listOfTouchdowns([[3,2]]).* Example of several touchdowns: *listOfTouchdowns([[x_1,y_1], [x_2, y_2], [x_3, y_3]]).* where where x_i,y_i ∈ [0, world_size].
4. Humans coordinates:
   Coordinates of humans are given by predicate *listOfHumans([]).*
   Example of one human in x = 3, y = 2: *listOfHumans([[3,2]]).* Example of several humans: *listOfHumans([[x_1,y_1], [x_2, y_2], [x_3, y_3]]).* where where x_i,y_i ∈ [0, world_size].
5. Orcs coordinates:
   Coordinates of orcs are given by predicate *listOfOrcs([]).*
   Example of one orc in x = 3, y = 2: *listOfOrcs([[3,2]]).* Example of several orcs: *listOfOrcs([[x_1,y_1], [x_2, y_2], [x_3, y_3]]).* where where x_i,y_i ∈ [0, world_size].

How to run the code:
Open the site. Paste the code of the program. Write to console of queries *start.*

Measurement of effectiveness of algorithms:
Effectiveness of algorithm was measured by execution time of the algorithm.
All measurements were done on the site. Consequently, there could be inaccuracy in measurements according to different congestion of the site.

Part 1.
3 algorithms were implemented:

1. Random search
2. Backtracking
3. Improved backtracking

1. Random Search:
This algorithm makes 100 attempts to reach the touchdown using random actions.
Actions = move up/ move down/ move right/ move left/ pass ball in diagonal/ pass ball up/pass ball down/ pass ball right/ pass ball left.
The agent has no eyesight in this algorithm, he/she does random actions in hope he/she will reach the touchdown. It is search by exploration. Although agent makes random actions, he/she does not visit already visited cells.

1 attempt = agent goes from his starting coordinates until reaching the touchdown/bumping into an orc/bumping into a wall/throwing a ball into a wall/throwing a ball to an orc/all cells around are visited already.

The source code of the algorithm could be found in file named *final_random.pl*

2. Backtracking:
This algorithm retrieves all possible paths from agent's starting coordinates to touchdown.
Actions = move up/ move down/ move right/ move left/ pass ball in diagonal/ pass ball up/pass ball down/ pass ball right/ pass ball left.
The agent has no eyesight in this algorithm, he/she does one of the possible actions, if it results in bumping into an orc/bumping into a wall/throwing a ball into a wall/throwing a ball to an orc/all cells around are visited already, it traces back until valid position.
At the end it prints one path with the minimum number of steps.

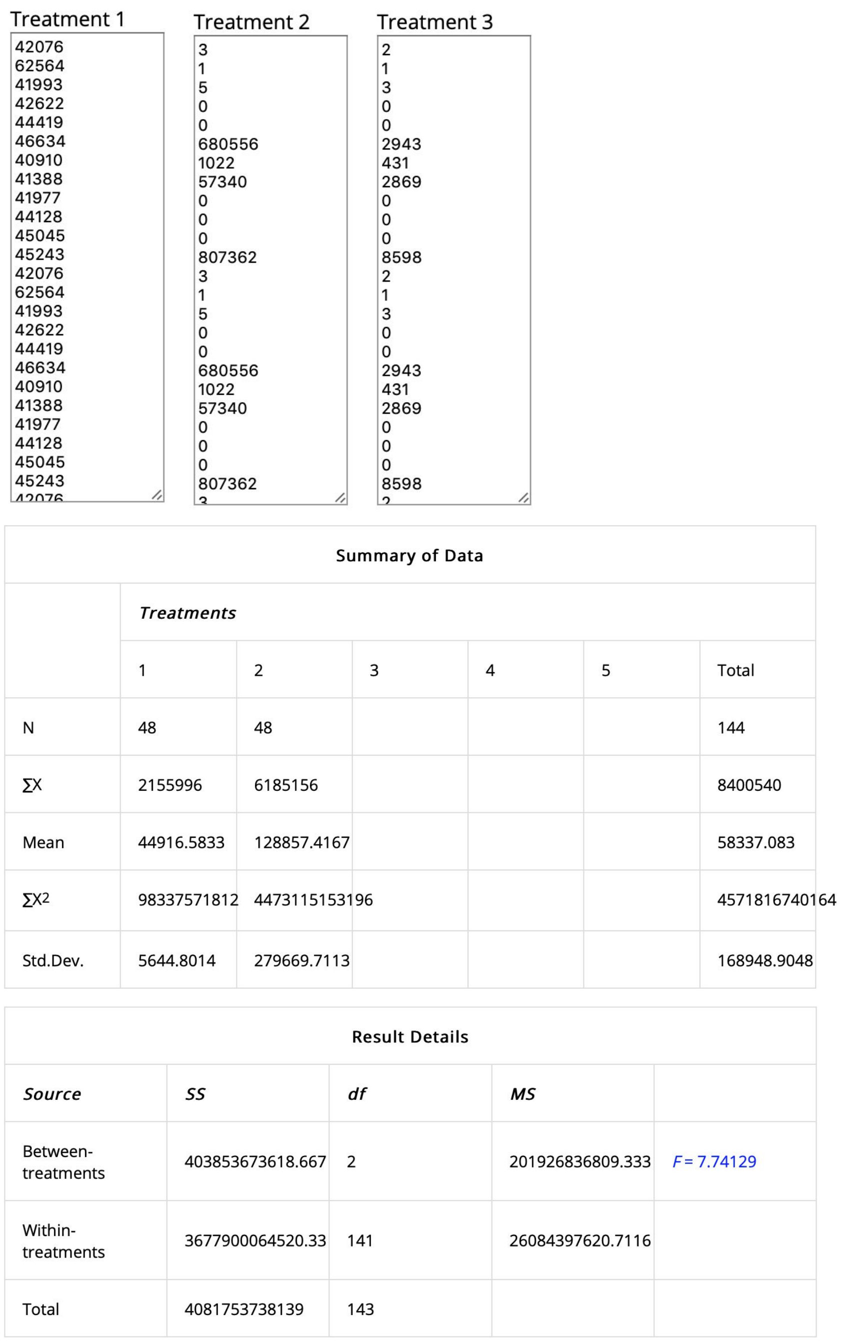
The source code of the algorithm could be found in file named *final_backtracking.pl*

3. Improved Backtracking:
This algorithm does absolutely the same actions as previous, but improves it by cutting all paths that are longer at least by one than current minimum path.

The source code of the algorithm could be found in file named *final_improved_backtracking.pl*

Tests
My H_0 is "The time of finding the optimal path does not depend on chosen method".
I have used ANOVA test for comparing the results and find out whether null hypothesis is true or not.
The idea of this test is in finding out how much of the total variance comes from:
- The variance between algorithms
- The variance within algorithms

We need calculate the F-ratio, the larger the ratio, the more likely we will reject our H_0 :
F = $\frac{between\ algorithms}{within\ algorithms}$; According to the table critical value F = 3.01.
$F(b, w),\ where\ b = 2,\ w = 69.\ F(2, 69) = 3.78829$
Our F is larger than critical value=> we reject our null hypothesis.
Conclusion: The time of finding the optimal path depends on chosen method.



Treatment 1	Treatment 2	Treatment 3
42076	3	2
62564	1	1
41993	5	3
42622	0	0
44419	0	0
46634	680556	2943
40910	1022	431
41388	57340	2869
41977	0	0
44128	0	0
45045	0	0
45243	807362	8598
42076	3	2
62564	1	1
41993	5	3
42622	0	0
44419	0	0
46634	680556	2943
40910	1022	431
41388	57340	2869
41977	0	0
44128	0	0
45045	0	0
45243	807362	8598

Summary of Data

	Treatments					
	1	2	3	4	5	Total
N	48	48				144
∑X	2155996	6185156				8400540
Mean	44916.5833	128857.4167				58337.083
$\sum X^2$	98337571812	4473115153196				4571816740164
Std.Dev.	5644.8014	279669.7113				168948.9048

Result Details

Source	*SS*	*df*	*MS*	
Between-treatments	403853673618.667	2	201926836809.333	*F* = 7.74129
Within-treatments	3677900064520.33	141	26084397620.7116	
Total	4081753738139	143		

The *f*-ratio value is 7.74129. The *p*-value is .000646. The result is significant at *p* < .05.

Part 2.

In my backtracking algorithms I didn't implement the feature to see 2 yards further, but there would be improvement in time of finding optimal path definitely.

We can think about other heuristics algorithms, such as greedy.

In greedy algorithm the feature of seeing in 2 yards can be a bad idea.

4	T				T
3	O	O			
2			O		
1			O		
0	A				
	0	1	2	3	4

Here is the example of existing the best path, but seeing in 2 yards for greedy will lead to fail. Agent will see that touchdown is in 4 steps and there is no orcs on the path, will definitely go there, but after going to (0,1) will fail, because there is no path to touchdown, only if you will move back from touchdown, which is impossible for greedy algorithm.

Part 3.

<u>Hard to solve:</u>

1)

4	T				
3	O		O		
2		O			
1				O	
0	A		O		
	0	1	2	3	4

For random search this map is really hard to solve, because by making random moves it has higher probability to bump into orc or wall in comparison to backtracking algorithm for example.

2)

5	T					
4						
3	H					
2			H			
1						H
0	A			H		
	0	1	2	3	4	5

For backtracking search this map is really hard to solve, because it needs a lot of time to compute all paths to conclude which path is the best .

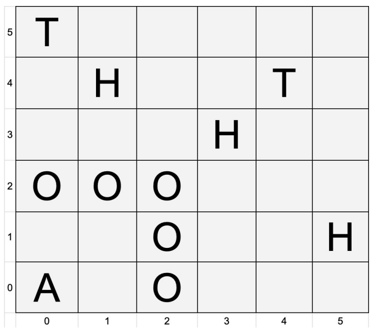
<u>Impossible to find the path:</u>

1)

5	T	O			O	
4	O	H		O	T	O
3					O	
2	H		H			
1						H
0	A		O			
	0	1	2	3	4	5

Impossible to find the path, because all of the touchdowns are blocked by orcs.

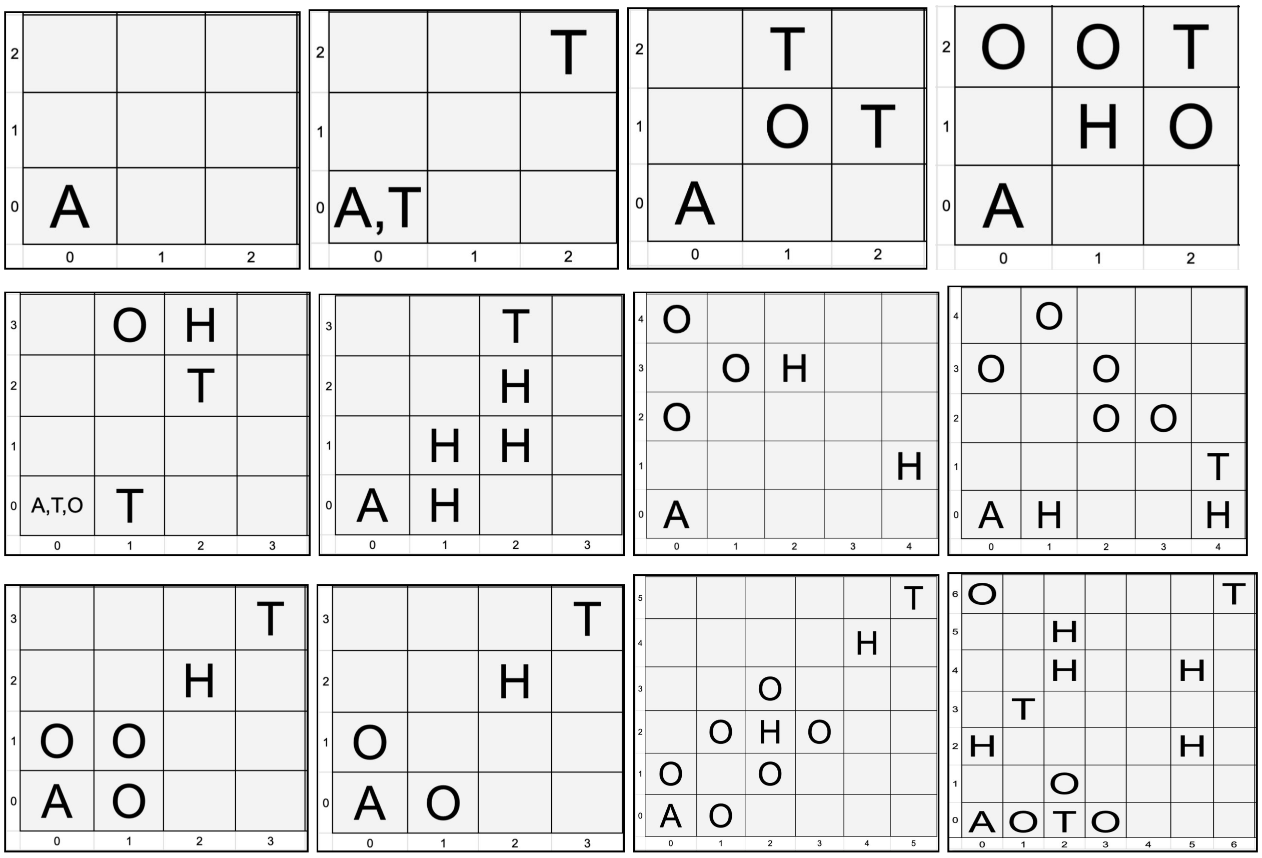
2)



Impossible to find the path, because all of the paths from agent are blocked by orcs.

Commonality: Impossible paths can be only when orcs block the target or agent, no way to pass or take the ball to reach the touchdown:(

Maps that were used for validness of the program and for ANOVA test in part 1:



```
--1
agent(0, 0).
listOfTouchdowns([]).
world(2,2).
listOfOrcs([]).
listOfHumans([]).
```

```
--2
agent(0, 0).
listOfTouchdowns([[0,0],[
2,2]]).
world(2,2).
listOfOrcs([]).
listOfHumans([]).
```

```
--3
agent(0, 0).
listOfTouchdowns([[1,2],[
2,1]]).
world(2,2).
listOfOrcs([[1,1]]).
listOfHumans([]).
```

```
--4
agent(0, 0).
listOfTouchdowns([[2,2]]).
world(2,2).
listOfOrcs([[0,2],[1,2],[2,1]
]).
listOfHumans([[1,1]]).
```

```
--5
agent(0, 0).
listOfTouchdowns([[0,0],[
1,0],[2,2]]).
world(3,3).
listOfOrcs([[0,0],[1,3]]).
listOfHumans([[2,3]]).
```

```
--6
agent(0, 0).
listOfTouchdowns([[2,3]]).
world(3,3).
listOfOrcs([]).
listOfHumans([[1,0],[1,1],[
2,1],[2,2]]).
```

```
--7
agent(0, 0).
listOfTouchdowns([]).
world(4,4).
listOfOrcs([[0,2],[0,4],[1,3]]).
listOfHumans([[4,1]]).
```

```
--8
agent(0, 0).
listOfTouchdowns([[4,1]]).
world(4,4).
listOfOrcs([[0,3],[1,4],[2,2],[2,3],[3,2]]).
listOfHumans([[1,0],[4,0]]).
```

```
--9
agent(0, 0).
listOfTouchdowns([[3,3]]).
world(3,3).
listOfOrcs([[0,1],[1,0],[1,1]]).
listOfHumans([[2,2]]).
```

```
--10
agent(0, 0).
listOfTouchdowns([[3,3]]).
world(3,3).
listOfOrcs([[0,1],[1,0]]).
listOfHumans([[2,2]]).
```

```
--11
agent(0, 0).
listOfTouchdowns([[5,5]]).
world(5,5).
listOfOrcs([[0,1],[1,0],[1,2],[2,1],[2,3],[3,2]])
.
listOfHumans([[2,2],[4,4]]).
```

```
--12
agent(0, 0).
listOfTouchdowns([[1,3],[2,0],[6,6]]).
world(6,6).
listOfOrcs([[0,6],[1,0],[2,1],[3,0]]).
listOfHumans([[0,2],[2,4],[2,5],[5,2],[5,4]]).
```

Some explanation about output:

Here letters identify the transition from previous state to the next:

P- pass from previous cell to next

M- move from previous cell to next

H-human from in next cell, identifies that we need not to count it as a step

```
Number of steps: 7
Path: [[0, 0], P, [0, 2], M, [0, 3], M, [0, 4], M, [1, 4], M, [1, 5], H, [2, 5], H, [2, 4], M, [2, 3], M, [1, 3]]
```