LSINF2275 - Project1

Battaglia Gabriele - 29181900 - MAP(Double degree)
De Winter Nathan - 35171600 - SINF

March 2020

1 Algorithm

1.1 Overview

Our implementation of the value iteration algorithm starts with defining two probability matrices, one for each dice, that map the probability to move from one cell to another, using the corresponding dice.

Using as example a short five cell layout with no traps we would have(S: safety dice, R: risky dice, $R^{'}$: risky dice with circle variant):

$$S = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad R = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & 0 & \frac{1}{3} & \frac{2}{3} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad R^{'} = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & 0 & 0 & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

We then assign an high arbitrary cost (100) to all cells (cv: cost vector)and loop the following operations:

- Compute the cost of each cell i using either the safety dice or the risky dice(S_i: row i of S):

$$costS_i = 1 + S_i * cv$$

$$costR_i = 1 + R_i * cv$$

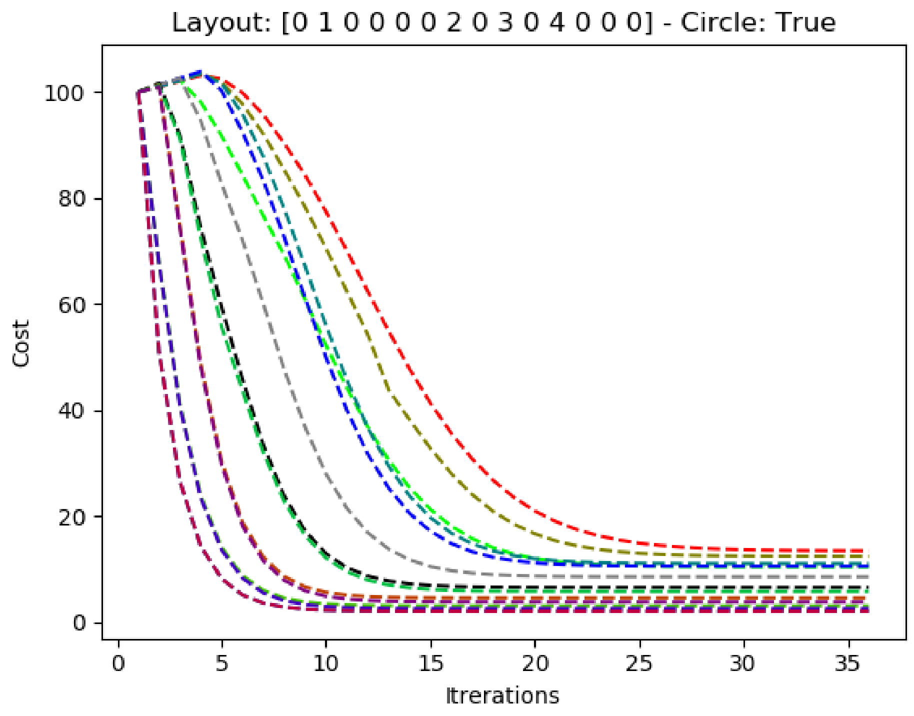
- Chose the dice with lower cost

- Update the cell cost

When we reach convergence (policy and costs stop changing), we have the optimal costs and policy.

1.2 Convergence time

In the figure we see the cost of each cell (shown in different colors) at each iteration of the algorithm with an arbitrary layout:



1.3 Traps

Penalty and Restart Since the probability matrices contain the information of where we move, we can modify them to include both the *restart* and *penalty* traps.
The two traps cause a move from one cell to another to actually move to a third cell; this means we can modify R (only the R matrix need to be modified since the safety dice ignores traps) by *moving* a column; for example if

we add a *penalty* trap to the forth cell:

$$R = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & 0 & \frac{1}{3} & \frac{2}{3} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \longrightarrow \begin{bmatrix} \frac{1}{3} & \frac{1}{3}+0 & \frac{1}{3} & 0-0 & 0 \\ 0 & \frac{1}{3}+\frac{1}{3} & \frac{1}{3} & \frac{1}{3}-\frac{1}{3} & 0 \\ 0 & 0+\frac{1}{3} & \frac{1}{3} & \frac{1}{3}-\frac{1}{3} & \frac{1}{3} \\ 0 & 0+\frac{1}{3} & 0 & \frac{1}{3}-\frac{1}{3} & \frac{2}{3} \\ 0 & 0+0 & 0 & 0-0 & 1 \end{bmatrix}$$

The two traps can be handled this way, the only difference is which column to modify.

Prison trap Since this trap only affects the cost of the cells we only need to adjust when computing the cost of the risky dice:

$$costR_i = 1 + R_i * cv + R_i * isPrison_i$$

($isPrison_i$: vector of binary values that has 1 at the indices corresponding to the prison traps)

Mystery trap This trap is a combination of the three so all three methods are used:

- the probability is *distributed* among the columns instead of being *moved*:

$$R = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & 0 & \frac{1}{3} & \frac{2}{3} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \longrightarrow \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ \frac{1}{9} & \frac{1}{3}+\frac{1}{9} & \frac{1}{3} & \frac{1}{9} & 0 \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{3} & \frac{1}{9} & \frac{1}{3} \\ \frac{1}{9} & \frac{1}{9} & 0 & \frac{1}{9} & \frac{2}{3} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

- The cost is adjusted

$$costR_i = 1 + R_i * cv + R_i * isPrison_i + (\frac{1}{3}) * R_i * isMystery_i$$

($isMystery_i$: vector of binary values that has 1 at the indices corresponding to the mystery traps)

2 Simulations

For testing our algorithm, we did some simulations with different boards to see if we get optimal and coherent results. By coherent we mean that the algorithm doesn't choose the risky dice if there are two traps ahead or if we have a circular board, it doesn't choose the risky dice if it is on the penultimate cell.

Circular board with no trap: For the first simulation, we used a circular board with no trap. And for this board we got a result where it chooses the risky dice on every cells with the exception of the cells 10 and 14 which are the penultimate cells of each lane. For the costs of each cells, we have a cost of 9.56 for the first cell and it keeps decreasing each time we advance on the board.

Classic board but not circular: This time, we used an average board, which can be realistically encountered, to see the reaction of the algorithm. So we can describe our board with the schema below:

$$layout = [0, 3, 0, 0, 4, 1, 0, 0, 2, 0, 4, 0, 1, 0]$$

And we get an optimal policy like this:

$$P_{opt} = [2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2]$$

We can see that the algorithm doesn't use the risky dice often with exception on the penultimate cells and the first one which is a good reaction because the board is not circular and the two cells following the first one have no punitive traps. We notice that waiting a turn on a trap of type 3 is better than using the safe dice. For the cost of the first cell, we get a value of 15.51 that keeps decreasing the closer the cell is to the last cell.

$$costs_{expec} = [15.51, 14.51, 12.5, 13.5, 11.5, 9.5, 7.5, 5.5, 3.5, 1.5, 7.5, 5.5, 3.5, 1.5]$$

We also launch this simulation one thousand times with a different starting cell each times to calculate the error who corresponds to:

$$err^i = c^i_{expec} - c^i_{avg}$$

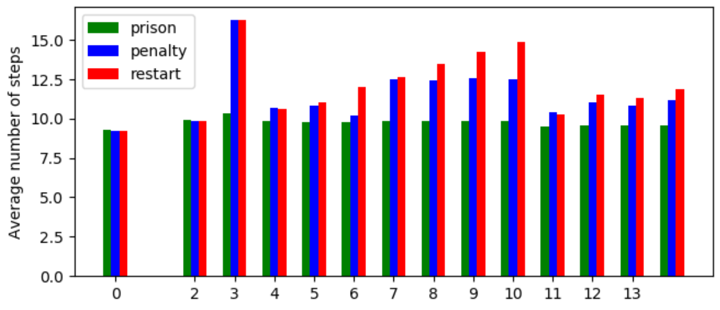
The average cost is calculated with the different simulations and corresponds to an index i. And for this simulation we get an error for each cells equals to:

$$err = [-0.01, 0.09, 0.05, 0.01, 0.02, -0.05, -0.02, -0.01, 0.01, 0, 0, 0, 0, 0]$$

We can observe the maximum error corresponding to 0.09 which is very low.

2.1 Traps impact

We then simulated games using only the risky dice, with only one trap, of type 1,2 or 3, in each cell (except the first one):
From this image we can deduce:



- As seen in the previous simulations, the type 3 trap has always a low impact
- The type 1 trap increases the cost the closest it is to the last cell
- The cell 3 is the worst place to have a trap of any kind; this is because is very likely to be triggered
- The type 2 trap has a constant cost in three zones: 4-6, 7-10, 11-13. Since it always makes the player go three steps back it makes sense to be a constant value while it is peculiar that in 7-10 the value is higher.

Investigating the last point we noticed that if there is even one trap in cells 4-10, the optimal policy will choose the safe dice at cell 2, presumably to maximize the chance to get on the fast lane; furthermore if the trap is in cells 7-10, it will use the safe dice to avoid it, while if the trap is in cells 4-6 it uses the risky dice; this because triggering a trap in cells 4-6 will give a second chance at entering the fast lane.

3 Conclusion

In conclusion, we designed an algorithm that solve a "Markov decision process" problem for a Snakes and ladder game. We saw that it can produce a good expectation for reaching the goal with a minimum turn with the good dices. The algorithm converges in a very low amount of iterations which makes good performances for its time complexity. With the simulations, we also saw that our maximum error value for a certain cell is pretty low which means that our algorithm is accurate for solving this problem. In resume, the algorithm finds a solution which is coherent and accurate to work in real conditions.