Group 8 :
BAILLY Gabriel
WAUTIER Lara
ZONE Corentin

Program:
DATS2M
DATS2M
DATS2M

LSINF2275 - Project 1

Markov Decision Processes

2020-2021

UCLouvain

LSBA / EPL

1 Introduction

This project is aimed at realising a practical application of Markov decision processes, by solving optimal strategy of a "Snakes and Ladders" games. It's a board game with 15 squares, which are numbered, can countain various traps, and are set to have the chance to go through a slow or a fast lane (see guidelines). The key function we will implement is `markovDecision(layout,circle)`, which solves the optimal strategy using the "value-iteration algorithm" method. The description and implementation of the function will be described later in this paper.

2 Method[1]

As said, we have to use the Value-Iteration algorithm. We aim at reaching the goal state $d = 15$ from initial state $s_0 = k_0 = 1$ with minimal cost (the less turns as possible), which can be done by using an optimal strategy π^* for each fixed state of the game. This optimal policy is the one which solves :

$$V^*(k_0) = \min_{\pi} \mathbb{E}_{\mathcal{S}} \left[\sum_{t=0}^{\infty} c\big(a \in \mathcal{A} \big| s_t\big) \;\Big|\; s_0 = k_0 \,,\, \pi \right]$$

where

- $\mathcal{S}$ is the set of time-dependent states s_t.
- $\mathcal{A}$ is the set of actions $\{a_1, a_2, a_3\}$ (respectively *security*, *normal* and *risky* dice).
- $c\big(a \in \mathcal{A} | s_t\big)$ is the cost function of taking action conditional on state, and it equals 1 for every action in the game (one action will cost one more turn).

Now, let $\mathcal{K}$ be the set of the k_i time-independent states (the squares on the board), with $i = 1, \dots, 15$. Then, we can find the optimal policy with value-iteration using the Bellman equation :

$$\begin{cases} \hat{V}(k_i) = \min\limits_{a \in \mathcal{A}} \; c(a|k) + \sum\limits_{k'=1}^{15} p(k'|k,a)\hat{V}(k') & \text{for } i = 1, \dots, 14 \\ \hat{V}(d = k_{15}) = 0 & \text{(absorbing state)} \end{cases}$$

where $p(k'|k,a)$ is the probability to land on any square k' in the board, given the current square and the chosen dice. Those probabilities depend on the dice, the probability of triggering a trap and the probability to go on the fast or slow lane. Those probabilities will be discussed in details in the next section.

[1] Mathematical notation blends [1] and [3]

3 Implementation

Before going into details, let us show in a pseudo-code how we solved the problem in practice:

Algorithm 1 Value-iteration (based on [1] and [3])

Result: determines the optimal strategy.

input : `layout` : a vector containing values representing the squares
`circle` : boolean, `True` if you must land on 15 to win

output: `Expec` : the (1×14) vector $V^*(\mathcal{K})$ of expected cost for each square
`Dice` : the (1×14) vector $\pi^*(\mathcal{K})$ of dice to throw for each square (the policy)

1 set $V(\mathcal{K})$ a (1×15) vector full of 0
2 set $\pi(\mathcal{K})$ a (1×14) vector full of 0
3 set $\theta \in \mathbb{R}^+$ arbitrary small
4 set `end` $\leftarrow$ `False`
5 **while** `end == False` **do**
6 set `converged` $\leftarrow 0$
7 **for** $k \in 1:14$ **do**
8 $v \leftarrow V(k)$
9 $V(k) \leftarrow \arg\min_a \left\{ c(a|k) + \sum_{s'} p(s'|s,a) V(s') \right\}$
10 $\pi(k) \leftarrow a$
11 $\Delta \leftarrow |v - V(k)|$
12 **if** $\Delta < \theta$ **then**
13 `converged` $\leftarrow$ `converged` + 1
14 **end**
15 **end**
16 **if** `converged == 14` **then**
17 `end` $\leftarrow$ `True`
18 **end**
19 **end**

The main idea behind the *while* loop is that we need to iterate the expected values for each square k from 1 to 14 conditionally on the others. We choose the action that minimizes this value (see next paragraph), compare it to the previous one and check if the difference is very small ($< \theta$, a very small positive number). We repeat this until every square has such a small difference with its previous value.

What was done in line 7 will now be explained. Let *Trap* be an update function of the expected value of k if its trap has been activated. Then, we drew a tree of possibilities to determine how the value iteration will work, so we have the following formulas to iterate the values:

- For the **security dice** $a = a_1$, we can either make moves of $+0$ or $+1$ with uniform probability. Also, we don't trigger any trap. Then, the expected values are iterated such as

$$\begin{cases} V(k) = 1 + \frac{1}{2}V(k) + \frac{1}{2}V(k+1) & \text{if square} \neq 3 \\ \\ V(3) = 1 + \frac{1}{2}V(3) + \frac{1}{4}V(4) + \frac{1}{4}V(11) & \text{if square} = 3 \end{cases}$$

- For the **normal dice** $a = a_2$, we can move from $+0$ up to $+2$ with uniform probability. Also, we have 50% chance to trigger the trap. Then, the expected values are iterated such as

$$\begin{cases} V(k) = 1 + \left(\frac{1}{6}V(k) + \frac{1}{6}\,Trap\{V(k)\}\right) + \left(\frac{1}{6}V(k+1) + \frac{1}{6}\,Trap\{V(k+1)\}\right) \\ \qquad\quad + \left(\frac{1}{6}V(k+2) + \frac{1}{6}\,Trap\{V(k+2)\}\right) & \text{if square} \neq 3 \\ \\ V(3) = 1 + \left(\frac{1}{6}V(3) + \frac{1}{6}\,Trap\{V(3)\}\right) + \left(\frac{1}{12}V(4) + \frac{1}{12}\,Trap\{V(4)\}\right) \\ \qquad\quad + \left(\frac{1}{12}V(5) + \frac{1}{12}\,Trap\{V(5)\}\right) + \left(\frac{1}{12}V(11) + \frac{1}{12}\,Trap\{V(11)\}\right) \\ \qquad\quad + \left(\frac{1}{12}V(12) + \frac{1}{12}\,Trap\{V(12)\}\right) & \text{if square} = 3 \end{cases}$$

- For the **risky dice** $a = a_3$, we can move from $+0$ up to $+3$ with uniform probability and we always trigger the traps. Then, the expected values are iterated such as

$$\begin{cases} V(k) = 1 + \frac{1}{4}\,Trap\{V(k)\} + \frac{1}{4}\,Trap\{V(k+1)\} \\ \qquad\quad + \frac{1}{4}\,Trap\{V(k+2)\} + \frac{1}{4}\,Trap\{V(k+3)\} & \text{if square} \neq 3 \\ \\ V(3) = 1 + \frac{1}{4}\,Trap\{V(3)\} + \frac{1}{8}\,Trap\{V(4)\} \\ \qquad\quad + \frac{1}{8}\,Trap\{V(5)\} + \frac{1}{8}\,Trap\{V(6)\} \\ \qquad\quad + \frac{1}{8}\,Trap\{V(11)\} + \frac{1}{8}\,Trap\{V(12)\} \\ \qquad\quad + \frac{1}{8}\,Trap\{V(13)\} & \text{if square} = 3 \end{cases}$$

The *Trap* function will convert $V(k)$ to $V(1)$ for the **trap 1**, to $V(k-3)$ for **trap 2**, to $1 + V(k)$ for the **trap 3** (as it adds the cost of one more turn) and to $\frac{1}{15}\sum_{k=1}^{15} V(k)$ for the **trap 4**.

4 Results

Now that we have shown how to implement the solver to our problem using the theoretical basis, we will show some of the results obtained with the algorithm. For this purpose, we will arbitrarily define a layout with various traps :

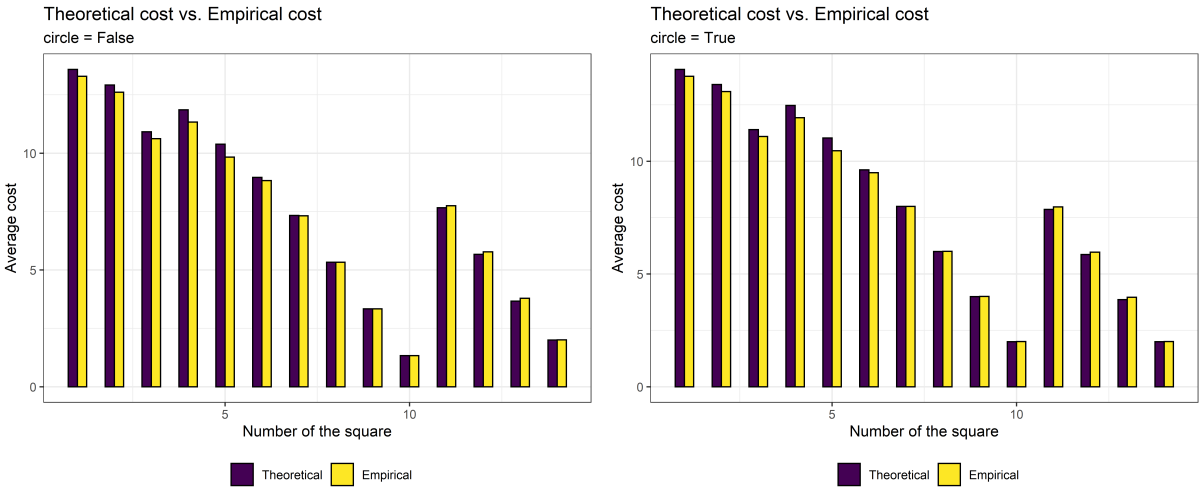
$$\texttt{layoutTest} = \big[0, 1, 0, 2, 3, 0, 2, 0, 1, 0, 4, 2, 0, 4, 0\big]$$

The optimal choices of dice computed through the Markov decision process for that particular layout of traps are shown below, in vector form as the output of the function. In those vectors, 1 represents the *security* dice, 2 the *normal* dice and 3 the *risky* dice. We see that the algorithm will not take the risk of going beyond the last square when we are in a circle game. We can also observe that the algorithm will avoid traps, but in some other tests it will not avoid the traps of type 4 when they are positioned close to the beginning of the game.

$$\pi^*(\mathcal{K}) \equiv \begin{cases} \texttt{Dice} = \big[2, 1, 3, 2, 3, 2, 1, 1, 1, 3, 1, 1, 3, 1\big] & \texttt{circle = False} \\ \texttt{Dice} = \big[2, 1, 3, 2, 3, 2, 1, 1, 1, 1, 1, 1, 2, 1\big] & \texttt{circle = True} \end{cases}$$

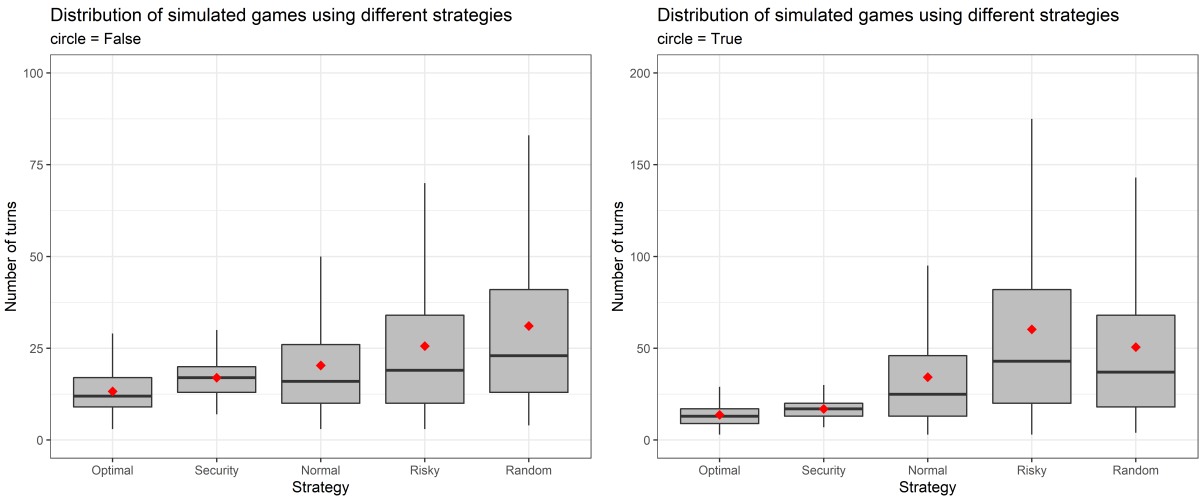
Now that we showed the optimal strategy, we will proceed to some empirical analysis to see if our model is unbiased for solving the problem, and if it is really a good strategy compared to other possible policies (one dice only, random, human intuition...)

First, we compare the **theoretical expected cost** we obtained with our algorithm and the **empirical average cost**. In order to evaluate if the optimal policies really meet the expected cost for each square for n games when $n \to \infty$, we simulated one million games in the file `simGames.py` and compared the mean for each square to the computed values $V(\mathcal{K})$ on a barplot.



We can observe that, for an important amount of simulations, the mean for each square doesn't differ a lot from the theoretical values. The Markov decision process we modelled seems therefore *unbiased*, with and without the circle rule.

Now, let us compare the empirical average cost obtained by the **optimal strategy** and the one from **other sub-optimal strategies**. We reported in boxplots the results from one million simulated games per strategy. We have simulated : the *optimal policy* computed by the value-iteration algorithm, the *security only*, the *normal only*, the *risky only* and finally the *purely random* strategy. We repeated this process both for the case where circle rule applies and the case where it does not. Note that the red diamonds on the boxes are the mean for each strategy. To make the plots more readable, we also removed the outliers.

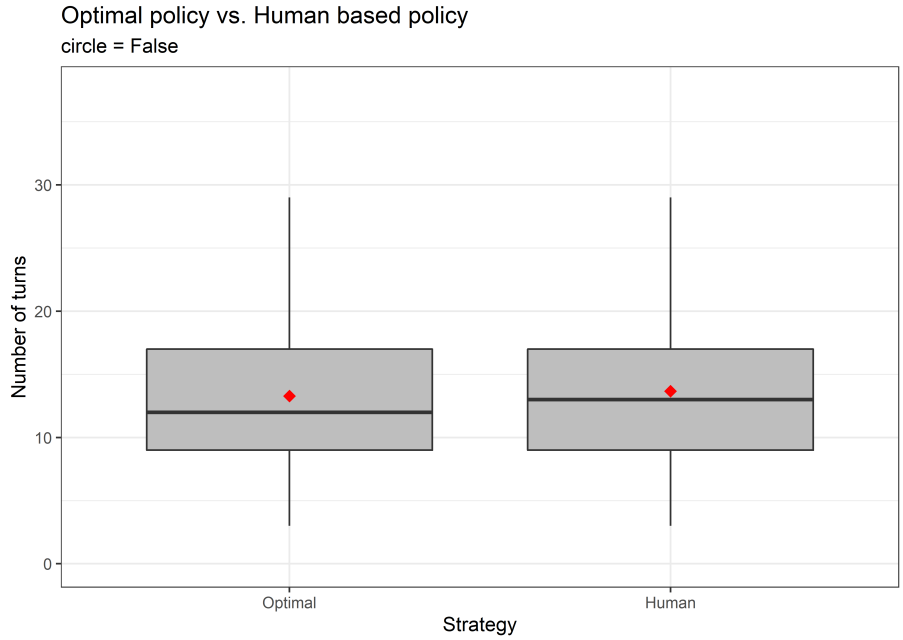


We can observe that the optimal policy we computed seems to be the most efficient in both cases, it doesn't have the lowest variance but it has the lowest mean and median.

Finally, we look for an human-based strategy ourselves : we simply look at the board and write down an intuitive strategy for all squares in the case where circle rule does not apply (it really is quite similar to the optimal strategy except for 2 squares, but we did not look at it to write down our new policy). We define a vector with this new strategy $\tilde{\pi}$:

$$\texttt{stratHuman} = \big[2, 1, 3, 1, 3, 1, 1, 1, 1, 3, 3, 1, 3, 1\big]$$

We than simulated one million games with this human brain solution and compared its distribution to the optimal one:



We see that the optimal policy stays the most efficient. They have a similar dispersion but the median and mean stay lower for the optimal (average cost is 13.28 for π^*, against 13.66 for our own strategy $\tilde{\pi}$).

5 Conclusion

In the frame of solving a pretty simple game, we implemented an algorithm which is quite efficient to solve the optimal decision making in a stochastic environment. Thanks to the ability to compute a large number of simulations, we could confirm that it is pretty accurate to predict the number of turns for a given layout, and that its performance is superior to other strategies we have tested, which seems intuitive for the "one dice only" and random policies but we have also seen that it is superior to the intuition of human brain.

References

[1] **Saerens, M., (2021).** *Markov decision processes: A short, informal, introduction (dynamic decisions)* [Slides]. Retrieved from UCLouvain LSINF 2275 Moodle.

[2] **Silver, D., (2015).** *Lectures on Reinforcement Learning* [Slides]. Retrieved from `https://www.davidsilver.uk/teaching/`

[3] **Sutton, R. & Barto, G. (2018).** *Reinforcement Learning: An Introduction*. MIT Press, A Bradford Book. Retrieved from `https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf`