LSINF2275 - Data Mining and Decision Making - Project 1
Markov Decision Processes

Group 10
d'Oultremont Augustin - 2239 1700 - INFO
de Voghel Brieuc - 5910 1600 - INFO
Lemaire Valentin - 1634 1700 - INFO

Monday 5th April, 2021

Abstract

This report details and discusses our methods for solving the project's Snakes and Ladders game with Markov Decision Processes (MDP). We begin with a short explanation of the mathematical model and the method used to find the optimal strategy. Then, we briefly describe our implementation. We finish with a comparison (a) between theoretical and empirical solutions and (b) between optimal and sub-optimal strategies for different layouts.

1 Mathematical Model

In this section we discuss how we mathematically modeled the game, the underlying Markov Chain and how we can apply the value iteration algorithm to derive an optimal strategy.

1.1 Markov Decision Process and Bellman equations

Snakes and Ladders as a Markov chain When faced with this problem, one can immediately see that we are faced with a Markov chain. Indeed, we have states and transition probabilities which are strictly independent of previous states.

The only case that needs a little trick is the prison. Indeed the state is different when we fall on a prison whether we trigger the trap or not. To avoid this complication we can take prisons into account when computing the cost of a certain action as explained in Section 1.2.

Notation In the following developments we will use the following notations:

- $V^*(k)$ the expected cost of playing the optimal strategy from state k
- $U(k)$ the set of possible actions from state k
- $c(a|k)$ the cost of executing action a when we are in state k
- $p(k'|k,a)$ the *à priori* probability of reaching state k' when in state k and executing action a
- d is the final state (objective), n is the number of states

Optimality Conditions For a solution to be optimal, it has to match the Bellman equations of optimality:

$$\begin{cases} V^*(k) = \min\limits_{a \in U(k)} \left\{ c(a|k) + \sum\limits_{k'=1}^{n} p(k'|k,a) V^*(k) \right\} \text{ for } k \neq d \\ V^*(k) = 0 \text{ for } k = d \end{cases} \tag{1}$$

1.2 Modeling the game

In the context of this game, we modeled the variables in the following way.

Actions At each state 3 actions are available. Indeed the player can throw any of the 3 dice. We therefore have:

$$U(k) = \{\texttt{SECURITY}, \texttt{NORMAL}, \texttt{RISKY}\} \quad \forall k$$

Costs The cost of throwing a die is always 1. However to this, we must add the probability of landing on a prison and triggering the trap (multiplied by the cost of loosing a turn i.e. 1).

$$c(a|k) = 1 + \sum_{\substack{k'=1 \\ k' \ is \ PRISON}}^{n} p_{nt}(k'|k,a) \cdot p(TRAP|a) \cdot 1$$

Where $p_{nt}(k'|k,a)$ is the probability of falling on a certain square if none of the squares are trapped. Indeed, since there are no cascading traps, we need to ensure we have fallen on this square because of a throw of dice and not because we have been teleported here due to a trap.

A priori probabilities Another value required in the Bellman equations is the *à priori* probabilities of entering a certain state knowing the current state and the action played. For this, we computed the probabilities for each combination of die and state to fall on another state taking into account :

- The probabilities of each possible result of the die.
- The probabilities of branching on either the fast or slow lane when on square 3.
- The probabilities of triggering a trapped square (and maybe being teleported somewhere else on the board according to the type of square we fall onto).
- The board being circular or not (i.e. must land exactly on goal state, or may overstep it).

We thus obtain the transition matrix for every die, corresponding to a particular board layout.

1.3 Implementation

The Bellman equations (1) give a criterion for the optimum but do not provide a way to compute the optimal action at each state, nor the expected cost of each state. This is done by the value iteration algorithm which initializes arbitrary costs to each of the states and iterates over the Bellman equations until convergence, using following formulas :

$$\begin{cases} \hat{V}_{t+1}(k) = \min\limits_{a \in U(k)} \left\{ c(a|k) + \sum\limits_{k'=1}^{n} p(k'|k,a) \hat{V}_t(k) \right\} \text{ for } k \neq d \\ \hat{V}_{t+1}(k) = 0 \text{ for } k = d \end{cases} \tag{2}$$

Upon convergence, we thus expect $\hat{V}_t \approx V^*$.

Algorithm Using this idea, the algorithm was implemented according to the following pseudo-code.

Algorithm 1 Value Iteration

$V(0,k) \leftarrow 0 \quad \forall k$
$\Delta \leftarrow \varepsilon + 1$
$t \leftarrow 1$
while $\Delta > \varepsilon$ **do**
    for all $k \neq d$ **do**
        $C \leftarrow 1 + \sum\limits_{\substack{k'=1 \\ k' \ is \ PRISON}}^{n} p_{nt}(k'|k,a) \cdot p(TRAP|a)$
        $V(t+1,k) \leftarrow \min\limits_{a \in U(k)} \left\{ C + \sum\limits_{k'=1}^{n} p(k'|k,a) V(t,k') \right\}$
    $\Delta \leftarrow \max\limits_{k} (|V(t+1,k) - V(t,k)|)$
    $t \leftarrow t + 1$

2 Model validation

To ensure the theoretically obtained model is correct (i.e. that the expected cost to finish the game from the starting position is indeed the one yielded by the value iteration algorithm), we simulate games and observe for each state the average number of turns needed to reach the goal state. With a large number of simulations, this average should converge to the theoretical value obtained by the value iteration algorithm.

2.1 Testing strategy

Strategy To compute the empirical mean, we simulate 1 million games, throwing the dice corresponding to the strategy yielded by the value iteration algorithm. Three random components are involved: the number of steps indicated by a die, the fact that a trap is triggered or not when using the `NORMAL` die and the branching when on square 3.

Computation For each of the simulations we maintain a list of already visited states and each time the agent plays from a state, we increment a counter for that state (by two if the player falls on a prison and triggers the trap, by 1 otherwise). When all simulations ended we compute the average for each of the states.

Layouts Throughout this section we refer to different layouts used in our tests. Here follows a short description of these layouts. All layouts are tested with and without the `circle` flag.

1. **Ordinary Layout** In this layout, all tiles are ordinary. It contains no traps.
2. **Penalty Layout** In this layout, all tiles are Penalty tiles, except for the first and last tiles.
3. **Prison Layout** In this layout, all tiles are prison tiles, except for the first and last tiles.
4. **Gamble Layout** In this layout, all tiles are gamble tiles, except for the first and last tiles.
5. **Restart Layout** In this layout, all tiles are restart tiles, except for the first and last tiles.
6. **Custom Layout 1** In this layout, all tiles are ordinary, except for tiles 8, 9 and 13 which are respectively gamble, restart and penalty tiles. This layout is represented in Figure 1.
7. **Custom Layout 2** This layout is more complicated and is represented in Figure 2.

The legend for both custom layout figures is in Figure 3.

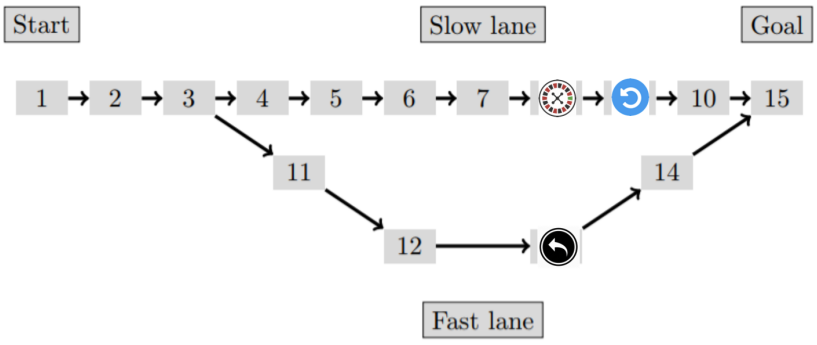


Figure 1: Custom Layout 1

2.2 Results

When running the tests detailed in section 2.1, we obtain both a theoretical expectation of number of turns from each state to the goal and an experimental mean for the same value. Figures 4a and 4b show these values for our custom layout 2 with and without circle. Table 1 regroups, for all the layouts, the norms of the difference between theoretical and experimental number of turns to reach the goal state from each state.

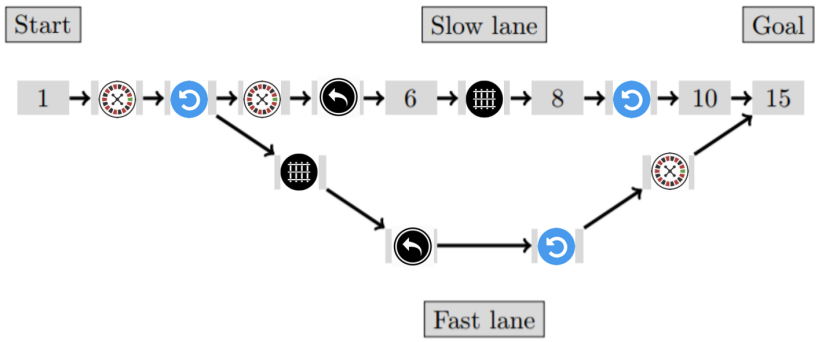
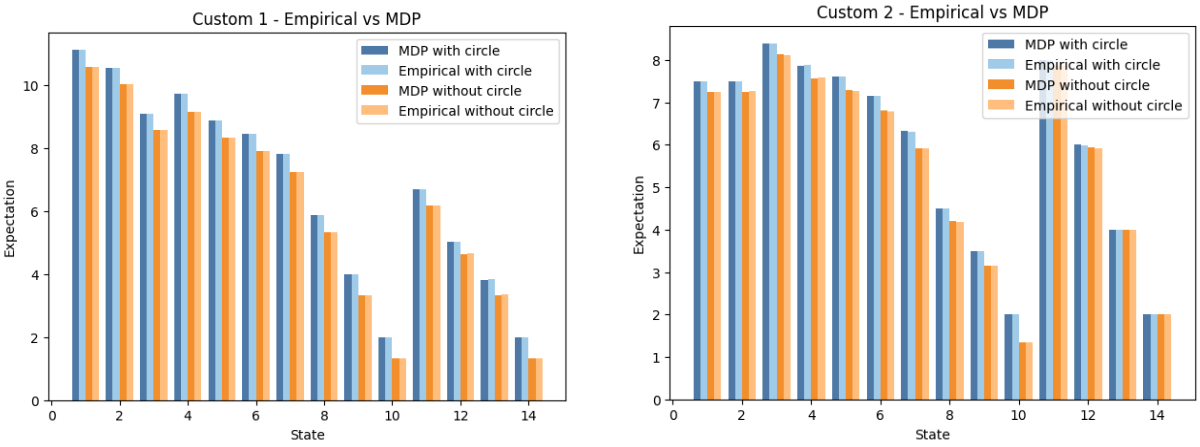


Figure 2: Custom Layout 2

Figure 3: Layout legend



(a) Custom Layout 1

(b) Custom Layout 2

Figure 4: Comparison between theoretical and empirical results

Layout	Circle	Norm of differences
Ordinary Layout	True	0.0034
	False	0.0027
Penalty Layout	True	0.0152
	False	0.0163
Prison Layout	True	0.0185
	False	0.0143
Gamble Layout	True	0.0238
	False	0.0188
Restart Layout	True	0.0474
	False	0.0542
Custom Layout 1	True	0.0325
	False	0.0340
Custom Layout 2	True	0.0294
	False	0.0456

Table 1: Norm of differences between theoretical and empirical results. The smaller, the better (i.e. empirical results match MDP computation).

When observing these results - which compare expectations obtained through theory of probabilities and empirical means obtained through independent random realizations - we quickly see that theory and practice match quite well. Indeed using random realizations of the game and computing the mean should converge to the expectation. This is indeed what we are seeing in figures 4a and 4b.

Table 1 also shows that no matter the layout (different types of traps, with or without circle) we always obtain the same order of magnitude between the theoretical results and the empirical ones (namely in the order of 10^{-2}). We expect it to only go down when simulating more games.

All these observations lead us to believe that our mathematical model was well posed, that the expectations computed are trustworthy and that they indeed tell us how many turns we will need to reach the goal at each state.

3 Optimality verification

To convince ourselves of the fact that the strategy obtained through our MDP is indeed optimal we proceed to compare it to different strategies. We expect each of these strategies to be less efficient (i.e. need more turns) than the one yielded by the value iteration algorithm.

3.1 Testing strategy

For this validation we wish to compare the derived strategy with different strategies. The following strategies are used.

1. **Secure strategy** In this strategy, we only throw the secure die. Thus making steps of 0 or 1 steps at every turn and never triggering any traps.
2. **Normal strategy** In this strategy we only throw the normal die. Thus making steps of 0, 1 or 2 at every turn and triggering with a chance of $1/2$.
3. **Risky strategy** In this strategy we only throw the risky die. Thus making steps of 0, 1, 2 or 3 at every turn and triggering any trap we fall onto.
4. **Random strategy** In this strategy we play completely at random. At every turn we throw one of the 3 dice with equal probability independently of the tile we are on, or of previous throws.
5. **Greedy strategy** As a more relevant point of comparison we implemented a greedy strategy. The algorithm yielding this strategy computes for each state and for each die the expected distance to the goal after throwing this die from this state (taking prisons into account by adding an extra distance of 1 to a prison tile if playing the risky die and of 0.5 if playing the normal die).

For our tests, we simulate 1 million games for each of the 7 layouts presented in section 2.1 (with both `circle` options) for each of the 5 strategies presented above as well as for the optimal strategy yielded by the value iteration algorithm. These results are presented below in section 3.2.

3.2 Results

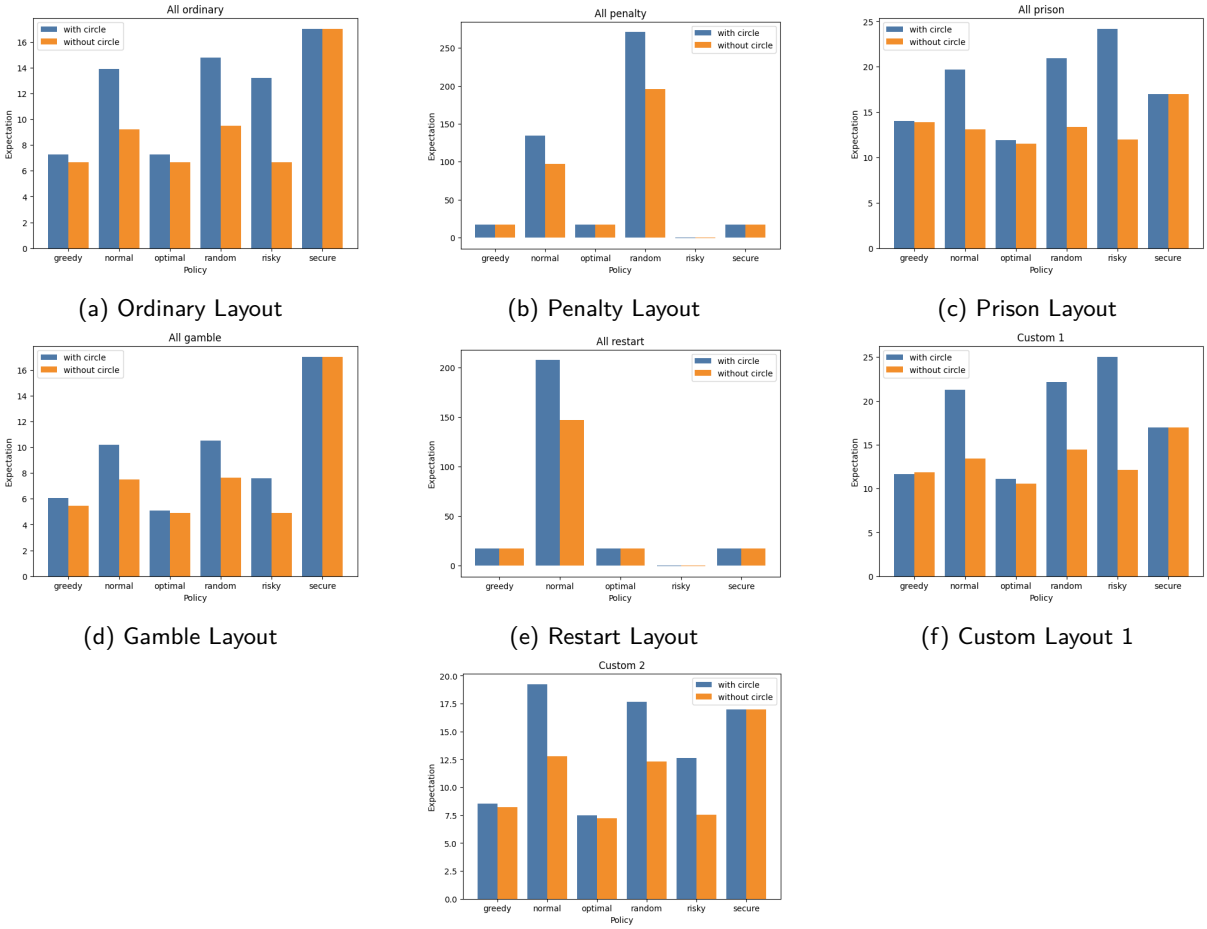
The results for the tests presented in section 3.1 are presented in figures 5a to 5g.

3.2.1 Optimal policy analysis

To understand the optimal strategy, we'll start by analyzing the optimal policy on the custom layout 1, with circle set to `True`[1]. Here are a few suppositions we can make, based on observations.

- The general strategy seems to be to go as fast as possible (using the risky die the most), with a few exceptions.

[1]The complete optimal policy for the custom layout 1 with circle is [1:RISKY, 2:RISKY, 3:NORMAL, 4:RISKY, 5:RISKY, 6:NORMAL, 7:RISKY, 8:RISKY, 9:SECURITY, 10:SECURITY 11:RISKY, 12:RISKY, 13:NORMAL, 14:SECURITY]



(a) Ordinary Layout

(b) Penalty Layout

(c) Prison Layout

(d) Gamble Layout

(e) Restart Layout

(f) Custom Layout 1

(g) Custom Layout 2

Figure 5: Strategy performance on different layouts

- When a tile n is trapped with restart or penalty, the risky die should not be used on tile $n-3$. This is quite obvious, since the advantage of the risky die is to have the possibility to throw a 3, but in that case, the player would land on the trap (restart or penalty) and have a 100% chance of activating it, thus ruining the advantage of the die. This can be observed on the custom layout 1, where the optimal policy for tiles 3 and 6 is to use the normal die.
- The cost of overshooting the target (in the case when `circle` is on) is also too high to take the risk. This explains the strategy for tiles 10 (security die) 13 (normal die) and 14 (security die) for the custom layout 1.
- The cost of a restart increases the further the player is (a restart from tile 10 is far worse than a restart from tile 3). This explains why the strategy for tile 9 is to use the security die (instead of the normal, according to the two preceding rules only).

We can then compare the previous strategy with the optimal policy for the same layout but with circle set to false[2] and observe the following.

- We can see that on tiles 10 and 14, having no risk to overshoot, we get back to the basic strategy which is to use always use the risky die.
- For the tiles 8 and 13, the optimal strategy seems to be more careful. This comes probably from the fact that when circle is False, it has a chance to reach the goal in 1 risky throw, but it will take many more throws if it uses a security throw since it will avoid an overshoot. This results in a higher reward if the

[2]The complete optimal policy for the custom layout 1 without circle is [1:RISKY, 2:RISKY, 3:NORMAL, 4:RISKY, 5:RISKY, 6:NORMAL, 7:RISKY, **8:SECURITY**, 9:SECURITY, **10:RISKY** 11:RISKY, 12:RISKY, **13:SECURITY**, **14:RISKY**], with the differences in bold

player does not fall on tile 9's restart compared to the non-circle version where it can always "speedrun" the last tiles.

For the 2nd custom layout with `circle`[3], we can observe that the optimal strategy is to play a riskier game in the beginning. We can observe that there are 2 gamble traps in the beginning, which may be more of an advantage than a setback, since it will probably bring the player closer to the target.

Looking at the 2nd custom layout without `circle`[4], we can conclude that the optimal solution never cares about taking the fast lane, since it will only shorten the road by 3 tiles, with a cost of slowing down a lot at the start if we roll the security or normal dice.

3.2.2 Strategy Comparison

When comparing the different strategies, we can observe that ... :

1. No matter the layout, the secure policy will always give the exact same result. This makes sense as the security die prevents us from triggering traps and prevents overshooting the goal in circular layouts.
2. In risky, normal and random strategies, the expected number of turns for circular layouts is always higher than for non-circular ones. Again this is sensible as those strategies allow to overshoot (or encourage it) which could lead to go further than necessary at the end of the game and thus restarting.
3. The optimal strategies have very close empirical means in the same layout with `circle` set to True or False. This is explained by the fact that the optimal strategy will choose a die at the end of the game such that overshooting does not happen in order to avoid having to start the game all over again. This can also be observed in state-by-state figures 4a and 4b from Section 2.2.
4. As expected, for all layouts, whether they are circular or not, the optimal policy yielded by the value iteration algorithm remains undefeated with observed mean number of turns lower or equal to any of the other strategies, and when they are equal, it is because the policies are actually equal as well.
5. The greedy method gives relatively good results (compared to the other naive strategies) but it never surpasses the optimal one and remains slightly weaker for non-trivial layouts.

N.B. When a bar in the bar chart is set to 0 - as for the risky strategy in Penalty and Restart layouts (respectively figures 5b and 5e) - it means that the objective is simply not attainable using this strategy (the risky die always triggers the penalty and restart traps, meaning that the player cannot move forward).

4 Conclusion

We have been able to develop a method for computing the optimal dice selection strategy and we have proven its optimality by comparing the obtained results with empirical testing.

The empirical tests were both performed to evaluate the MDP's robustness against different game layouts, but also its superiority compared to other strategies, both naïve and greedy.

In the last section we also discuss how reasonable the optimal strategy seems, knowing and understanding the larger implications of the die selection.

[3]The complete optimal policy for the custom layout 2 with `circle` is [1:RISKY, 2:RISKY, 3:RISKY, 4:RISKY, 5:RISKY, 6:NORMAL, 7:RISKY, 8:RISKY, 9:NORMAL, 10:SECURITY 11:NORMAL, 12:SECURITY, 13:SECURITY, 14:SECURITY]

[4]The complete optimal policy for the custom layout 2 with `circle` is [1:RISKY, 2:RISKY, 3:RISKY, 4:RISKY, 5:RISKY, 6:NORMAL, 7:RISKY, 8:RISKY, 9:RISKY, 10:RISKY 11:NORMAL, 12:RISKY, 13:SECURITY, 14:SECURITY]