École Polytechnique de Louvain

Project 1 : Markov Decision Process

Academic year 2019 – 2020

LSINF2275 - Data Mining and Decision Making

Group 23

Gevorgyan Edgar - 20181600 - INFO2MS
Navarre Louis - 12351600 - INFO2MS

March 22, 2020

1 Introduction

The objective of the project is to implement a *Markov Decision Process (MDP)* in the *Snakes and Ladders* game. The map is shown below:

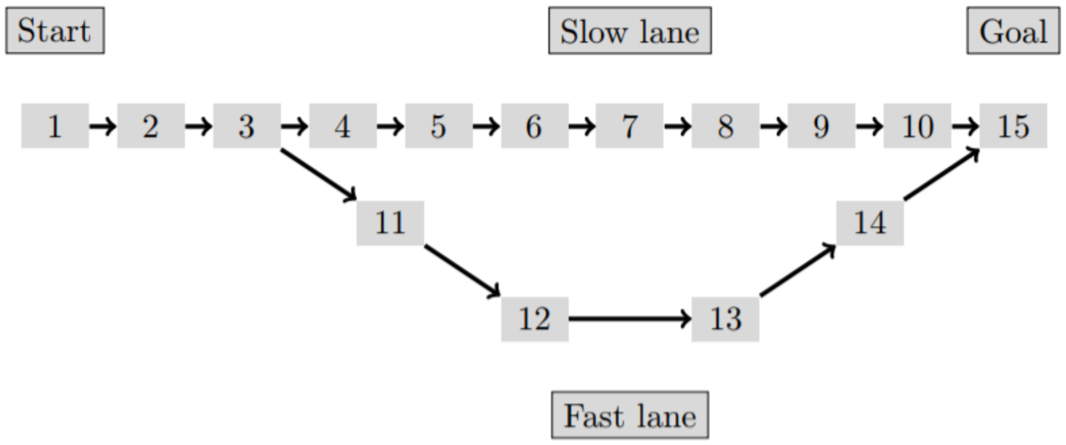


Figure 1: *Snakes and Ladders* game's map

First, this report explains the method that is used to determine the optimal strategy for the dice choice, and how it is implemented in the context of this project. Of course, the detailed and commented source code (in `Python`) is also available. Then, a comparison between theoretical and experimental results is provided, in order to verify if the *Markov Decision Process* gives a good estimation of the real value of the number of turns required to complete the game. Finally, the document contains a discussion between the performance of the strategy computed by the *MDP*, and other (simpler) strategies. As *MDP* is supposed to give the optimal strategy, one would expect it to outperform the others.

2 Optimal strategy

Snake and Ladders is a stochastic game. Indeed, randomness has a big impact on the outcome of the game: the dice (which is assumed to be fair) decides if the player moves forward or not, or if the player can go on the fast or the slow path. However, the positions of the traps is deterministic, and allows to compute a strategy to avoid such traps. This strategy is only based on the choice of the dice at a given position on the map. Indeed, when the player is at position x, the history of moves of the player is irrelevant for its future; only its current position is important. Then, the game possesses the *Markov Property*. As a result, a *Markov Decision Process* can be used in order to find the optimal strategy.

The goal of the algorithm is to minimize the expected cost at each state. As the outcome at a specific state is not deterministic, one must take into account all possibilities to compute the expected cost at this state, which must me minimized. Moreover, as the expected cost at state x is an estimate, and as it may depend on states $y > x$, as well as on states $y < x$, the computation of the expectation must be repeated until convergence. An iteration for a state k is shown below (d is the destination state):

$$\begin{cases} \hat{V}(k) = \min_{a \in U(k)} \{ c(a|k) + \sum_{k'=1}^{n} p(k'|k,a) \cdot \hat{V}(k') \}, & \text{if } k \neq d \\ \hat{V}(d) = 0, & \text{otherwise.} \end{cases}$$

Where:

- $\hat{V}(k)$ is the expected cost to reach destination d from state k,
- $p(k'|k,a)$ is the probability to reach state k' from state k using dice a,
- $c(a|k)$ is the cost to use the dice a from state k (in this case this cost is always equals to 1, expect if state k contains a trap of type 3, freezing the player),
- $U(k)$ is the set of actions (i.e., the dice) that can be taken at state k (in this project, this set is independent of the state k: $U(k) = \{1,2\} \forall k$).

Thus, the optimal action (i.e. optimal dice to use) at state k is

$$\underset{a \in U(k)}{\arg\min} \{c(a|k) + \sum_{k'=1}^{n} p(k'|k,a) \cdot \hat{V}(k')\}.$$

3 Implementation

This sections discusses the particularities of the implementation of the computation of the *MDP* for this game. A first choice is the "until convergence" condition of the value-iteration algorithm. Here, a sum-of-squared error is used. Then condition to stop the iterations is $||\hat{V}_{new} - \hat{V}_{previous}||^2 \leq \theta$, with $\hat{V}_{new}$ and $\hat{V}_{previous}$ be respectively the vector of expected values at the current iteration, and at the previous iteration.

Another particularity of the implementation is how the value-iteration is performed. Indeed, during the lectures, two methods were explained to compute it. The first was building a transition probability matrix, and computing the expectation using a matrix-vector product until convergence. The second was to build the new values of expectation separately. This implementation uses the second method. First, the transition probability matrix is sparse (even if `numpy` can store efficiently this kind of matrices). Secondly, this approach is easier to implement the constraints of the game. Indeed, for example, the 4th kind of trap is easy to compute using this method: if the player is at state x, x and $x+1$ are states without trap, and $x+2$ contains a trap of type 4, then the expectation when taking dice 2 (the normal dice) becomes:

$$\hat{V}_2(x) = \frac{1}{3} * (\hat{V}(x) + 1) + \frac{1}{3} * (\hat{V}(x+1) + 1) + \frac{1}{3} * (\hat{V}(x) + 2)$$

This kind of computation is done for every state (except the final state), for both dices. A more general update rule for states $k \neq d$ is given by the following equations.

$$\begin{cases} \hat{V}_2(k) &= \frac{1}{3} * \Big(f\big(\hat{V}(k)\big) + f\big(\hat{V}(k+1)\big) + f\big(\hat{V}(k+2)\big) + 3 \Big), \\ \hat{V}_1(k) &= \frac{1}{3} * \Big(\hat{V}(k) + \hat{V}(k+1) \Big). \end{cases}$$

Where $f\big(\hat{V}(k)\big)$ is a function handling the traps at state k, and giving back the expected value of the state after the (eventual) triggering of the trap. If the trap is of kind 3, this function also returns an additional cost of $+1$ to simulate the freezing of the player. To lighten the document, the equations do not show how the computation behaves when it has to choose between the *fast* and *slow* path. This function is resumed below, but the reader should take a look at the source code to get the full picture. The function returns for the traps of:

- type 1: $\hat{V}(1)$ since this trap moves the player back to the first state of the game,

- type 2: $max(1, \hat{V}(k-3))$. If $11 \leq k \leq 13$ then it becomes $\hat{V}(k-10)$,
- type 3: $\hat{V}(k) + 1$,
- type 4: $\frac{1}{3} * (\text{type } 1 + \text{type } 2 + \text{type } 3)$.

4 Comparison of theoretical and experimental results

This discussion, and the discussion of the next section will mainly be based on figures 2 to 5. Despite the fact that the same figures are used for both discussions (mainly due to lack of space), they are interpreted in different manner.

This section discusses the comparison between theoretical and experimental results of the *Markov Decision Process*. The program outputs two important values: the optimal strategy (which dice to throw at each state), and the expected number of turns before completing the game (also for each possible state). One could expect the expected value to be close to the true number of turns before winning the game. However, randomness is a big variable in this game, and to get a good approximation of the mean value, many games must be played. The experiments leading to the figures below have been run according to some rules, to minimize the impact of randomness. First, as the figures show, the tested attribute is the number of traps (of a given type) on the map. Maps are generated randomly, to get the largest possible sample for testing the model. Yet, it adds another level of randomness in the program. To deal with it, at each tested indices (i.e. number of traps on the board), the program repeats the following process 40 times: it generates a random map (with the specified number of traps of the tested type) and calls the `markovDecision` algorithm to compute the optimum strategy and the expected result. Then, with this strategy, 600 games are played, and the mean number of turns to complete the game (from the start) of these 600 experiments is stored. Finally, the 40 values of experimental and expected number of turns are averaged, to only two values for this number of traps on the board. This gives more robustness to the computed value.

The figures show that, for each kind of trap, and for each number of traps on the map, the average experimental number of turns before completing the game is really close to the expected value. This is good news: the expected result is a true approximation of the experimental mean value over a large sample of games.

5 Comparison between experimental results of different strategies

The previous section showed that the implemented *Markov Decision Process* algorithm is consistent. Indeed, the theoretical expectation (computed by the algorithm) is quite similar to the results obtained by simulation of the game; and it is the case no matter the kind of trap, nor the value of `circle`.

The second point to show is the efficiency of the *Markov Decision Process* strategy. Indeed, This strategy is supposed to be optimal, and one would expect it to win over all the others. The figures (2 to 5) also show the experimental result of simulation of three other strategies: `Always safe`, which always chooses the first dice, avoiding traps; `Always normal`, which always chooses the second dice, taking the risk of falling into traps, and `Always random`, which randomly chooses between the two dices. The figures do not show the number of turns before completing the game if this number is above 20: as one can see, the `Always safe` and *MDP* strategies are always below this threshold, and one is only interested in the optimal strategy; hence it is useless to check above that threshold.

Firstly, the `Always safe` strategy always gives the same result, no matter the kind nor the number of traps on the board. A more interesting point to notice is that the *MDP* always tends to the same value as `Always safe`, as the number of traps on the map increases. Moreover, when this number is equal to 12, the two strategies give more or less the same result. This is consistent with human understanding of the problem: when the map is full of traps, it is more safe to use the first dice. The only exception regarding this observation is when the map is filled with traps of type 3. This difference could be explained by the fact that this trap only freezes the player for one turn, but does not make him move backwards.

This last points leads to the comparison between the *MDP* and `Always normal` strategies. The value of the attribute `circle` has a big impact on the performances of `Always normal`: this strategy takes the risk of coming back to the beginning if it goes too far. On the other hand, when `circle = False`, and there are no trap on the board, both strategies give the same result. This is consistent: *MDP* will always try to go as fast as possible when there is no danger. Regarding the traps of type 1, 2 and 4, one can see that *MDP* always outperforms `Always normal`. Now considering the traps of type 3, one can see that the two strategies give close results (as discussed before, this type of traps seems to not influence that much the choice of the second dice). However, when the number of traps increases, *MDP* tends to use more and more the safe dice, decreasing the risk of being frozen, and increasing the performance.

Lastly, the `Always frozen` strategy is really bad, as it is most of the time outperformed by the other strategies. This result shows that one should always play *Snake and Ladders* with a minimum strategy, and not always believe in luck.

More generally, the figures show that the implemented *MDP* is always the best strategy, as it always leads to the best value.

6 Conclusion

Throughout this report, the objective was to show the implementation of the *Markov Decision Process* for the game *Snake and Ladders*, and to simulate the game to both prove the correctness of the algorithm, and the performance compared to other strategies. For this last point, very basic strategies were used to show the efficiency of *MDP*. A further improvement of this work would be precisely to implement better strategies to be experimentally compared with the *Markov Deicison Process*. However, as the optimality of *MDP* has theoretically been proven, this work should lead to the same results as those in this paper.

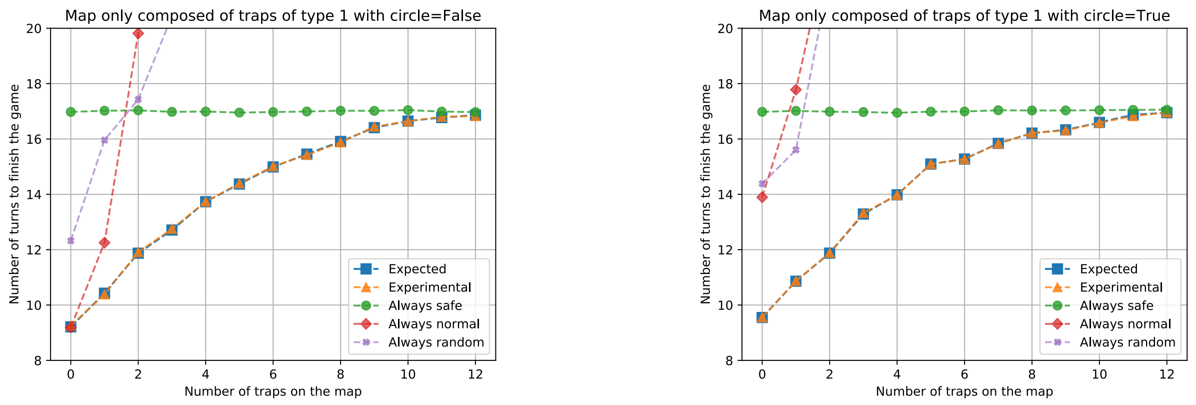


Figure 2: Evolution of the number of turns to complete the game from the first state of the board, when the number of traps of type 1 increases. On the left, the `circle` value is set to `False`, and set to `True` on the right

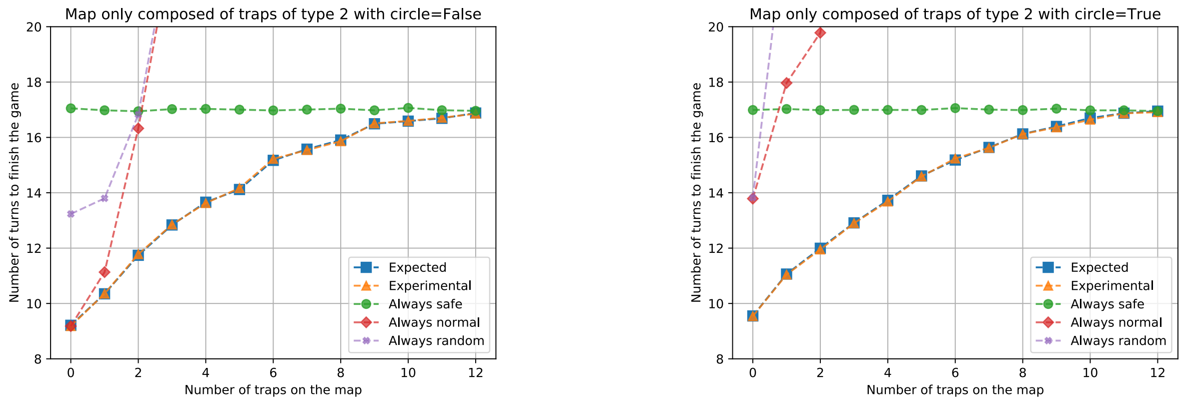


Figure 3: Evolution of the number of turns to complete the game from the first state of the board, when the number of traps of type 2 increases. On the left, the `circle` value is set to `False`, and set to `True` on the right

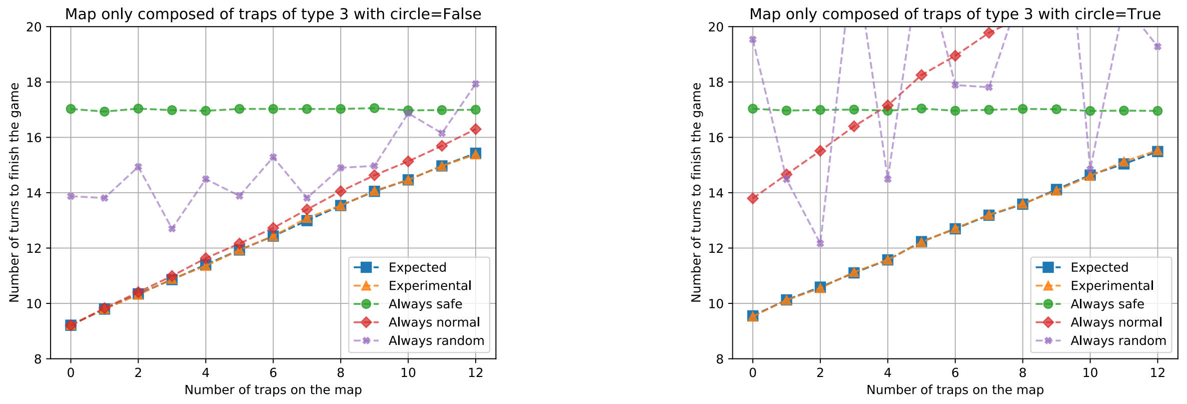
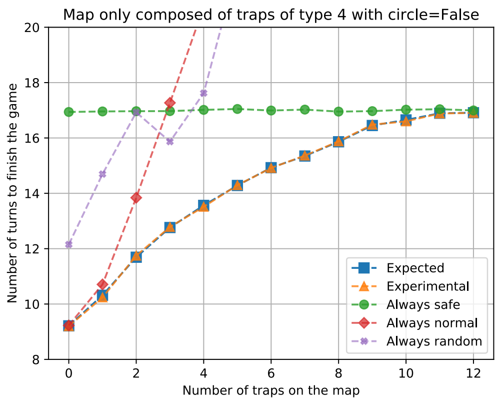


Figure 4: Evolution of the number of turns to complete the game from the first state of the board, when the number of traps of type 3 increases. On the left, the `circle` value is set to `False`, and set to `True` on the right



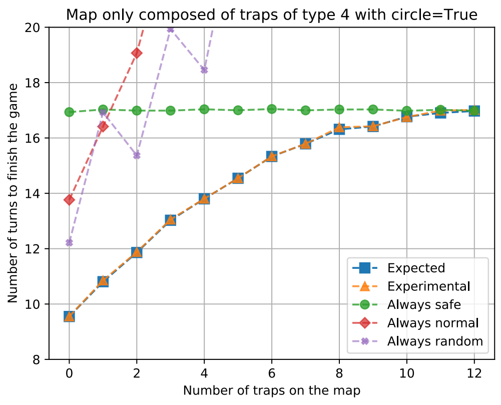


Figure 5: Evolution of the number of turns to complete the game from the first state of the board, when the number of traps of type 4 increases. On the left, the `circle` value is set to `False`, and set to `True` on the right