Learning to play Monopoly: A Reinforcement Learning approach

Panagiotis Bailis and **Anestis Fachantidis** and **Ioannis Vlahavas** [1]

Abstract. Reinforcement Learning is a rather popular machine learning paradigm which relies on an agent interacting with an environment and learning through trial and error to maximize the cummulative sum of rewards received by it. In this paper, we are proposing a novel representation of the famous board game Monopoly as a Markov Decision Process and a Reinforcement Learning agent capable of playing and learning winning strategies. The conclusions drawn from the experiments are particularly positive, since the proposed agent demonstrated intelligent behavior and high win rates against different types of agent-players.

1 Introduction

In recent years, *Reinforcement Learning* (RL) [5] has found application in a variety of fields, including video and board games [6]. Reinforcement Learning is a Machine Learning paradigm in which an agent learns an action policy in a sequential decision problem by maximizing a scalar reward received from the environment. Monopoly is one of the most famous board games, played by millions of people worldwide and was originally designed to illustrate an economic principle, namely the Georgist concept of a single land value tax. Despite the all-time popularity of the game, to the best of our knowledge, an RL approach for agents learning to play Monopoly, has not been discussed (there are some older attempts to model Monopoly as Markov Process including [1]). Monopoly representation as a *Markov decision Process* (MDP) poses a series of challenging problems such as the large state space size and a highly stochastic transition function.

This work uses the RL approach of modelling the Monopoly board game as a MDP allowing RL agents to play and learn winning strategies. We implement the corresponding environment and a basic Q-Learning agent for it. Moreover we compare its performance with two -baseline- agents, one using a random policy and one with a fixed, rule-based policy.

The main contributions of this work are i) a novel MDP representation and modelling of the game; ii) the implementation of the corresponding Reinforcement Learning environment for the Monopoly game, suitable for RL agents and finally; iii) a Q-Learning RL agent for the game.

2 Background

2.1 Reinforcement Learning

Reinforcement Learning (RL) addresses the problem of how an agent can learn a behaviour through trial-and-error interactions with a dynamic environment [5]. In a RL task the agent, at each time step t, receives an observation of the environment's state, $s_t \in S$, where S is the set of possible states, and selects an action $a_t \in A(s_t)$ to execute, where $A(s_t)$ is the finite set of possible actions in state s_t. The agent then receives a reward, $r_{t+1} \in \Re$, and moves to a new state s_{t+1}. The general goal of the agent is to maximize the expected return, where the return, R_t, is a function of the reward sequence.

A *policy* function $\pi(s, a)$, expresses the probability of taking action a in state s. For any policy π, the *state-value function*, $V^\pi(s)$, denotes the expected discounted return, if the agent starts from s and follows policy π thereafter.

Similarly, the *action-value function*, $Q^\pi(s, a)$, under a policy π is defined as the expected discounted return for executing a in state s and thereafter following π.

The optimal policy, π^*, is the one that maximizes the value, $V^\pi(s)$, for all states s, or the state-action value, $Q^\pi(s, a)$, for all state-action pairs.

A widely used algorithm for finding the optimal policy is the Q-learning algorithm [7] which approximates the Q^* function with the following form:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)).$$

3 Monopoly as a Reinforcement Learning Task

In the following, we attempt to model Monopoly as a single-agent RL task. Despite the non-stationarity of the multi-player Monopoly game invalidating most of the single-agent RL theoretical guarantees, single-agent RL algorithms have been extensively used in the literature in natively multi-agent settings (e.g. Q-Learning) [2, 4] with successful results and are considered suitable for a first approach on modelling Monopoly as a RL task.

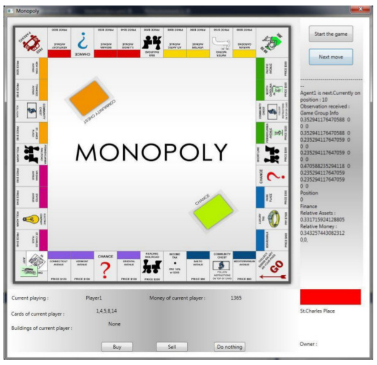
3.1 Modeling Monopoly as an MDP

3.1.1 State Space

To represent Monopoly as a MDP we first represent the full set of knowledge a real human player would have, as the (observed) state of the agent. We formulate the state s_t as a 3-dimensional vector of objects containing information about the game's *area, position* and *finance* current status at time t.

The *area* object, contains information about the game's properties, meaning the properties possessed from the current player and his opponents at time t. More specifically, it is a 10×2 matrix where the first column represents the agent-player, the second the rest of its opponents and each row corresponds to one of the game's colour-groups (8 property groups, the group of all utilities and the group of all rail-roads) . Each element (x,y) specifies the percentage that the player y owns from group x, where value 0 indicates that the player does not own anything and 1 that at least one hotel has been built.

[1] Department of Informatics, Aristotle University of Thessaloniki, Greece, email: {pmpailis, afa, vlahavas}@csd.auth.gr



```
--
Agent1 is next.Currently on
position : 10
Observation received :
Game Group Info
0.352941176470588  0
0  0
0.352941176470588  0
0.235294117647059  0
0  0
0.235294117647059  0
0  0
0.470588235294118  0
0.235294117647059
0.235294117647059
0  0
Position
0
Finance
Relative Assets :
0.331715924128805
Relative Money :
0.343257443082312
0,0,
```

Current playing : Player1
Money of current player : 1365
Cards of current player : 1,4,5,8,14
Buildings of current player : None
St.Charles Place
Owner :

Figure 1. Graphical User Interface for Monopoly displaying detailed information for the state variable values, also providing the basic framework for real human players to participate

More specifically, the domain of the parameter x can be described as follows :

- 0, player does not own anything from the current group
- $0 < x < \frac{12}{17}$ player possesses at least one property of the current group
- $\frac{12}{17}$ player has all the properties of the current group but has not built anything
- $\frac{12}{17} < x \leq 1$ player has built at least one house in at least one of the current group's properties

where 12 being the least common multiple of 2,3 and 4 (the number of properties a monopoly colour group can have) representing the full ownership of all the properties in a group. Adding up to that 5, the maximum built houses on a property, we obtain 17 as the denominator of the threshold.

The *position* variable determines the player's current position on the board in relation to its colour-group, scaled to [0,1] e.g. If the player is in the fourth colour group this value would be 0.4.

The *finance* vector consists of 2 values, specifying the current player's number of properties in comparison to those of his opponents' as well as a calculation of his current amount of money. Specifically concerning the first value, given players a, b and c and the number of properties they own as p_a, p_b, p_c it will be $\frac{p_a}{p_a+p_b+p_c}$. For the second value, since the maximum amount of money owned by a player, x, varies significantly, the corresponding variable is transformed to a bounded one with the use of the sigmoid function $f(x) = \frac{x}{1+|x|}$.

A sample state s at a given time t would be as follows :

$s_t = \{0.3529, 0, 0, 0.2352, 0, 0, ..., 0, 0.6543, 0.3319, 0.3432\}$

where the first 20 values represent the area vector, the next one the player's position on board and the last 2 the player's current financial status. In this specific example, the player has completed 0.3529% of the maximum available development (building) in area 0, and his current financial status is equal to 0.3432.

3.1.2 *Action Space*

The action set A available in the proposed Monopoly RL environment is applicable to a group of properties. Specifically each action is a vector whose elements represent an action for each of the 10 colour-groups of the game from the following:

- *Spend*: The spending action can be described as every possible action that decreases the player's money, with the following order: building a hotel or house, unmortgaging a property and last, buying an unowned property (if the other two are not applicable).
- *Sell*: This action, drives the player to find a source to earn more money and can be accomplished either by (in order of preference) selling a hotel, a house or mortgaging an already owned property.
- *Do Nothing*: Do not take any action for the specific property group.

Algorithm 1 Bidding algorithm

```
1: if The agent's current position is a property without an owner then
2:     Initialize m , the amount of markup
3:     while More than one player wants to buy this property do
4:         $bid = property\_value \times m$
5:         Check every player's action for current bid
6:         $m = m + 0.2$
7:     end while
8: end if
9: If there is only one player left declare him winner of the bid
```

For both actions the order in which the actual behaviours are chosen can be considered a form of knowledge injection in the process. However the specified orderings are considered natural for Monopoly since i.e. when a human player seeks to earn money he will first sell buildings and as a last resort will mortgage a property .

3.1.3 *Reward Model*

The reward signal of the environment is designed so that its output is bounded, provides discriminant ability between positive and negative pairs of states-actions and considers most of the information available to the players. Therefore, a sigmoid function as described previously, is considered suitable since its output values are bounded in [-1,1]. The proposed reward model is represented by the following equation :

$$r = \frac{\frac{v}{p} * c}{1 + |\frac{v}{p} * c|} + \frac{1}{p} * m,$$

where:

- *p* is the number of players
- *c* is a smoothing factor
- *v* is a quantity representing the player's total assets value and is calculated by adding the value of all the properties in the possession of the player, minus the properties of all of his opponents
- *m* is a quantity representing the player's finance and is equal to the percentage of the money the player has to the sum of all players' money

We should emphasize here that v and m are values computed each time the agent interacts with the environment and represent the agent's financial status considering all of the other players as well.

From the reward signal equation presented here it can be inferred that the reward signal attributes more importance to the total property than to the players' money since in Monopoly, the amount of money a player has may vary significantly due to random or uncontrolled events (such as decision cards).

3.2 Bidding and Distance Metrics for Monopoly

Following the original rules of Monopoly [2], a bidding contest was designed and implemented within the proposed environment. The bidding process, takes place every time a player lands on an unowned

[2] The special *get out jail* cards have been removed from the game in order to simplify the case where a player is send to prison.

property and does not decide to buy it. The algorithm that implements this procedure is described in Algorithm 1.

Since the state variables, take values from a continuous space there is a significant need for a state similarity metric that neglects slight state differences, assisting therefore the value function approximation. The scheme we propose in this paper uses threshold values for the 3 state object described earlier in this text, to determine whether two different states could be considered as the same. More specifically two states s_1 and s_2 can be considered as *equal* when all of the following statements are true:

- $|(area[0,j]_{s_1} - area[0,j]_{s_2}) + (area[1,j]_{s_1} - area[1,j]_{s_2})| \leq 0.1, 0 \leq j \leq 9$
- $|finance_{s_1} - finance_{s_2}| \leq 0.1$
- $|position_{s_1} - position_{s_2}| = 0$

4 Experiments and Results

In order to evaluate the proposed representation and environment, three types of players were implemented[3]. First, a *random* player whose actions are selected randomly ignoring the state signal. A *fixed-policy* player whose action selection is based on the money he possesses. He sells if he has less than 150, buys when he has more than 350 and does nothing between. These thresholds were tuned appropriately for his best performance. Finally an RL agent with action selection based on the ε-*greedy* algorithm. For the visualization of the experiments a simple Graphical User Interface for Monopoly was developed (see Figure 3), displaying the state variables detailed information and also providing the basic framework for real human-players to participate. In the following we describe the series of experiments conducted in the proposed novel Monopoly RL environment.

4.1 Random vs Fixed Policy

We first compare the two baseline players discussed above. From the experiments it was clearly shown that a player with even a simple empirical policy can have almost a perfect win ratio against a random agent. Specifically the win ratio in 1000 games was 97% to 3% in favor of the fixed-policy agent, rendering him a suitable opponent for the RL agent.

4.2 Training a RL agent

This experiment represents the main training procedure, and assisted on tuning the parameters of the learning algorithm. The modelling of the agent is based on [5] and relies on the use of the Q(λ)-learning algorithm [3]. Since the state space is continuous, it necessitates the need for function approximation. A 3-layer back propagation neural network was used for that purpose, estimating the Q values for each state experienced. Agents with different number of neurons in the hidden layer were tested in order to find the best possible equilibrium between efficiency and computational cost.

After conducting several experiments to tune the RL agent, his best performance was achieved using the following parameters : learning rate a = 0.2,$\gamma = 0.95$, the NN learning rate $\alpha = 0.2$ and $\lambda = 0.85$.

In Fig. 2 we can see the avg. performance of 25 trials for 1000 episodes with the Q-Learning agent training against a random policy agent. After 1000 episodes the RL agent shows 100% win ratio

[3] The implementation of the Monopoly environment and the agents are available at: https://github.com/pmpailis/rlmonopoly

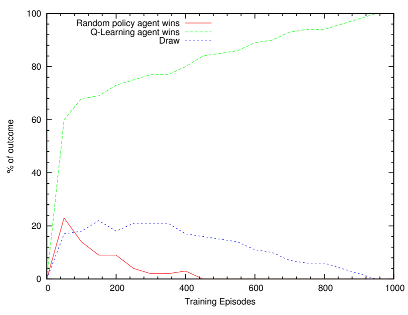


Figure 2. Win percentage of RL agents trained for a different number of episodes versus a random player

4.3 RL Agent vs Fixed Policy vs Random Policy

Finally, the trained agent's performance was compared to that of the fixed-policy and random policy players discussed earlier. The results displayed in Table 1 , show that the RL agent performs significantly better than a player whose decision making process is based solely on a fixed empirical policy with an avg. win percentage of 69.4%. The RL agent demonstrated the ability to change his strategy and appropriately react to rapid changes in the game's environment in order to win.

Player	Number of games won	Percentage
Random	20	2%
Fixed Policy	286	28.6%
RL agent	694	69.4%
Total	1000	100%

Table 1. Number of wins for each agent over 1000 games

5 Conclusions and Future work

A novel reinforcement learning approach for the Monopoly game was presented. The results of the experiments indicate that the proposed representation was successful allowing a simple RL agent accompanied with a NN function approximator to learn winning strategies against a fixed policy player and a random one. The RL agent demonstrated several times an intelligent behaviour of sacrificing some of his temporary wealth in order to invest and secure a more prosperous future. Future work includes adopting a multi-agent approach for the learning algorithms of the RL agents and a performance comparison to existing Monopoly video games and/or human players.

REFERENCES

[1] Robert B Ash and Richard L Bishop, 'Monopoly as a markov process', *Mathematics Magazine*, **45**(1), 26–29, (1972).
[2] AG Barto and RH Crites, 'Improving elevator performance using reinforcement learning', *Advances in neural information processing systems*, **8**, 1017–1023, (1996).
[3] Jing Peng and Ronald J Williams, 'Incremental multi-step q-learning', *Machine Learning*, **22**(1-3), 283–290, (1996).
[4] Sandip Sen, Ip Sen, Mahendra Sekaran, and John Hale, 'Learning to coordinate without sharing information', in *In Proceedings of the Twelfth National Conference on Artificial Intelligence*, pp. 426–431, (1994).
[5] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning, An Introduction*, MIT Press, 1998.
[6] Nees Jan van Eck and Michiel van Wezel, 'Application of reinforcement learning to the game of othello', *Computers & Operations Research*, **35**(6), 1999–2017, (2008).
[7] Christopher JCH Watkins and Peter Dayan, 'Q-learning', *Machine learning*, **8**(3), 279–292, (1992).