An artificial intelligent player trained based on reinforcement learning using Q-learning for the purpose of playing LUDO

Keerthikan Ratnarajah

University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark
Kerat12@student.sdu.dk

Abstract. This paper will present a method for developing a artifical intelligent player for the game Ludo. The player was trained using a reinforcement learning method named Q-learning. To apply the method to the game, was the game divided into a certain number of states and actions, which the player during game time could reside in. The player covered in this paper is then compared against two other players. One of these players has been trained in a supervised manner, in which a feedforward neural network determines the action it has to perform, and the last one is a further improvement of the first one using genetic algorithm.

1 Introduction

This report will entail information about the implementation of an artificial intelligent (AI) player trained using a reinforcement learning, more specifically Q-learning. The tools used in this paper, and to develop the player, are tools from the AI2 course held by Associate professor Poramate Manoonpong at the University of Southern Denmark. The trained AI player will be tested against two different players developed by Jes jepsen and Kristoffer Honor. One of the competing AI players is based on feedforward based neural network trained with back propagation, and the second AI player is based on the previous AI player improved with an genetics algorithm applied on to it. The conclusion of this paper will state who will be the victor of this game.

2 Method

The AI player developed in this paper is trained using Q-learning[1]. This method was chosen as it easy to implement, and is very suitable for this type of problems, in which specific world model is not available. The method requires the ludo game to be divided into possible states and possible action. The possible states were defined as such:

Table 1. The different states and how they are represented in the program, and which row in the Q-table they can be read from

Binary string	States	Row in Q-table
1 0 0 0 0 0 0	"In home"	1
0 1 0 0 0 0 0	"On globe"	2
0 0 1 0 0 0 0	"On a star"	3
0 0 0 1 0 0 0	"In Goal"	4
0 0 0 0 1 0 0	"On the winner road"	5
0 0 0 0 0 1 0	"In safety with same colored token"	6
0 0 0 0 0 0 1	"On freespace"	7

Table 2. The different actions and how they are represented in the program,and which column in the Q-table they can be read from

Binary string	Actions	Column in Q-table
1 0 0 0 0 0 0 0 0	"Out from home"	1
0 1 0 0 0 0 0 0 0	"Get into goal"	2
0 0 0 1 0 0 0 0 0	"Move to globe"	3
0 0 0 0 1 0 0 0 0	"Move to star"	4
0 1 0 0 1 0 0 0 0	"Move to goal via star"	5
0 0 0 0 0 1 0 0 0	"Get into safety with same colored token"	6
0 0 0 0 0 0 1 0 0	"Get into the winner road"	7
0 0 0 0 0 0 0 0 1	"Commit suicide"	8
0 0 0 0 0 0 0 1 0	"Kill opponent"	9
0 0 1 0 0 0 0 0 0	"Just move"	10
0 0 0 0 0 0 0 0 0	"No move possible"	11

For each states were 11 actions as seen in table 2. The states are detected by comparing the position of the token, relative to the game board, and thereby determine where in the game the token resides. The different states and their relative position can be seen in figure 1. The possible actions is found by adding the dice value on to the current position of the AI player token, and thereby can it be determined what "action" would be performed if the token performed it, by looking at all the position of all tokens.

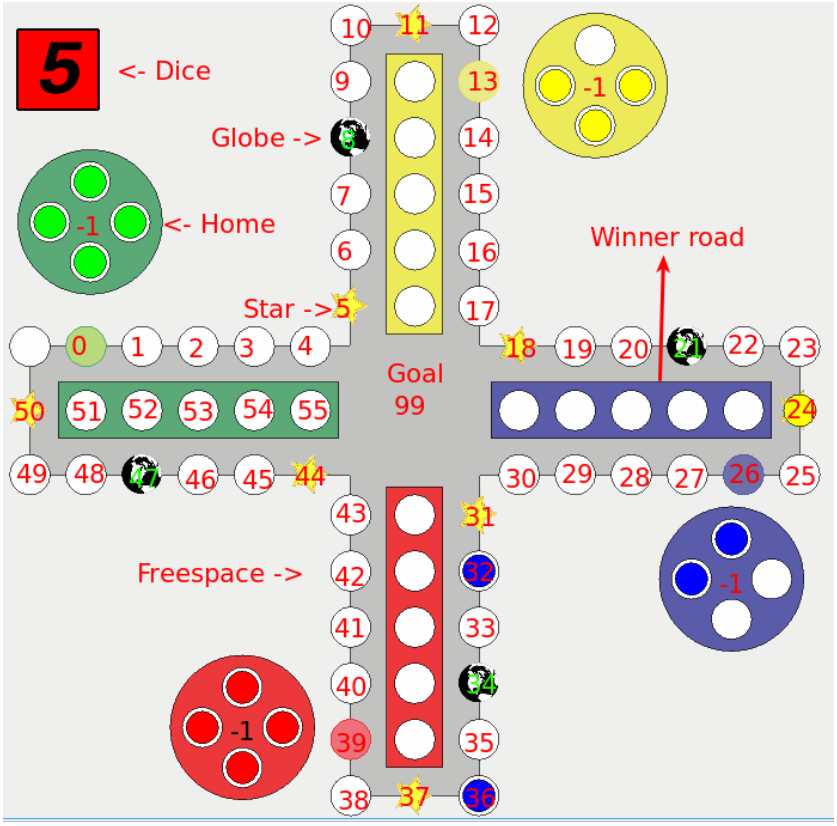

Fig. 1. Illustration of the game board and the position number of the game board seen from player 1 (Player green).

The immediate reward is given based on the position of the token piece that was played. This enforces the strategy that it is desired that the tokens move forward in the game, rather staying in the same position. When the goal state has been reached will a bonus reward of 100 be added additionally to the position based reward, to enforce the strategy that moving to the goal position is desired. A minor punishment is given if the player does not move forward, and chooses to not do any move, unless the token is at home position, and forced to stay in home. The punishment consist of subtracting 0.3 from the computed reward. This punishment was partly added to learn the player that doing nothing is not the best choice, as the positional reward is higher by moving forward, but if no other possible actions is available, or if the movement of token lead it being killed and sent back home, will the punishment be even greater, and thus will the right choice sometimes be not to move the token which is farthest in the game. The minor punishment is intended to be a "friendly reminder" that staying in the same position is not good, but is allowed for a certain number

of turns, in which the token should be in a state where moving will not lead to suicide. If a token is in the goal state, will it not be considered as movable, and ignored the rest of the game.

The system is trained using ϵ - greedy[2] such that the rate of exploration and exploitation is controllable [3]. The training is done in stages, with different ratios of exploration and exploitation. After each stage is a test performed to see how well the Q - table performs against the 3 random players, and based on the result will it be determined whether more exploitation or exploration is needed. While training will the discount factor (γ) be set to a low value, thus making it only consider immediate reward, and not the long term. This is was chosen as this system has many different valued rewards, and each of them provide the Q-table some form of knowledge where the token is in the game, and update it based on that. A higher discount factor would be ideal if the system had fewer rewards. The learning rate (α) will be changed according to result from the test. A higher learning rate would make the system learn based on new information, but might have a hard time converge to the ideal values, and vice versa would a lower learning rate make it converge slowly to the ideal values.

The player generated with Q-learning competed against two other players, one based on a feed forward neural network, trained with back-propagation, and the same network with a genetic algorithm applied on to it. Different for the method used here are the neural network trained in a supervised manner, in which it has been trained to follow a certain strategy set by the user, unlike Q-learning which only been given a finite number of states it can detect and action it can perform based on the state it in, and thereby guided by reward find an optimal strategy.

3 Result and discussion

The first training will consist of full exploration, with a high learning rate. These were chosen such that the player would get an idea of how the world is, and get an rough estimate of how the Q-table should be.

The Q-table is initialized with all its entries being zero. First training consist of $\alpha = 0.7$, $\gamma = 0.1$ and $\epsilon = 1$, for 4000. The training returns this table 3

Table 3. First Table generated with $\alpha = 0.7$, $\gamma = 0.01$ and $\epsilon = 1$

3.03773	0	0	0	0	0	0	0	0	0	-1.90448
0	9.2829	5.43241	10.0345	76.9815	3.65767	31.5851	2.4933	10.5403	4.36399	0
0	111.303	13.776	23.4981	193.999	15.895	30.3937	2.28143	23.2157	3.26186	0
0	0	0	0	0	0	0	0	0	0	0
0	37.6306	0	0	0	0	44.899	30.6251	0.286614	36.4623	0
0	125.997	17.1916	13.6513	153.59	4.73704	52.9153	1.61601	7.97247	16.0524	0
0	81.7465	10.2539	21.6026	22.0395	12.678	33.885	1.66179	18.3656	14.6915	0

Table 3 shows that the player quickly learns that moving forward is an attractive move. It prioritize moving a token forward in the freespace higher than a piece in the home section, thus following the strategy of moving a token to goal.

The player was then tested against an 3 random players for 4000 plays, from which the result can be seen in figure 2.

It was decided by the author that the system would be tested based on how many tokens is in goal, when a winner has been declared. This does not explicitly state who would have become the runner up and so on, but provides an idea of the how far the runner up has been compared to the other players, such that one would be able to argue whether the loss was significanoutperformst or not. The result of the test shows that the AI player wins 22 % of the time, against random players, but still loses overall against them. It seems that the player has a hard time moving the last piece into the goal, which can clearly be seen in figure 2 as the player most of the time has 3 tokens in goal, compared to the others.

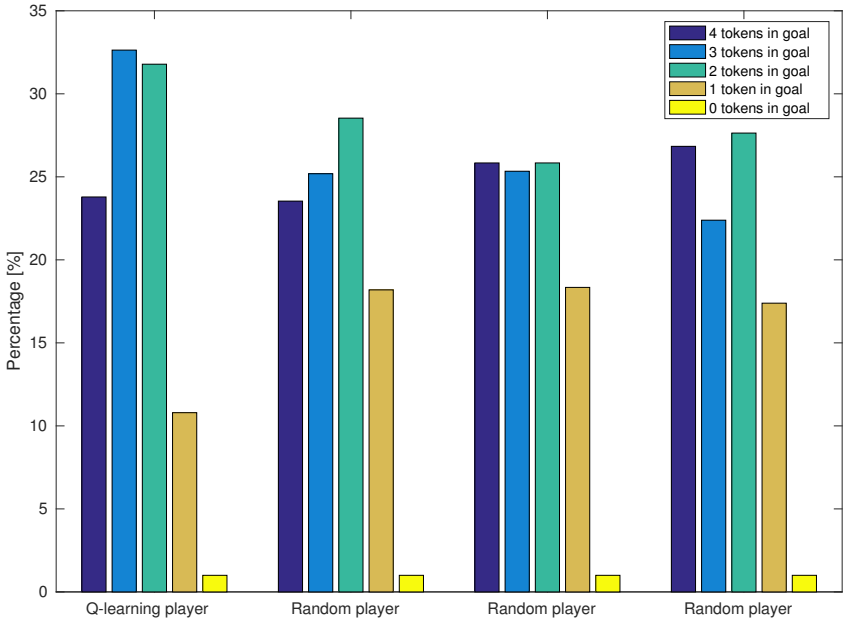

Fig. 2. Testing AI player against 3 random player, after initial training

One of the reason why the player might perform poorly against these random players could be due to the player not be able to determine when it should be aggressive or defensive, as none of the state explicitly states when it should one or the other. One way of countering this, using the already defined states, would be training with a combination of exploration and exploitation, such that the player also would receive punishment when the most desirable action is chosen, and thus adjust the value based on how likely an defensive strategy or aggressive strategy should be used. The next training session will therefore consist of the being 70% exploitation and 30% random. The training and testing was done for 2000 iterations each. To avoid the table being changed completely was it decided to lower the learning rate to 0.5. The returned table by the training can be seen here 4.

Table 4. Table generated after second training with 70% exploitation and 30% exploration, $\alpha = 0.5$, $\gamma = 0.1$ and $\epsilon = 0.3$

49.5981	0	0	0	0	0	0	0	0	0	-1.3
0	197.902	12.1233	31.3932	189.812	11.5276	48.3324	24.5621	30.0304	20.6168	0
0	195.509	18.029	31.4243	197.992	17.6874	55.1491	2.1486	28.9567	40.2133	0
0	0	0	0	0	0	0	0	0	0	0
0	197.997	0	0	0	0	54.9937	55.3389	53.6452	52.6392	0
0	195.749	33.8193	11.9056	153.59	4.90463	29.9821	4.20344	11.8583	3.78396	0
0	123.498	28.1728	43.5418	113.497	6.8573	54.6788	4.6489	11.2915	26.564	0

It can be seen by comparing table 3 and table 4, that some of the actions has become more defined, and a sense intuitive strategy seem to be learned. The system prioritizes that moving forward highest, and if the possibility of moving forward by jumping from star to star occurs will that option be chosen. Another huge improvement is that the system does not consider committing suicide as often as it would before when the token was is in freespace, preventing it from winning the game. The table does also highlight some implementation flaws, as some actions seem to be possible, although it is not possible in the game. An example would be to kill an opponent when the token is on the winner road, or commit suicide on the winner road, two actions which is not possible, but are highly prioritized as they move token forward, thus leading to a higher win rate.

The player was again tested against 3 random players, which resulted in performance boost in a win rate from 22% to 48% as seen in figure 3

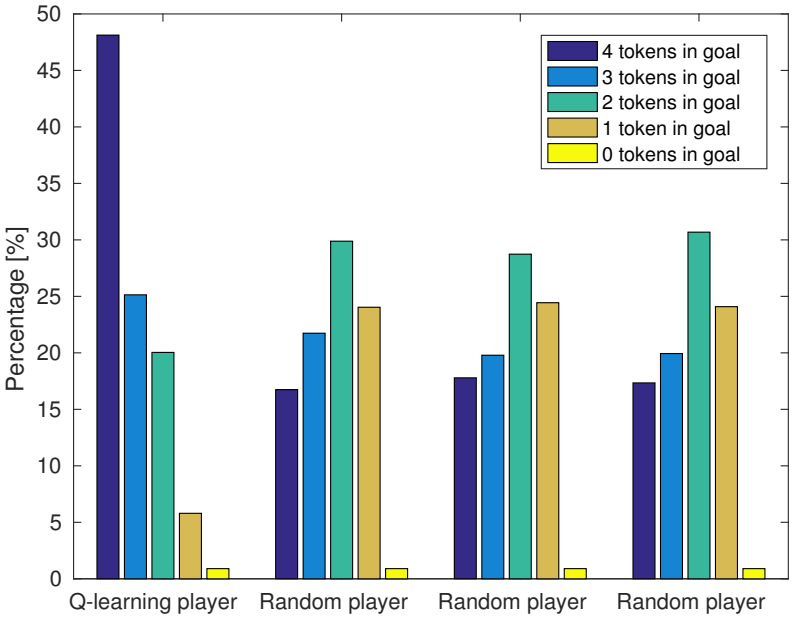

Fig. 3. Testing AI player against 3 random player, after second training

It was decided after the last perfomace test, to test the system against the other AI players trained by Jes Jepsen and his neural network based player and Kristoffer Honor and his player based on Jes Jepsen neural network based player with a genetic algorithm applied onto it. 4000 games was played from which these results was logged. The result of the test can be seen in figure 4.

The final test shows that the player trained using Q-learning won overall, which was also expected. The win rate of the player trained with NNBP has a win-rate of approx. 34% and the player trained with Q-learning has a win-rate approx. at 38%. The player trained using neural network, was trained in a supervised manner. The network was trained to mimic a certain strategy, which made it vulnerable against other strategies, in contrast to the player trained with Q-learning which only has a finite states and actions and positional based reward which devised a strategy more flexible towards different situations. It was expected that the player improved with a genetic algorithm would have outperformed all the other player, but due to time and problem with implementation was it not possible to train a suitable player with a genetic algorithm, which is why figure 4 shows the player having poor performance related to having most tokens in goal when a winner has been declared.

The win-rate would theoretically have been better if the AI player was trained against these players , but was not tested due to time.

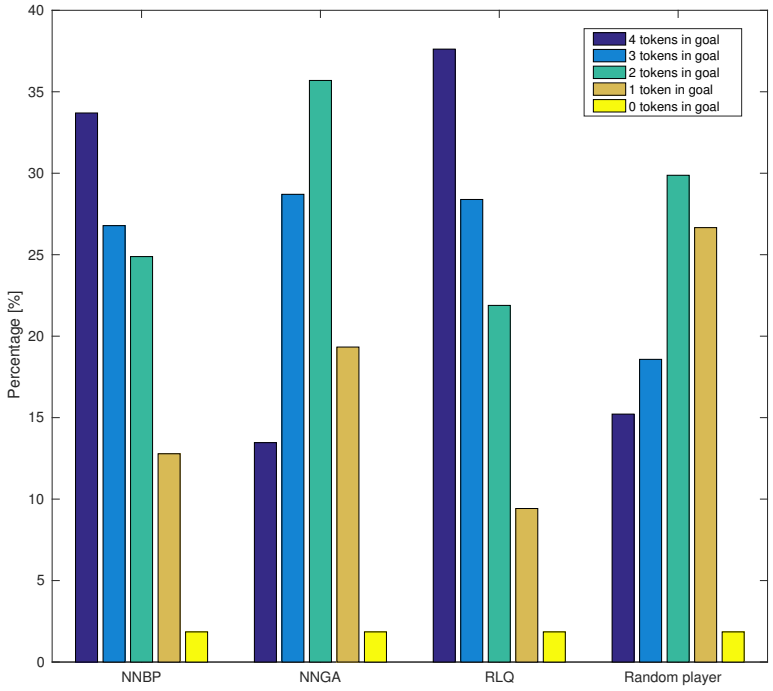

Fig. 4. Testing AI player trained with Q - learning (RLQ) against a player based on feedforward neural network trained with back propagation (NNBP), and with a neural network based player improved by genetic algorithm (NNGA).

4 Future work

Beside fixing some of the bugs discussed section 3, were other improvements thought of while implementing this AI player, but not tested due to time. One of these improvement would be adding an additional state which checks whether the player is on a opponent globe, rather than checking for globes in general. It was prior to the implementation not known by the author, that being on a opponent globe could cause it being killed if the opponent move one of his tokens out from home, which might have caused confusion on the globe state, and thus not be able to properly settle. In general would it have been better adding more states, and actions for other cases, to ensure that all cases will be interpreted

uniquely rather than fitting them into a finite number of states determined by reading some specific signals.

In section 3 were the training performed with a fixed discount factor on purpose, as the reward given here is based on the current position of the token, is provided as a reward. Keeping the discount factor low, result the update phase in the Q-learning being short sighted, which here is beneficial for the type of reward chosen. It would have been ideal testing different values to see if the assumption was true.

It was also thought whether it would have been beneficial to train the player using softmax[4] or other types of exploration vs. exploitation algorithm and see whether it would make it better. Problem with ϵ - greedy is that prioritizes all actions equivalently besides the highest ranking action. So if two actions has both were to be rather promising, would it only explore the most promising rather the next promising, as it would only be choose it as often as the worst possible action. One way of remedy this would be to use softmax and select an action based on a probability of a Q[s,a], to give it greater probability of being chosen.

Future work would also applying genetic algorithm to the Q-learned player, and see how much improvement could have been made.

5 Conclusion

An AI player was trained using reinforcement learning. The AI player was trained using a method called Q-learning. To train the system using Q-learning was a finite set of states and actions determining where the AI player was in the game, and what it could do. The AI player was trained using an ϵ - greedy method, such that the exploration rate and exploitation rate was controllable while training the player. As the player began to show promising result, competed it against the other AI player, which showed that the player trained with Q-learning won most often compared to the other players, but not a win definitely as the change in win-rate between the winner and runner up was around 4%.

6 Acknowledgement

The author of this paper would like to thank Poramate Manoonpong, Associate professor at SDU, for his inspirational lectures within the field of AI, and providing us the tools needed to completing this assignment.

The author would also like to thank Jes Jepsen and Kristoffer Honor developing two different ludo players with the help of Neural network, and improving

it with the help of Genetic algorithms.

Lastly would the author like to thank Nikolaj Iversen and Nicolai Lynnerup for developing the LUDO simulator for which players was tested on, and trained on.

References

1. Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.
2. Michel Tokic. Adaptive ε-greedy exploration in reinforcement learning based on value differences. In *KI 2010: Advances in Artificial Intelligence*, pages 203–210. Springer, 2010.
3. Melanie Coggan. Exploration and exploitation in reinforcement learning. *Research supervised by Prof. Doina Precup, CRA-W DMP Project at McGill University*, 2004.
4. Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 1998.