Q-Learning - Ludo

Jens Otto Hee Iversen
jeive17@student.sdu.dk

University of Southern Denmark

Abstract. This paper explores the possibilities of using Q-Learning to teach an AI agent to play the board game Ludo. A complex, compact and generalized state representation is proposed together with a reward function. Three agents were trained with varying reward function parameters and compared to each other and an agent trained using genetic algorithm (GA). Results show that the three agents gets a win rate of 47.2%, 42.4% and 50.5% then evaluated over 20.000 games. The agents could not outperform the GA agent who had a win rate of 75.27%, but where still able to outperform non-intelligent agents.

1 Introduction

Artificial agents have been used to solve many different board games that have been around for years. A classic examples is chess that have been conquered by the artificial intelligence (AI) AlphaZero [2] or the board game Go that that AlphaZero and AlphaGo [1] also have conquered. Ludo [5] is another board game and is the focus in this paper. Ludo is 2-4 player game where each player has four pieces. It is a race game, where the objective is to be the first player to get all four pieces around the linear map and into the goal. The pieces moves according to the dice that is rolled each turn, making the game non-deterministic. Ludo has an extremely high state-space complexity estimated to be approximately 10^{22} [3]. This makes it very unlikely that a perfect representation of all possible state of the board could be made. It also unlikely that a perfect evaluation function can be constructed. Q-Learning is the method used in this paper and aims to tackle this high complexity, with a complex, compact and generalized state representation of the game. Q-Learning is trained over a high number of games and its performance is compared with a similar agent that is modelled after GA.

All implementation is done in Python and the source code can be found at GitHub [4].

2 Methods

Section 2.1 explains how the state representation have been design and the functionality of it. Section 2.2 explains the reward functions and how it is used to punish and rewards the agents. Section 2.3 explains how the agent see's the

game, what rules that are used and how the players chooses move. It also explains the modification to the Q-Learning algorithm. Section 2.4 explains a GA agent trained on ludo. This will be used as comparison to the method presented in this paper.

2.1 State representation

Multiple state representation was proposed. The first one took all 16 pieces into account by using 6 bits for each piece to represent the location on the board. Only 6 bits is needed since the there is 60 squares on the board. The idea here was to concatenate all the bits for the pieces together and use this single number to represent a single state. The problem here is that the the number of possible combination for all 16 pieces is extremely high and is even hard to calculate in theory, making it almost impossible to implement. Therefore was this representation discarded.

Therefore was another representation proposed. This representation proposes that instead of keeping track of the location for each piece, it keeps track of general information for the agents pieces. The idea here is that the information can be applied to multiple squares on the board. The representation consist of things that can happen due to the special squares (globes, stars, etc) or the possibilities that occur due to other pieces being near the agents pieces. There was proposed six sub-states per piece and they can be seen in table 1.

Home	Safe	Vulnerable	Attacking	Finish Line	Finish

Table 1: The six sub-states per piece.

Home tells if the piece is on its starting square. Safe tells if the piece is either standing on a globe or if its occupying a square that a friendly piece is also occupying. Vulnerable tells if an enemy's piece is within 1-6 squares behind the piece, but only if the agents piece is not safe. Attacking tells if there is an enemy's piece within 1-6 squares in front of the piece, but only if enemy's piece is not safe. Finish Line tells if the piece is standing on last six squares before the goal. Finish tells if the pieces has reach the goal.

This representation is for one piece and is used for all of the four pieces that the agent can control. The representation is done by using bits, meaning that each piece has 6 bits one for each sub-state. A bit being 0 means that it is false and a bit being 1 means that it is true. The agent has a total of $6 \cdot 4 = 24$ bits to represent its four pieces. On top of this the dice is also used in the state representation. So for the final state representation a 24 bit number and the dice is used. This gives the state for the Q-table. The action space has a maximum size of four and determines which piece to move. The number of actions will depend on the number of legal piece that can be moved. So for a given state it can be between zero and four.

To make the understanding of the state representation easier a state representation is derived from a Ludo state which can be seen in figure 1.

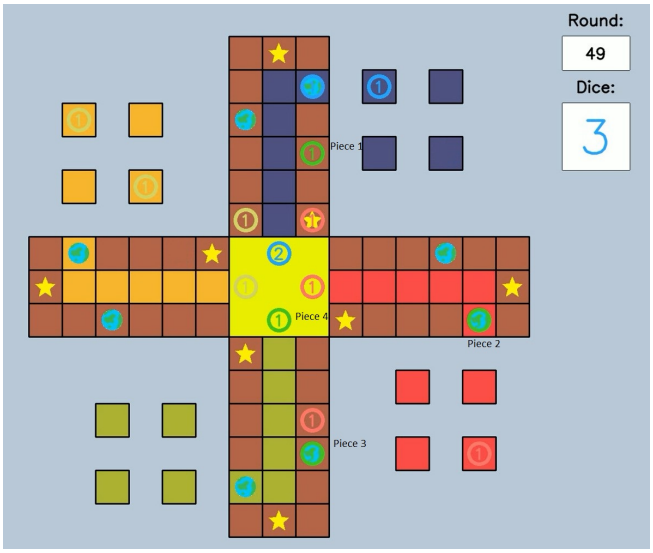

Fig. 1: An arbitrary state of a Ludo game.

Here there four sub-state for the green player would be

Piece 1 - 001100
Piece 2 - 011000
Piece 3 - 010000
Piece 4 - 000001

It can be seen that this state representation does not work perfectly, since its saying piece 1 is attacking the red players piece that is standing on the star. If piece 1 lands on this square it will not hit the enemy's piece home, but it would instead move onto the next star. This is a problem in the state representation and could be a future improvement to the agent. It can also be seen that piece 2 is both safe and vulnerable at the same time, since it standing on a globe, but can be hit home if red manage to get a piece out. This is an indented situation which is hoping that the agent learns it is standing on an enemy's globe.

2.2 Reward Function

The state representation was made in mind with the reward function such that it would be easy to implement and construct. The reward function is built as

a vector so that each element is a weight for a sub-state. The final reward is calculated by taking the dot product of the reward vector and the state vector, which is the change between the current state and state after taking the chosen action. Each element in the state vector is how the sub-state changed when doing the chosen action. If a sub-state changes from $0 \rightarrow 1$ it is assigned a 1, if it changes from $1 \rightarrow 0$ it is assigned a -1 and if it does not changed it is assigned a 0. The idea with this is to punish the agent when leaving a good state or entering a bad state. But it will be reward when entering a good state or leaving a bad state.

For better understanding an example of how the reward is calculated is presented. The example from section 2.1 is used and it is chosen to move piece 2. The following state for the piece becomes

$$\begin{aligned}
&\text{Piece 1 - } 001100 \rightarrow 001100 \\
&\text{Piece 2 - } 011000 \rightarrow 000100 \\
&\text{Piece 3 - } 001100 \rightarrow 001100 \\
&\text{Piece 4 - } 001100 \rightarrow 001100
\end{aligned}$$

This results in the change vector for piece 2

$$\text{Change Vector} = \begin{bmatrix} 0 \\ -1 \\ -1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{1}$$

Then the dot product is performed with the weights of the reward function

$$\begin{bmatrix} 0 \\ -1 \\ -1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} w_{\text{home}} \\ w_{\text{safe}} \\ w_{\text{vulnerable}} \\ w_{\text{Attacking}} \\ w_{\text{FinishLine}} \\ w_{\text{Finished}} \end{bmatrix} = \text{reward} \tag{2}$$

Different weights for the reward function is tested and will be elaborated on in section 3.

2.3 The Game and Q-Learning

The Python library ludopy [6] has been used to simulate the game. This makes it easy to make moves and get the necessary information of the game. All four players are simulated. The agent is set as player 1, but could also be any of the other players. All four players have to be control by the program. When the three enemy players have to make a move, a random choice is pick each time. This is

done to make them non-intelligent, so that the agent is the only intelligent player. When the agent has make a move it chooses based on the highest Q-values from the Q-Table given a certain state, just as standard Q-Learning. Doing training the move will be random 15% of the time. This is done to make the agent evolve and tried different approach to succeed in the game.

When updating the Q-Table under training, equation (3) is used.

$$Q_{new}(s_t, a_t) = Q(s_t, a_t) + \alpha \left(r_t + \gamma \cdot maxQ(s_{t+1}, a) - Q(s_t, a_t) \right) \tag{3}$$

where Q is the Q-Table, α is the learning rate, r_t is the reward a time step t, γ is the discount factor, s_t is the state at time step t and a is the action.

Since the state representation contains the dice, it is not possible to know which dice will be the next. Therefore there is six next states which all could be possible depending on the dice. To get the maximum of the Q-Table at the next state, the average of all the six possible next states are used. This gives a good estimate on the future maximum Q-Table value.

2.4 Genetic Algorithm Comparison

Sina Pour Soltani present in his paper [7] an agent trained using the GA method. He proposes a methods he calls State Transition Function (STF) which uses 9 descriptors for the information of the game. The methods is similar as the one presented in section 2.1, which is why this paper is used as comparison. The 9 descriptors are:

Move out:	The moving piece can exit home
Travel:	The moving piece can land on a star and travel to next star
Hit:	The moving piece can hit an enemy player home
Suicide:	The moving piece can hit itself home
Safe:	The piece is on a square where it can't be hit home
Vulnerable:	The piece is 1-6 fields ahead of enemy and not safe
Attacking:	The piece is 1-6 behind enemy
Runway:	The piece is on the the 6 squares before goal
Finish:	The piece is on the goal and out of the game

The difference between this state representation and the one presented in this paper is that the STF uses the 9 descriptors to evaluate which piece is the best to move. There the one presented in this paper uses the 6 sub-state to evaluate if the piece that was move, was a good move. But this is just the nature of the two algorithms.

3 Results

Three different agents have been trained, where they each have their unique weights in the reward function. The first agent (equal weights) is where all the

weights are equal. This is properly not the best weight set, but it gives a good base line and can be used to see if some weights are more important than other, when compared to the other agents. The second agent (human weights) is there the author of the paper has set the weights following his intuition of the game and prioritising the weights that he feels is the most important. The third agent (GA weights) is applying the weights from the paper [7], which is explained briefly in section 2.4. The reason this is possible is, because Soltani has a similar state representation and it is therefore possible to transfer some of the weights over to Q-Learning. The reason for doing this is that in comparison to Q-Learning, GA finds out how to distribute the weights and prioritise the ones that are good, taking away the tedious task of setting the weights. So in theory using the same weights from a GA agent should result in similar performance for the Q-Learning. The three weights sets can be seen in table 2.

Weight	**Equal Weights**	**Human Weights**	**GA Weights**
Home	$-1/6$	-0.15	-0.4482421875
Safe	$1/6$	0.10	0.2578125
Vulnerable	$-1/6$	-0.10	-0.4052734375
Attacking	$1/6$	0.25	0.1923828125
Finish Line	$1/6$	0.05	0.0576171875
Finished	$1/6$	0.35	0.494140625

Table 2: Reward weights for the three agents.

For each agent the Q-Table was randomly initialized and was trained over 30.000 games. At every 200 games, a evaluation of 120 games was executed. This gives 18.000 evaluation games. Making a total of 48.000 games for each agent. As mentioned in section 2.3 15% ($\epsilon = 0.15$) of time a random action was chosen instead of the best when training. For the evaluation games there was not taking any random actions ($\epsilon = 0$). This was done to see the full capabilities of the agent. For each test the same discount factor (γ) and learning rate (α) has been kept and is set to $\gamma = 0.95$ and $\alpha = 0.1$

A graph of the cumulative win rate for the 30.000 training games and a graph for the average win rate over the 120 games at each evaluation step has been done for the three agents. The graph for the first, second and third agent can be seen in figure 2, figure 3 and figure 4 respectively. Each graph also contains the win rate for the three enemy players.

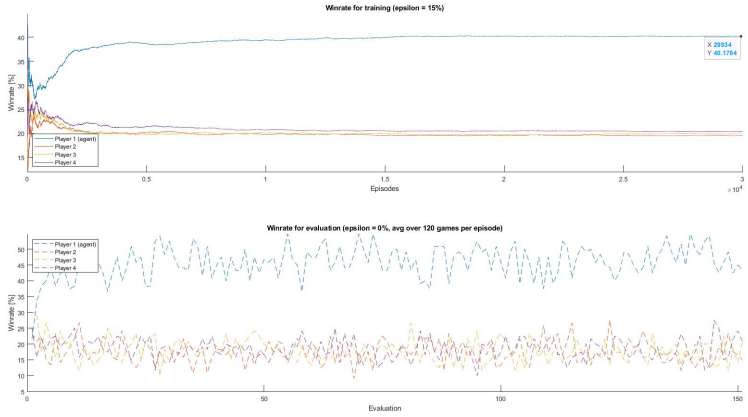

Fig. 2: Cumulative win rate and average win rate for the even weights agent.

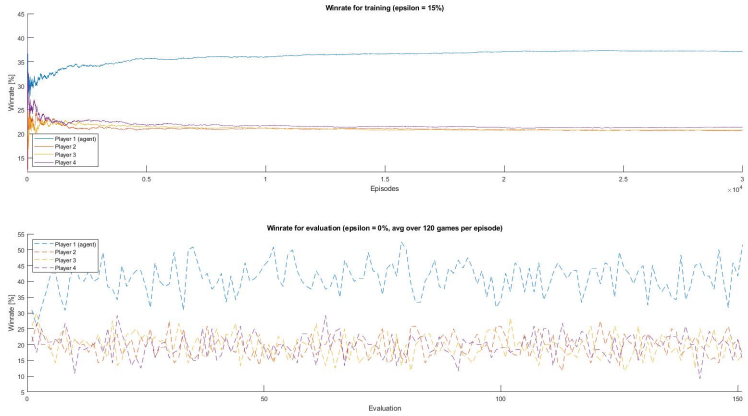

Fig. 3: Cumulative win rate and average win rate for the human weights agent.

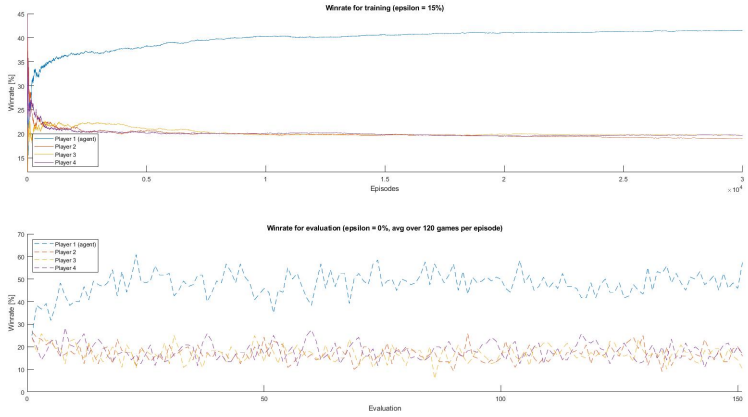

Fig. 4: Cumulative win rate and average win rate for the GA weights agent.

The final win rate over the 30.000 training games for each agent can be seen in table

Agent	Equal weights	Human weights	GA weights
Win rate [%]	40.20	37.11	41.56

Table 3: Final win rate over the 30.000 training games for each agent.

After each agent has been trained they were evaluated over 20.000 games to get the average win rate. Here no random moves was taken. The win rates can be seen in table 4

Evaluation on agent	Equal weights	Human weights	GA weights
Win rate [%]	47.2	42.4	50.5

Table 4: Win rate for the final agents evaluated over 20.000 games.

The result of the STF explained in section 2.4 is a graph with win rate over the generations. See figure 5.

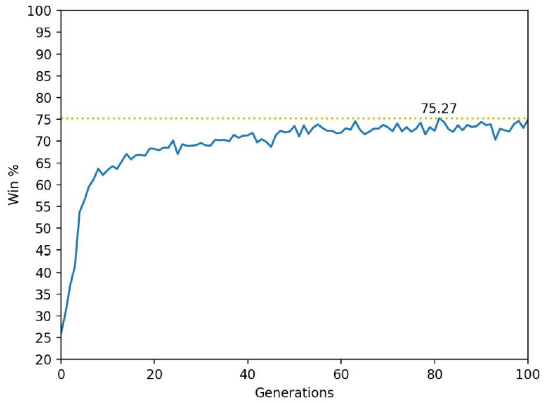

Fig. 5: Win-rate of the STF

4 Analysis and Discussion

From figure 2, 3 and 4 it can be seen that all three agents have similar evolution. The jitter in the start of each graph for the training, is because there is not enough games played yet. So the cumulative win rate is affected a lot by just a single loss or win. But after more games are performed it can be seen that the agents starts to perform better than the enemy's, meaning it is finding a better tactic to beat them. It can be seen that all three enemy players has around the same win rate, meaning that the agent is not proficient in beat a specific player, but rather all of them. This is a good things, since it is more desired to find a general approach to winning games, instead of winning over a specific player. This is the cases for all three agents trained. Also for all three agents, it can be seen that the training win rate stars to flatten around 5.000 training games in. From here on out, only a small increase is happening over the rest of the training. From the three figures it can also be seen that the 120 evaluated each evaluation, is not enough to indicate a steady win rate. If more than 120 games would have been evaluated, a more steady evaluation win rate would properly have been seen. 120 games is not enough remove the effect of the randomization from the dice roll. It can still be seen that the evaluation win rate is still far better than the three enemy's. It can also be seen that the evaluation win rate is often higher than the training win rate. This makes good sense, since the evaluation does not take any random moves, but only the best. This is not the case under training, since 15% of the moves are random. So here the change of getting a worse move is higher. A good idea here could have been to decrease ϵ over the training. So that the agents would take less random moves, then becoming more intelligent.

From table 3 and 4 it can be seen that the GA weights agent performance the best in both cases, the equal weights agent the seconds best in both and the human weights agent the worst in both cases. It makes good sense that the GA weights agent performance the best, since the weights have been trained

on a GA agent, which finds its own weights instead of using preset ones. The equal weights agents from table 3 is very close to having the same performance as the GA weights agent. This can seem like that the reward weights does not have a big influence on the performance of the agent. This is not true, since the human weights agent has a 4.45% and 3.3% lower win rate under training and evaluation respectively. But not just any rewards is necessarily good, since the human weights agent performance worse than the other too. So some weights have more influence on winning that others due. To say which one exactly can not be concluded, since a test where the individual weights had to be perform.

Table 3 and 4 also gives the information that the win rate is higher when evaluation the agents, compared to when training them. This makes good sense since under evaluation the agents does not take any random moves, like they did under training. This means that when taking random moves 15% of the times, severely decreases the performance.

From figure 5 it can be seen that using GA as its agent it has its highest win rate at 75.27%. This is considerably larger than the 50.5% when using Q-Learning. It is a 24.77% increase in performance when using a another algorithm for the agent. But the GA agent has 3 more descriptors than Q-Learning agent. This gives it access to more information about the game. The GA agent knows when it will hit its own piece home, when it will hit an enemy's piece home and when it can land on a star. All of these contribute to a better understanding on what is the best move and can explained the huge difference in performance. The three Q-Learning agents was not able to outperform the GA agent.

A excepted minimum average win rate would be 25% for each the three agents, since all four players in the game should win 25% of the time, if they were all to do random moves, which experiments confirm. Since all the Q-Learning agents have a considerably higher win rate than this, it can still be said that they did perform well. Just not well enough to beat an agent trained using GA.

5 Conclusion

This paper explored the Q-Learning algorithms ability to succeed at the game Ludo. A complex, compact and generalized state representation was design together with a reward function to yield the best environment for three agents that were trained. The agents was trained and tested over 30.000 training games and 18.000 evaluation games, with varying reward functions. The agents where evaluated over 20.000 games after training and the best one used weights trained by an agent using GA and resulting in having a win rate of 50.5%. The agents was compared to an agent training using GA, but could not outperform it. They where still able to outperform non-intelligent players, that took random moves each turn.

6 Acknowledgements

The work in this paper has been inspired by the teaching of Poramate Manoonpong a Professor at Southern Danish University who taught the course Tools of Artificial intelligence[1]. I thank him for taking the time to teach the course, I also thank my fellow students Christian Jannick Eberhardt and Sina Pour Soltani for discussing the project with me and given me useful critique. I thank Sina for allowing me to use his paper in mine.

References

[1] *AlphaGo: The story so far*. Deepmind. URL: /research/case-studies/alphago-the-story-so-far (visited on 05/21/2021).

[2] *AlphaZero: Shedding new light on the grand games of chess, shogi and Go*. Deepmind. URL: /blog/article/alphazero-shedding-new-light-grand-games-chess-shogi-and-go (visited on 05/21/2021).

[3] Faisal Alvi and Moataz Ahmed. "Complexity analysis and playing strategies for Ludo and its variant race games". In: *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*. 2011 IEEE Conference on Computational Intelligence and Games (CIG'11). ISSN: 2325-4289. Aug. 2011, pp. 134–141. DOI: 10.1109/CIG.2011.6031999.

[4] Jens Otto Hee Iversen. *JensOHI/Q-Learning-Ludo*. original-date: 2021-05-23T10:53:31Z. May 23, 2021. URL: https://github.com/JensOHI/Q-Learning-Ludo (visited on 05/23/2021).

[5] *Ludo (board game)*. In: *Wikipedia*. Page Version ID: 1022948236. May 13, 2021. URL: https://en.wikipedia.org/w/index.php?title=Ludo_(board_game)&oldid=1022948236 (visited on 05/21/2021).

[6] SimonLBS. *SimonLBSoerensen/LUDOpy*. original-date: 2020-03-26T17:59:23Z. Aug. 12, 2020. URL: https://github.com/SimonLBSoerensen/LUDOpy (visited on 05/21/2021).

[7] Sina Pour Soltani. "Ludo - Genetic Algorithm". In: ().

[1] Course ID: T550021101