Using Artificial Intelligence to Play Ludo

Lars Pedersen

University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark
larpe17@student.sdu.dk

Abstract. An artificial intelligent (AI) method to play the game ludo is developed. The method used is called Q-learning and it uses a special representation of the game so that the method is compatible. The representation is composed of states that the player piece can be in and actions which can be performed by the player pieces. For the method to understand and be able to win the game training is need to help the Q-learning player learn the game. A reward function is also used when training to determine if the Q-learning player is making a good or bad decision. The training is done against 3 random acting players and it achieves an average win rate of 59.5%. The algorithm is evaluated in two different test, against an alternative AI method and against the alternative method and 2 random acting players. In the first test the presented method achieves a win rate of 56.43% compared to 43.56% of the alternative method. In the second test it achieves a win rate of 46.89% compared to 37.5 % with the alternative method and the random acting players with a win rate or around 7.8%.

1 Introduction

Artificial intelligence (AI) is used for many things today and the task that can be solve by these tools, becomes more and more complex. The game ludo is a semi hard game to play and win for an AI and as well a human, as it depends on strategy and a lot of luck. As the pieces can only be moved depending on the dice and the outcome of the dice is completely random. In this paper we will examine if the use of a reinforcement technique called Q-learning can increase the win rate against other players. The Q-learning will use predefined information about the game, such as states and actions and will then be trained and tested. The code for the implementation is available here [2021]

2 Methods

2.1 Frame Work

To be able to play the game ludo an existing version is being used which already contains the frame work of the game, different methods to retrieve information and other useful functions [2020]. The game also has demo code presented with players that takes completely random actions with no regard to the current game. These players will later be used in the development of the AI method.

2.2 Q-Learning

To play the game of ludo, the method Q-learning is used. Q-learning is a form of reinforcement learning, which means that it uses positive and negative rewards to learn an optimal policy that will enable the agent to choose the best actions in different scenarios or states [2018]. It uses a Q-table which stores the information gathered during a training phase to pick the action which possibly will return the highest reward in the long run for the agent. It can be seen as a lookup table, so when a agent is in a specific state and have a number of actions available the highest Q-value is then picked. To find a usable Q-table the algorithm first have to be trained on the specific problem. The training or learning phase of the algorithm uses equation 1 to update the Q-table.

$$Q(A,S) = Q(A,S) + \alpha \cdot [R + \gamma \cdot \max_a Q(S',a) - Q(S,A)) \quad (1)$$

The way the Q-table is updated is by taking the current Q-value and adding a fraction of the reward and the difference between the highest Q-value based on the actions available in the next state and the current Q-value. The Q(S,A) represents the Q-value of the current state, S, and action, A. α is the learning rate and it determines how much the new information should impact the current Q-value. The value of α is between 0 and 1, where at 0 the Q-table is not updated and nothing is learned and 1 means that all the new data is used [2021]. R is the reward to the agent for taking the action in the current state. The reward is based on a reward function and is specially design for each use case. γ is the discount factor, which determines how much a future reward impacts the current state. If γ is set to 0 or close to, the agent will mostly take into account the current reward given. If γ is set to 1 or close to, a future high reward will have a higher impact. To find the parameters, α and γ, that fits the scenario requires a great deal of tuning and will be addressed late in this paper.

2.3 Implementation of Q-learning with Ludo

To use the Q-learning algorithm to play ludo, it is necessary to set up appropriate states, actions and a reward function which will enable the agent or player to understand the game and learn. The states used for the Q-learning are kept simple and the position of each piece is used. This means that there are a total of 60 states, where state 0 is the home position and state 59 is the goal. As not all the fields in the game are the same, the actions are handling it. There are a total of 11 different actions depending on the pieces position and the enemy pieces. The actions can be seen on the list below.

- Move out of home.
- Use a star.
- Go to globe.
- Move away from globe.
- Enter goal zone.
- Normal movement.
- Stay in goal zone.
- Enter goal.
- Kill enemy player.
- Commit suicide.
- No action.

The actions are based of human interpretation of the game and with expert knowledge about which actions could lead to the player winning the game.

Each time it is the Q-learning players turn, an action is found for each of the 4 pieces. The actions and the states are then used in the Q-table, so the player picks the pieces which will return the highest reward.

The term $\max_a Q(S',a)$ uses the next state and the best action in that state this results in a small problem, as the next state is not known. It is not known because the next state depends on how the enemy players play the game or if they win the game. Therefore the Q-table is updated one action or turn behind. The current state can then be used to find the highest Q-value. That the Q-table is updated behind is also used to make the reward function. As the outcome of taking an action in a state is not known before it is the players turn again. To calculate the reward, the reward function looks at the previous action and the reward is set to a positive or negative value. Actions such as, move out, use a star, kill an enemy and go to goal adds a positive value, while suicide action, move away from safety and no action results in a negative value. Beside the actions different events in the game also triggers an additional reward. This could be wining or losing the game or that the piece closes to home is moved forward. This is done to help enforce a strategy and help the Q-learning player win the game. The size and sign of the reward depends on the action and is set using trial and error and expert knowledge about the game. When initialising the Q-table every element is set to 0 and to help the Q-table convert to a good policy ϵ-greedy is also implemented [2020]. ϵ-greedy enables the player to take some random action with some explore rate to obtain a balance between exploitation and exploration, which is useful and necessary when training. This makes converges of the policy to the optimal one a lot easier and faster.

2.4 Alternative Method for Playing Ludo

To evaluate the method presented in this paper another approach is used to compare it with and it is developed by Jakob Rasmussen [2021]. The approach is also based on Q-learning, but with a different representation of the states, actions and the reward function. This can be used to show how much an impact the setup can have on the results of the algorithm. In total there are 4 local states, one for each of the player pieces. Each local state is composed of 10 boolean statements where each is check and saved as true or false. The boolean statements are made as a human would see the game and this gives the algorithm some prior knowledge or possible strategy which might help it learn. The statements are as follows:

- Is the player piece at home?
- Can the player piece get out of home?
- Has the player piece finished?
- Can the player piece move to goal?
- Can the player piece eliminate an enemy?
- Can it move to a star?
- Is the player piece placed on a safe spot?

- Can the player piece reach a safe spot?
- Is there any enemy player piece within 6 squares from behind?

The action is defined as choosing the player piece which has the highest Q value. The reward function is based on the events in the game and in total it takes into account 8 different scenarios which are as follows:

- A player piece is sent home and in the previous state it is indicated that an enemy was near.
- The player piece eliminated itself.
- The player piece moved out of safety.
- The player piece eliminated one of the opponents tokens.
- The player piece finished.
- The player piece moved to a square containing a star.
- The player piece moved into safety.
- The game is finished and the player won.

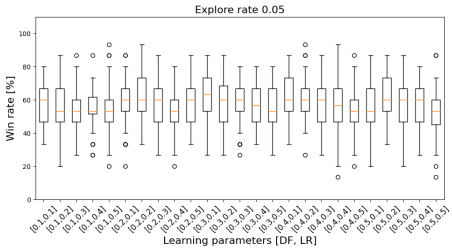
The value of the reward from different events in the game is found using expert knowledge about the game. The alternative method is trained against 3 Q-learning players all using the same Q-table and updating it. This could make the agent learn more and better since its opponents are more sophisticated and their actions are more intelligent. Some tuning have gone into find the best parameters to make sure that the player achieves the highest win rate possible, which can be seen in the paper [2021].

3 Results

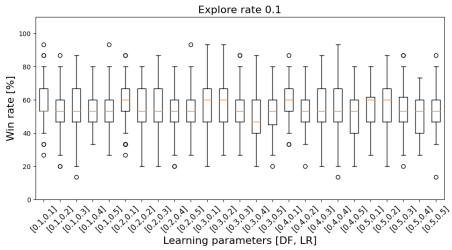
3.1 Tuning of Parameters

In total there are 3 parameters that needs to be found and optimised to assure a good training phase, the learning rate, the discount factor and the explore rate. Multiple tests are conducted to establish what parameters would achieve the highest win rate. In total four different explore rate, five different discount factor and five different learning rates are tried, which resulted in 100 test as all parameters are tested on each other. The Q-learning player is pitched against 3 random action taking players and the Q-learning player is trained on 400 games or in episodes and is then stopped. After each game is completed, the explore rate is set to 0 and training is turned off. This makes it possible to find the win rate and see if the Q-learning player has learned anything. The win rate is then found by playing 15 games and dividing number of wins by number of games played. The explore rate is then reset and training continues.

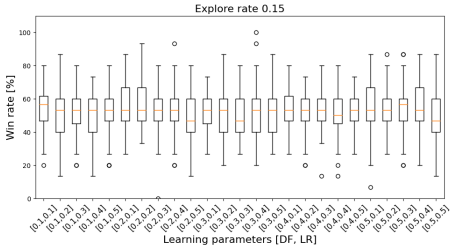
On figure 1 the result of the 100 test can be seen. The box plots visualise the win rate of the last 100 games and each subplot is with a different explore rate. From the plots it can be seen that the median win rate is around 50% for each test no matter the parameters. To find the best parameters, an average of the last 100 episodes of the training is taken and the one with the highest win rate is with an explore rate on 0.05, discount factor on 0.4 and with a learning rate on



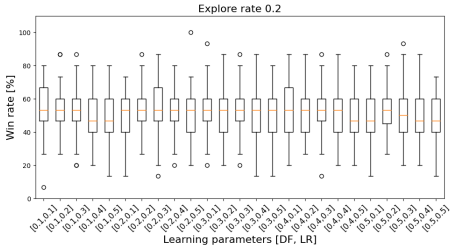
(a) Explore rate 0.05.



(b) Explore rate 0.1.



(c) Explore rate 0.15.



(d) Explore rate 0.2.

Fig. 1: Box plots showing the win rate from the last 100 episodes of the training with different parameters. The box plots are plotted against the used learning parameters for the discount factor and learning rate.

0.1. This resulted in an average win rate of 62.9% for the last 100 episodes. The same parameters are then used to train the Q-learning player on 1000 games to see if the performance could be improved with more training. After each game a total of 25 games are used to find the win rate, with explore rate set to 0 and training turned off. On figure 2 the result can be seen.

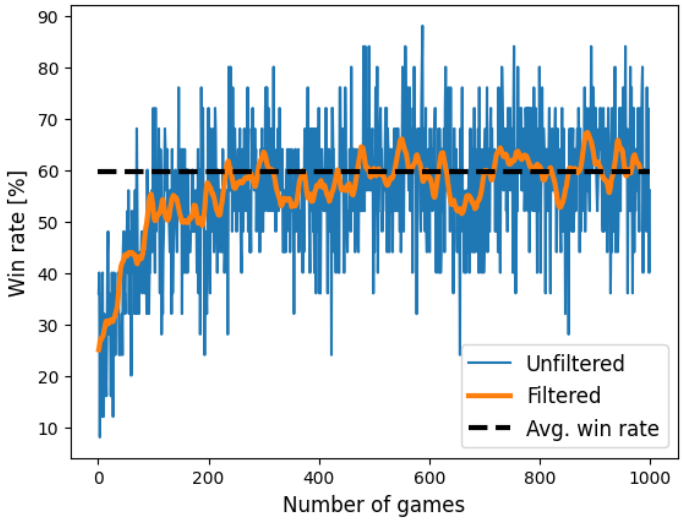


Fig. 2: Explore rate: 0.05, discount factor: 0.4 and learning rate: 0.1.
A moving average filter is used to filter the data, which takes an average of the last 15 data points. The dashed line represents the average win rate of the last 500 episodes of training.

The blue line on the figure is the unfiltered win rate, which is noisy do to the random element in the game. A mean average filter is used to better interpret the data without the noise. It can be seen that the Q-learning player learns a policy in the first 100-200 games, where the win rate goes from about 25% to around 50% and after that the win rate only improves slightly. The average win rate of the last 500 games, can be seen on the figure as the dashed line and results in a win rate of 59.5% against the 3 random action taking players.

3.2 Presented Method vs. Alt. Q-learning method

To compare the presented method with the alternative method two test are performed. The data collected is the number of wins for each method over multiple games, so it is possible to find the win rate. The tests are performed using the Q-table learned with the best found parameters presented earlier in the paper. It is trained using 3 random action taking players and a total of 1000 games. The

alternative method provided by Jakob, also usest a pretrained Q-table with the use of the best parameters stated in the paper [2021]. While the test is running, both Q-learning methods have an explore rate of 0 and the Q-tables are not updated.

The first test pitch the two Q-learning methods against each other, with the last two players idle. A total of 10000 games are played to find an accurate win rate of each player and the result of the test can be seen on figure 3.

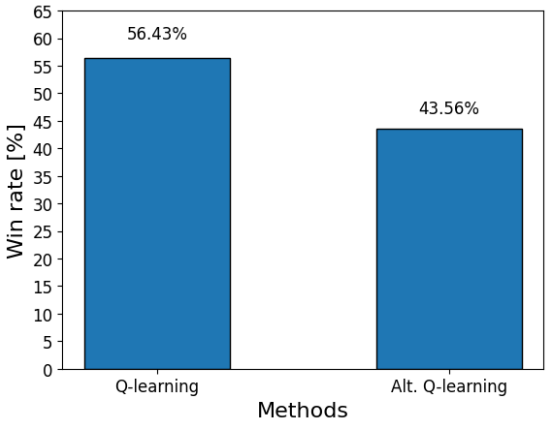


Fig. 3: Win rate for the different methods when playing against each other based on 10000 games.

The tested showed that the presented method is able to achieve a win rate of 56.43% and the alternative method achieved a win rate of 43.56%. This means that the presented Q-learning method have a higher win rate and performed better in this test. The second test is performed in a similar manner as the first test, except that the two idle players are now random action taking players. The result of the test can be seen on figure 4. It can be seen that the presented method achieved a win rate of 46.89% and the alternative method at bit lower at 37.5%. The two random action taking players have about the same win rate at 7.77% and 7.84%. The presented method again got the highest win rate from 10000 games played.

4 Analysis and Discussion

The presented method reached an average win rate of 59.5%, but it can be seen on figure 2 that the win rate fluctuates a lot do to the randomness of the game. This affects how the policy is learned, which means that multiple runs with the same parameters could result in different win rates and in the end a different set of parameters achieving the highest win rate. It is possible that the

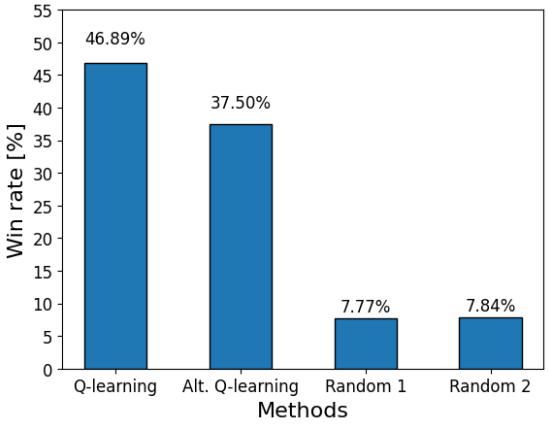


Fig. 4: Win rate for the different methods when playing against each other based on 10000 games.

learning curves has settled which means that the policy has already converged and the randomness is removed or minimised. Only more training and more test with different parameters could substantiated this. The reason for this not being examined more is due to time constrains.

Another problem is with the current training method. The Q-learning player is trained on random acting players. This means that the policy is learned based on how a random acting player is playing and not how the game of ludo actually is played by a human or a different smart player. This could have an impact on the win rate when pitched against the alternative player. An alternative method could have been used to train the Q-table. It could have been pitched against a real human or trained on multiple Q-learning players which all update the same Q-table, as presented in the paper with the alternative Q-learning algorithm. This might have given the Q-learning player a better understanding of the game and resulting in a higher win rate in the end.

Another thing that has an impact on the results of the Q-learning player is the reward function. The reward function used is found by trial and error and some knowledge about the game, but not a lot of thought have gone into it. As the reward function is used to determine what the algorithm defines as good and bad and how much, it is an essential part of the algorithm.

Based on the two test comparing the presented and alternative Q-learning method, the best method is the one presented in this paper. It achieved a win rate at about 10% points higher than that of the alternative method even though that they are based on the same reinforcement learning method. The alternative method still obtain a high win rate compared to the two random action taking players, which also shows that the alternative method works and takes strategical actions. This shows how big an impact the states and action have on the results

and the policy learned by the methods. The two methods have a similar manner of interpreting the ludo game. Both of the methods uses Q-learning, but learned different things. From the test with two Q-learning players and the two random acting players it should also be noted how high the win rates are of the random acting players. In a game with two trained AI algorithms the two other methods are still able to obtain a win rate of around 8%. It substantiates how much luck has a say in the game and makes it impossible to achieve a win rate of 100% for any algorithm implemented.

5 Conclusion

A reinforcement learning method name Q-learning is implemented to play the game of ludo. The method needs an interpretation of the game to be able to play the game in the form of what states to be in, what actions to take and how different actions results in different rewards. The states are based on the position of each piece in the game. The actions are based on how a human would interpret different actions in the game, such as move out of home and use star. The rewards functions uses previous actions and knowledge about the current game, to determine what reward should be give. The size of the reward and sign is predefined and values are found by trial and error. Based on this, the method is trained against 3 random acting players with multiple different parameters trying to find the optimal once's that will enable the algorithm to achieve the highest win rate. The best parameters is found to be explore rate on 0.05, discount factor on 0.4 and with a learning rate on 0.1. The win rate with these parameters is 59.5% on average after 1000 training games. The method is also tried against another Q-learning method developed by Jakob Rasmussen. It has a different interpretations of the states, the actions and the reward function, which gives in a different way of picking the actions. A game is setup where the two methods are playing against each other and the result of this is that the presented method in this paper achieves the highest win rate of 56.43% compared to 43.56% of the alternative method. A second game is also setup where the two method are playing against each other together with two random acting players. In this case the presented method achieves a win rate of 46.89% compared to 37.5 % with the alternative method and the random acting players with a win rate or around 7.8%. Based on these test the presented method is the best methods to play the game ludo.

6 Acknowledgements

A special thanks to Jakob Rasmussen for providing the alternative method used to test the presented method, which made it possible to implement and test. He also helped with ideas, suggestions and discussion of the project.

7 Bibliography

References

[2021a] Lars Pedersen : ludo game AI2
https://github.com/kulus96/ludo_game_AI2
[2021b] Wikipedia : Q-learning
https://en.wikipedia.org/wiki/Q-learning
[2021c] Jakob G. Rasmussen (jakra17) : Q-learning and GA based Ludo Players
[2020a] GeeksforGeeks : Epsilon-Greedy Algorithm in Reinforcement Learning
https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/
[2020b] Simon L. B. Soerensen : LUDOpy
https://github.com/SimonLBSoerensen/LUDOpy
[2018] Richard S. Sutton, Andrew G. Barto : Reinforcement Learning, second edition,
page 131-132.