An Evolving Ludo Player Based on Genetic Algorithm

Petr Batek

University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark
pebat16@student.sdu.dk

1 Introduction

This paper has been written as a semester project for class AI2 at the University of Southern Denmark. Goal of the project was to develop an automatic Ludo player. The player had to be based on some learning or evolution tool, which was covered by the course, such as Artificial Neural Networks (ANN), Temporal Difference learning (TD) or Genetic Algorithms (GA).

This paper presents a player based on GA.

To evaluate player's performance in a game, the player is compared to simple Ludo player, which takes only random turns, 3 types of strategy players, each of which uses one simple strategy and one advanced but tuned by human expert player.

AI players develop by classmates Michael Kjaer Schmidt, Bjarki Pall Sigurdsson and Jesper Bogh Poder were used for further comparison in a tournament.

2 Methods

A genetic algorithm was selected for developing the AI player. The very first step in GA is to select features, which represent a chromosome of the agent. In order to define a suitable chromosome, different possible play strategies were studied first and are described in the next paragraph.

2.1 LUDO strategies

Fast play strategy player follows one simple rule: Always move a piece, which is the furthest in a game play. This rule minimizes the risk of having knocked out the piece, for which player utilized the largest number of turns. Naive strategy is to move this piece no matter the consequences (possibility to being knocked out when hitting an opponent on a globe field or his blockade). This naive strategy can be combined with avoiding of self kills.

Aggressive play strategy always tries to eliminate opponent. When it is player's turn and dice roll is known, it is possible to identify, whether there is possibility to hit opponent's piece by one of available move. This strategic player prefers

move, which leads to hitting opponent. If there is no such a possibility, move can be selected randomly. Naive player again does not take into account that its move can eliminate its own piece, but naive strategy can be combined with advanced ones.

Defensive play strategy first evaluates how many of opponent's pieces endanger each of defensive player's pieces. After this the next turn is simulated for each of possible moves and number of endangering pieces is computed again. Piece to be moved is selected based on the largest difference of these two values. Deuces are broken randomly. This is the most advanced strategy from the ones described so far. It can be combined with others again.

Other play strategies

There are many other features in the LUDO game, which are worth considering for putting into chromosome of AI player. Here is their brief list:

- Prefer escapes from jail - after each dice roll of 6, player prefers to move its piece out of jail in case there is still any
- Try to avoid wasting turns by moving a piece, which is in a goal stretch and its move does not result in hitting a goal field.
- Prefer moves, which leads to placing piece on the goal.
- Avoid self kill - filter out the moves, which ends in killing player's piece by its own move.

For evaluating players performance one extra player was defined. It is random player, which always identifies all possible moves and picks one of them randomly.

For reference of a strength of these basic players, they were put through a test which consisted of 1000 LUDO games where statistics of their wins were taken. As can be seen from Fig. 1a defensive player performs the best from basic strategic players and fast and aggressive are close to each other.

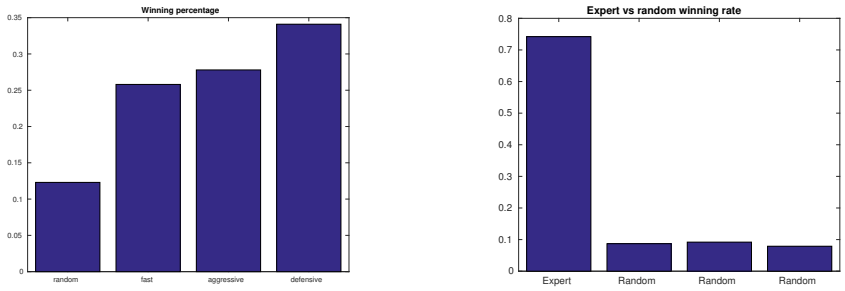
It is apparent that some combination of individual strategies leads to even better performance.

Expert player

An expert player was developed based on experience gained during designing of basic strategic players. This player identifies defensive, aggressive and other features of each move. Every feature was given a weight, which was manually fine tuned. Performance of the expert player against random players is shown on Fig. 1b.

Winning rate of 70% is a very good performance, but it was necessary to manually tune weights for all features of the moves.

Development description of automatic LUDO player, capable of self tunning its weights, follows in the next sections. Genetic algorithm was used to automatically find optimal weight for each feature of the moves.



(a) Strategic players vs. random (b) Expert player against random

Fig. 1: Comparison of strength of strategic and expert players

2.2 Genetic Algorithm design

Population size was chosen of 4 players, since the LUDO game is game designed for 4 players. Main reason behind this choice was the simplest way of implementation even though larger populations might give better results. Even this small population size gives a possibility to perform genetic algorithms as is described subsequently.

Chromosome of the AI player is possible to compose after identification of basic game features. Chromosome consists of values which represent weights applied to different game features, which were described. The weights are represented as a floating point numbers, which leads to a value encoding of the chromosome.

Table 1: Chromosome as weight for game features

defensive	aggressive	fast	escape from jail	move in goal stretch	move to goal	self kill move

Let W represent vector of the weights = chromosome. A value vector of the same size as W is identified for each player's turn. Eg.when the move leads to hitting an opponent, it has second item equal to 1. Or if the move ends up in a self kill, it has value of 1 for the last element. Every element of feature vector has value either 0 or 1 except of defensive feature. This takes value of the difference of endangering opponent before and after move.

Feature vector for all of the possible movements is multiplied with the values from the chromosome vector W every turn and the movement with largest result

is selected as the next move. This move evaluation makes it possible to change players behavior by adjusting chromosome values.

Fitness function for LUDO game is very simple. 1000 games are played for every generation and individual player performance is evaluated based on the percentage of winning.

Replacement strategy was designed so it keeps two best players for the next set of 1000 games, whereas two worst are replaced by offspring of the best players. This leads to elitism replacement strategy of half the population.

Recombination is performed by selection of two best players, which are used to create new offspring. Uniform crossover was selected for gene recombination. Each gene is crossovered with probability 0.2. Mutation takes place after crossover. Each of the genes is mutated with probability 0.5 and the mutation value generated from normal distribution $N(0; 0.1)$ is added to the previous value of the gene. This results in sequential building of weights across multiple generations.

3 Evolution and Results

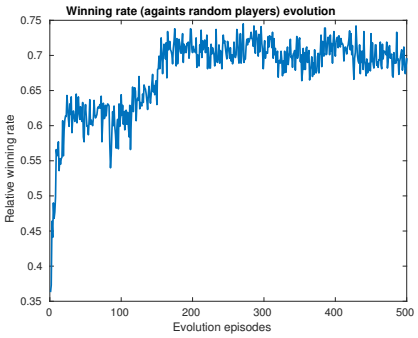
Following setup was used for training and testing of the algorithm. 4 genetic players were created with these chromosomes:

- player 1 (defensive): $[1, 0, 0, 0, 0, 0, 0]$
- player 2 (aggressive): $[0, 1, 0, 0, 0, 0, 0]$
- player 3 (fast): $[0, 0, 1, 0, 0, 0, 0]$
- player 4 (escaper from jail): $[0, 0, 0, 1, 0, 0, 0]$

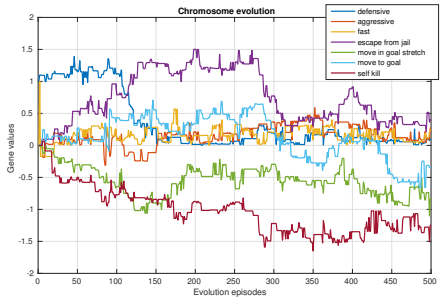
These players play against each other during training sets of 1000 episodes. After each set, they are recombined and the best player from set's generation is tested against 3 random players. Test is performed by playing another set of 1000 episodes. Tests against random players are taken as standard measure of genetic player performance. Evolution of the winning rate of the best genetic players is on Fig. 2a.

Apparent improvement in the genetic player performance can be seen. Winning rate starts around 35% for a defensive player and evolves to the level of 70%. There are significant oscillations due to recombination and subsequent rise of the new champion among genetic players. Another reason beyond these oscillations is a probabilistic nature of the game. Rolling of a dice is random event and therefore even the game results are stochastic.

Evolution of chromosome of the best genetic player is plotted on Fig. 2b. Oscillation from the reason of recombination and substitution of the best players are again present. Even thought these oscillations, there is present convergence pattern towards optimal values of genes.



(a) Wining rate evolution



(b) Chromosome evolution

Fig. 2: Evolution of Genetic Player

Table 2: Chromosome after 500 evolution cycles

defensive	aggressive	fast	escape from jail	move in goal stretch	move to goal	self kill move
0.0972	0.2729	0.2363	0.3999	−0.8356	−0.3455	−1.2492

Chromosome values at the end of 500 evolution episodes are in table 2.

This however is not the best player, which emerged during evolution. It is possible to find that the best player from the whole evolution achieved a winning rate of 74.5% and had chromosome from table 3.

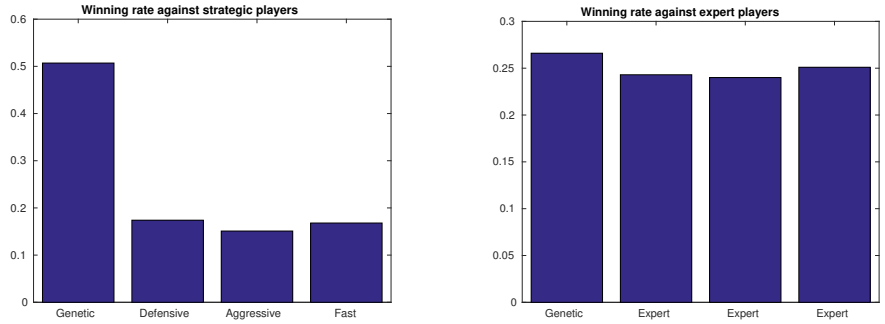
Table 3: Chromosome of the best player throughout evolution

defensive	aggressive	fast	escape from jail	move in goal stretch	move to goal	self kill move
0.1060	0.2681	0.2329	1.2725	−0.3701	0.5848	−1.0245

Game play against more difficult opponents

The genetic player from the last generation was tested in games with more advanced opponents. One test was performed against 3 players using basic strategy (Fig. 3a). Opponents were: defensive, aggressive and fast strategy players.

Next test was performed against an expert player. Each move for this player was evaluated and categorized similarly as for genetic player. However the weights were assigned fixed and manually fine tuned in order to perform reasonably well. Performance of evolved genetic player against the expert is shown on Fig. 3b.



(a) Winning rate against strategic players (b) Winning rate against expert players

Fig. 3: Winning rates against advanced opponents

As can be seen on Fig. 3 genetic player performed great against basic strategic players. However when played against fine tuned by human expert player its dominance was only light.

4 Analysis and Discussion

Chromosome evolution is one part worth of analyzing. After inspection Fig. 2b interesting happenings can be discussed.

Notice for example, that weight for moving onto goal becomes negative at around generation of number 300. This is in contradiction with the main goal of the game. However there is possible reason why evolution resulted in weight being negative. The most of the moves onto goal takes place from the goal stretch. Piece, which is in the goal stretch is safe from other opponents. And assignment of negative weight is a way how to take this game state into account.

Another interesting thing is the start of the evolution. Notice that the defensive chromosome dominated the first 100 generations until it was replaced by preference of escapes from jail. This progress is dependent on chromosome values of initial players. In case of aforementioned initial players it shows that defensive game strategy outperforms other strategies.

The aforementioned winning rates are just one observation of the winning rate for the set of 1000 games. This means that there can be exceptional cases where evolved player performs differently. These cases will constitute only minority or exceptions. Some statistical tests or confidence intervals would surely better validate obtained results. It would require to simulate several sets of 1000 of test games for each evaluated genetic player. Mean, variance or confidence intervals of sample's winning rates could be computed from results of these test. However because of time restrictions it was not possible to perform these tests.

5 Comparison against other AI player

The evolved player was compared to 3 other classmates AI players for the final evaluation. Because of limited space for this paper, only one strategy of these players is described. AI player of Michael Kjaer Schmidt was used for description since it uses different AI method than the one used in this paper.

5.1 Michael's player

Even though Michael's paper claims it uses Q-learning AI method I would rather call it Temporal Difference learning based on the reasons described in this section.

Implementation of Qlearning uses a Q-table to represent *state* x *action* value. And as is described in [1] these *state* x *action* vales can be represented either explicitly by a table or by an ANN. ANN has a benefit of possible use for larger state space problems; however, it requires sophisticated method for training. Michael used value representation by a table which lead him to necessity of reducing state representation for his player.

He represented the state by the position of the brick (0-57), the current dice roll(1-6), a boolean value representing if the current dice roll moves the brick to an opponent(true/false) and a boolean value representing if current dice roll moves the brick to a friendly brick(true/false). This resulted in Q table of size 1392.

This is still very large matrix traded for the big reduction of state information.

Designer needs to specify current state rewards when using a Q learning method. Assigned rewards are specified in Tab. 4.

Table 4: Rewards and punishments the agent receives. Taken from [1]

Move:	Reward/punishment:
Brick leaving base	+5
Selecting brick in base when dice roll is not 5	-1
Brick moving to a star	+5
Brick moving to a globe	+5
Brick moving from a globe	-10
Brick moving to an opponent (sending opponent home)	+5
Brick moving to a friend	+10
Brick moving to goal	+5
Brick overshooting the goal	-10

The value table is updated every turn by the formula:

$$\Delta Q(s) = \alpha \left(r + \gamma \cdot maxQ(s_1) - Q(s) \right).$$

This is not a typical form of Q-learning. Action is completely missing in the update and according to [1] it was omitted intentionally. Based on this reason Michaels method is more just a Temporal Difference learning than a Q-learning.

Fig. 4 shows that the player's best performance was 53.6% winning rate. However its performance oscillates quite significantly.

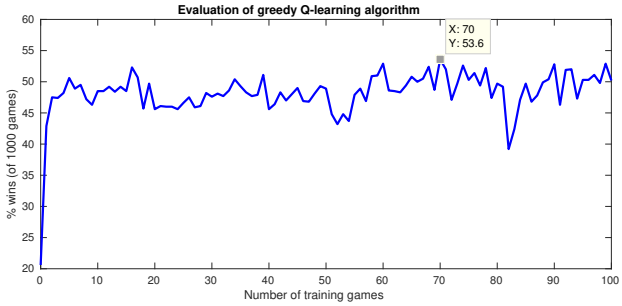


Fig. 4: Performance of Michaels player Taken from [1]

5.2 Tournament

A tournament was played for the final evaluation of the players. Players in the tournament were:

- Genetic player described in this paper
- Michael's player using TD-learning [1]
- Jesper's genetic player [2]
- Bjarki's genetic player [3]

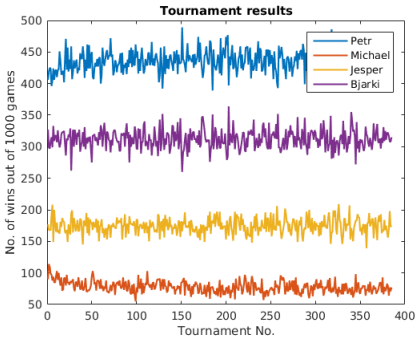
Both of the other genetic player were evolved in a larger population than the one developed in this paper. However they played only against random opponents during their evolution.

These 4 players were placed into the game and multiple tournaments consisting of 1000 games were played. Results of the games are on Fig. 5.

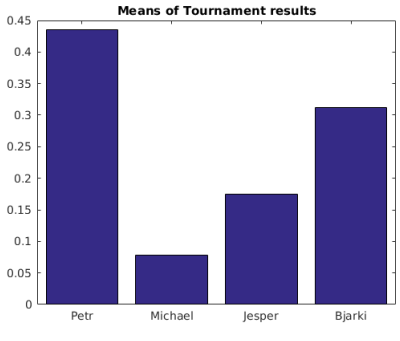
6 GA vs. Q-learning Discussion

Tournament revealed that all GA players performed better than the one using TD or Q-learning. Even though this can be just because of the one particular implementation of the algorithm this situation needs a discussion.

I think that GA is better suited for the LUDO game. There is a one main problem in using Q or TD learning for this game. Since these methods require to choose the second action from the state where the agent ended up after the previous move. This is very difficult to follow in games where several players



(a) Strategic players vs. random



(b) Expert player against random

Fig. 5: Results of the Tournament

play against each other. The value of the state can not be evaluated only based on the next action which is possible to be taken. There are other opponents in the game, which can change player's state before its next turn. This results into a situation in which the one particular state can have different values according to whether it is the state before player had done its turn or after.

It is hard to conclude whether one algorithm is better than the other. But based on the experience that I had chosen to implement Q-learning algorithm at first and gave up after long but unsuccessful work, I can conclude that implementing the Q-learning for a LUDO game is more difficult task.

It is my believe that Q-learning could actually outperform GA players. The reason behind this is that Q-learning takes information about different state values into account. On the other hand GA only performs more or less random evolution of player chromosome. By random I mean the mutation of the genes. There is of course elitism crossover part of evolution which is deterministic.

The main benefit of Q-learnig over the GA is that it can take much more information into account. And this is also its main drawback since it is very demanding to organize and evaluate.

7 Conclusion

This project showed that a much simpler AI agent can outperform more sophisticated implementations. Based on the previous discussion I consider Q learning agent the sophisticated one. Yet still it was defeated by all of the other players.

The player described in this paper is definitely one of the simpler ones. Even the population size for its evolution was chosen to be only of 4 players which is generally a small population size. In contrast the chosen training technique, namely evolving GA players inside their own population turned out to be a good choice, which helped in winning the tournament.

For the next work I would definitely consider implementation of a Q-learning player. The reasons were described above. GA for this project was chosen mainly because of time restrictions and its ease of implementation.

8 Acknowledgement

I would like to thank to our class Associate Professor Poramate Manoonpong whose lectures guided us throughout the project. The next acknowledgement belongs to people who provided their AI agents for final evaluation tournament. They are namely Michael Kjaer Schmidt, Bjarki Pall Sigurdsson and Jesper Bogh Poder. And finally one more thanks to Michael who proof-read this report and suggested a few better formulations and phrases.

References

1. M. K. Schmidt, “Playing ludo using q-learning,” 2017.
2. J. B. Poder, “Using a ga to play ludo,” 2017.
3. B. P. Sigurdsson, “A genetic ludo-playing algorithm,” 2017.