Tools of Artificial Intelligence - Genetic Algorithm Ludo player

Frederik Mazur Andersen

University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark
fande14@student.sdu.dk

1 Introduction

This paper is about creating an optimal ludo player by using an artificial intelligence method. The player will then play against another trained and implemented artificial intelligence method. In this paper the method of Genetic Algorithm(GA) will be used to evolve a population of ludo players into better players. The players will make decision with a weighted sum to pick the right option. The GA will train the weights of the player.

2 Methods

In this section the methods, of the implemented GA algorithm and the representation of the problem, will be discussed. A brief overview of the secondary artificial intelligence method for comparison will also be discussed.

2.1 Representation of the problem

The problem is to select the correct piece given a dice roll. The implemented representation of the problem has the parameters seen in table 1. The parameters are taken into account and used to give a weighted sum for each piece. The piece with the highest weighted sum is then selected. If a piece is in the finished goal it isn't included as a valid piece to make decision for, as it has already won.

The parameters all return `0` or `1`, besides `Moved Distance` and `Risk of getting killed` that both returns integers, based on the tiles moved, or how many enemies that is within range. The parameters that represent negative actions like `suicide`, `risk of getting killed`, `loss of safety` and `move in goal stretch` returns negative values as they are risks or bad behaviours and thus should counteract selecting the piece.

Parameter	Return Values	Return Type
Moved distance	Number of tiles moved	integer
Gain of safety	Returns 1 if true else 0	boolean
Loss of safety	Returns -1 if true else 0	boolean
Can escape home	Returns 1 if true else 0	boolean
Can kill another player	Returns 1 if true else 0	boolean
Move is suicide	Returns -1 if true else 0	boolean
Risk of getting killed	Negative number of enemy pieces within range	integer
Move to goal stretch	Returns 1 if true else 0	boolean
Move to final goal	Returns 1 if true else 0	boolean
Move within goal stretch	Returns -1 if true else 0	boolean

Table 1. Parameters used to create the weighted sum for decision making. They represent the problem.

The parameters representing the problem will briefly be described in the following sections. The representation of the problem tries to take as many parameters into account for the different states of the game, while still being a simple representation.

Moved distance This is the number of tiles moved. It also takes the movement of stars into account to accurately give the number moved. This parameter is there to help select the piece that can move the longest and push for a behaviour where it takes stars when possible and still stay relatively safe.

Gain of safety This checks if the new position gives safety to the piece. This could be if there already was a friendly piece on the new position or if the position is an unoccupied globe. If no safety is gained it returns zero and doesn't get taken into account. This helps reinforce to get safety when possible.

Loss of safety This checks if the piece is moved away from a safe position such as globe or being together with another friendly piece. This tries to get the piece to stay in safety if there is another piece that could be moved instead without getting into a dangerous position.

Can escape home Simply checks if the dice-roll is a 6 and allows to move a piece out from home. This is important to have as an parameter to avoid wasted turns because no other pieces is out and the dice is not rolling sixes.

Can kill another player This checks if the new position means sending another player home. This makes the player more aggressive, but also helps in delaying the opponents and thus make it more plausible that the player will win.

Move is suicide This checks if doing this move kills yourself by hitting an enemy that is safe. This parameter ensures the player doesn't make very bad decisions like this, when another piece can be selected.

Risk of getting killed This checks the new position and a range of 6 tiles behind the position. It then sums all enemy pieces within this range as they have the possibility to kill this piece on their next turn. If the new position is a star, then a range of 6 tiles behind the previous star is also checked, as that also would be able to kill the piece. This helps to detect if moving a piece to a dangerous position.

Move to goal stretch This returns positive value if the piece can move to the goal stretch. This helps the player prioritise to get pieces into the goal stretch when possible as the piece then is totally safe.

Move to final goal This checks if the piece can move to the final goal and finish for good. This helps the player to select a piece in the goal stretch to get into the goal, when the correct dice roll to hit the final goal is archived. This is useful if there is no other better move and it can finish a piece.

Move within goal stretch If the piece is in the goal stretch where it is completely safe, then it shouldn't move unless it can land on the final goal. Other moves would just make it go back and forth within the goal stretch and potentially waste moves. This checks if the piece is doing just that.

2.2 Own implemented GA method

The implemented training is a steady state single, 20 specimen, population GA with standard real value mutation, tournament selection[3] and discrete crossover.[1] It is chosen and implemented to make between 20% and 50% new children every generation. The new children replaces the current specimens with the worst fitness. This is chosen to gain a faster convergence but still have a steady state population.

Most of the parameters of the GA was selected based on testing of different parameters. The experiments was trying out different parameters and look at how fast it converges and how good fitness it manages to acquire. It was also noted how much a difference the reported result had when tested over 40 000 games. This paper will not show results from all these experiments, but will merely report the optimal parameters found and why this is thought to be the optimal parameter.

The program is implemented with a population-manager class that receives statistics from the games and handle the interaction between training specimen and the game. It controls when the next game can be played and it changes the weights of the player when a new specimen should be trained. It uses four signals to interact with the game class and control the training.

Population size A size of 20 is chosen as it appeared to perform better than higher size populations with faster convergence and just as good fitness results if not better. Smaller populations can require more mutations to cover the full search space, but also requires less training per generation. The optimal size is the compromise of covering a large search space while still being small enough for faster training. A size of 20 seemed like a good compromise.[4]

Fitness The fitness is based on win-rate over the training games. Each specimen plays a set amount of training games and as results get an average win-rate. The reason for multiple training games is to accurate get the performance of the specimen and not just be lucky in one game and get a much higher performance than what the specimen in reality would converge to. The amount of training games is set based on results of a experiment. The experiment is letting four random players play against each other and see how many games it takes before they converge at approximately 25% win-rate each. The experiment allowed +-1% from 25% for each player, but required to have 30 games in a row where the win-rate was within this interval for it to be converged. The experiment showed that around 1800 games should be played to accurately converge. This however changed the running time per generation quite a lot and didn't seem to help converge faster, just give more accurate average win-rate. Thus based on experiments with different training games the training games was set to be 1000. Results showed that the final win-rate would show 3-4% wrong when tested over 40000 games, but this small deviation was deemed acceptable as it decreased the training time substantial.

Initial values The specimen was randomly set between 0 and 20 per weight when initialised. This range was chosen as a guess and it should give enough span between the weights to adjust correctly in.

Tournament size To keep diversity but still get majority of the fittest players, the tournament size is set at four. Experiments showed a size of four with a population of 20 gave good performance. The tournament is implemented to find the same amount of winners as wanted offspring plus two. This means that multiple tournaments would be held until enough winners was selected. The extra two winners was added to handle cases of only wanting one or two offspring and would thus need a few extra winners to be able to create enough offspring.

Crossover Standard discrete crossover with randomly selecting which parent provides what chromosome, is used. As the method has a probability to how many offspring it should create the method takes all the winners from the tournament and crossover each winner with every other winner. This creates a big pool of offspring. The offspring is then selected randomly from the pool until enough offspring has been selected. The selected offspring then replaces the worst fit specimen in the population.

Mutation The probability for mutations was set to 2% and the range of possible mutation was a real value of +- 2. This is a bit higher than what some recommends[2], but experiments showed it gave better performance than lower probabilities. Using steady state also means that it is harder to keep diversity in the population and thus a higher mutation probability can help with that.

Training opponents Both purely random players and the semi-smart players was used to train against. Experiments showed that training against the random players gave higher average win-rates, but when tested against the semi-smart players the win-rates would fall quite an amount. Thus the last experiments with the best performance was only training against the semi-smart players. This meant that when the trained specimen then played against random players the trained win-rate wouldn't be accurate, but instead increase. This indicates that training against semi-smart players gave best results in general when playing against different ludo players.

Termination of training The training was set to terminate at maximum 1500 generations. Experiments showed that most populations of size 20 converged to their best result before generation 1000. A graph displaying this is shown in figure 3 in section 3.

Performance testing Testing the performance of the best specimen was done by letting the specimen play against three random players, then three semi-smart players and three players from the competitors method. It would play 10000 games in each experiment and notice the average win-rate from the games. The experiment would then be repeated 100 times to obtain a mean win-rate. In total one million games would be played. The resulting average win-rate is then used as results.

2.3 Competition method

The competition method given by Per Breum is using the reinforcement learning method Q-Learning to make the decisions. It is implemented and trained by Per Breum and then employed to test against my best player from the GA approach. The Q-Learning method uses 9 states and 15 actions which makes up the Q-table. The states are typical states like goal, home, and globe positions. The actions is likewise typical actions in a ludo game like move to globe, move out of home, kill an enemy piece and so forth.

3 Results

A graph over the better populations and their convergence is shown in figure 1. The graph displays the fittest player in each generation and the average fitness of each generation. It shows that the initial guesses is important in terms of quickly getting higher fitness, but in the end the populations converge to about the same fitness. Note how the smaller populations outperforms the bigger population both in convergence rate and fitness. A box-plot of every 10th generation from the best performing population is shown in figure 2. The box-plot gives a good overview of the development and how each generation performs. It is seen that especially in the steep increase of fitness in the beginning there is a wide spread of performance. As the fitness starts to converge the spread starts to get smaller and smaller. As expected outliers can be seen especially at the bottom of each generation as this is the new child that has replaced the worst performing specimen. As the children performance is unknown when the replacement happens the new specimen could be worse and thus the amount of regular outliers in the bottom of the population. It is also noted that the best specimen is a small outlier as the average population isn't as fit as the best individual.

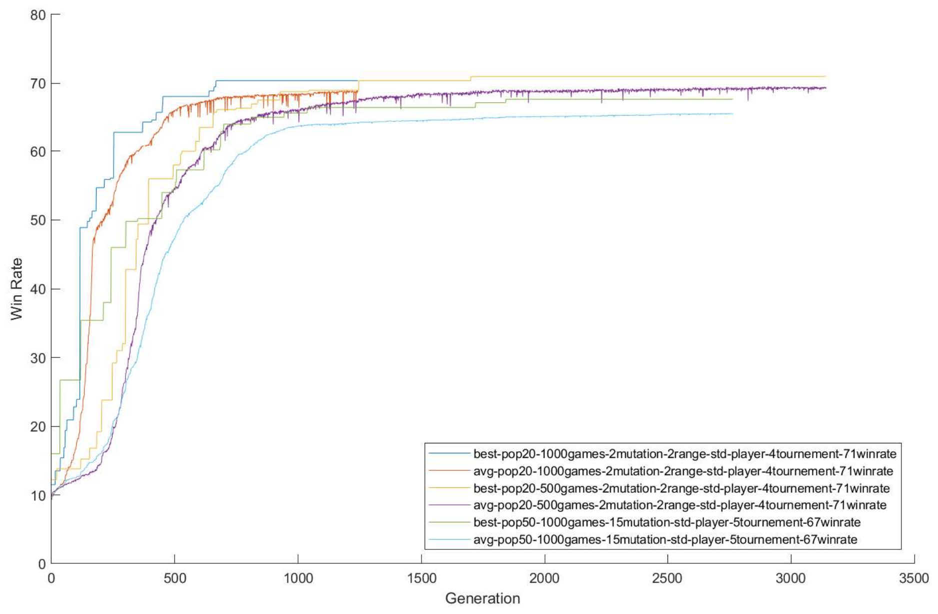

Fig. 1. Showing the best specimens fitness and the average fitness for each generation for multiple populations when training against semi-smart players.

The chromosomes or weights, for the best specimen is shown in table 2. Some of the weights makes sense to be weighted high, so as not making suicide moves and escape home. It is however interesting to see that gain of safety isn't seemed as important as if loosing safety. Also the risk of getting killed isn't as important as killing another player or moving a further distance. It is also interesting in seeing how high it weighs moving back and forth in the goal stretch without

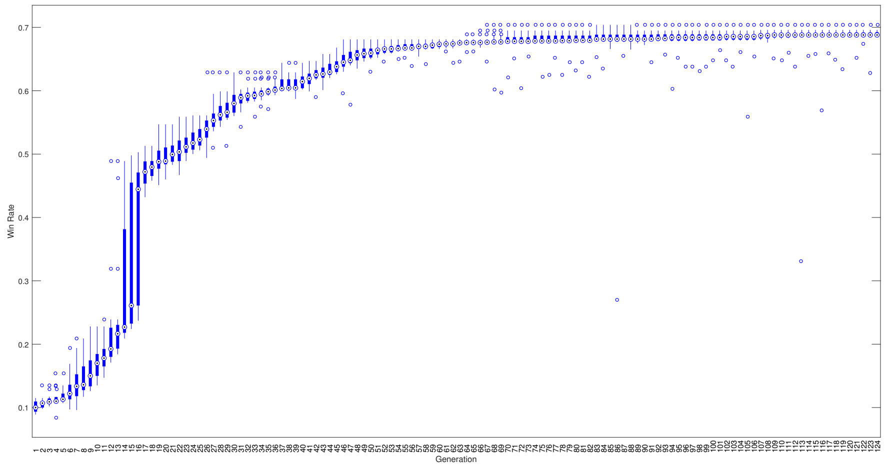

Fig. 2. Showing the box-plot of every 10th generation of the best performing population when training against semi-smart players.

getting to the final goal, however it does make sense to not waste moves doing it. Most of the populations converged around the same average win-rate of 64-66%

Parameter	Value
Moved distance	0.4010978
Gain of safety	0.6720623
Loss of safety	3.774511
Can escape home	17.25477
Can kill another player	4.38893
Move is suicide	18.55596
Risk of getting killed	0.3250972
Move to goal stretch	11.13567
Move to final goal	6.814829
Move within goal stretch	20.42609

Table 2. Chromosomes for the best specimen archived.

against semi-smart players. The best specimen was selected and tested against both semi-smart players, random players and the competitors method. It was always the trained player against 3 of the same opponents. The test was 10000 games repeated 100 times to get an accurate converged win-rate. The results is shown for both this papers trained player and the opponents in tables 3 and 4.

Opponents	Highest win-rate	Mean win-rate	Std. Deviation
Against Randoms	76.97%	75.89%	$1.7891e^{-05}$
Against Semi-smart	67.49%	66.06%	$2.4660e^{-05}$
Against Competitor	74.07%	73.04%	$2.1271e^{-05}$

Table 3. This papers trained player's performance.

Opponents	Highest win-rate	Mean win-rate	Std. Deviation
Randoms	8.51%	7.89%	0.0025
Semi-smart	12.65%	11.89%	0.0035
Competitor	11.32%	10.47%	0.0028

Table 4. The opponents performance.

4 Analysis and Discussion

The results from table 3 shows that the best player's win-rate has a mean of 66.06% and a standard deviation of $2.466e^{-05}$ against the semi-smart opponents. This is a good mean win-rate with little deviation. It turns out the best opponent was the semi-smart players as the other implemented method performed rather poorly.

To statically confirm the mean win-rate is better than the best win-rate of the best opponent, a one-sided t-test was conducted. To ensure that the test data was normal distributed, a normal probability plot was created and a Kolmogorov-Smirnov(KS) test performed.[5] The normal probability plot can be seen in figure 3, and shows the data is reasonable normal distributed. The KS-test also failed to reject that the data was normal distributed at 5% significance level.

The t-test was conducted with a significance level of 1%. The null hypothesis is that the trained players mean win-rate is less or equal to the semi-smart player. The result of the t-test is shown in table 5. The p-value from the t-test shows that the null hypothesis is very strongly rejected and thus the mean win-rate from the trained player must be higher than the mean from the semi-smart players. A t-test was also conducted for the other opponents for comparison, but as they all performed worse than the semi-smart player it is quite expected they all strongly reject the null hypothesis.

The fitness of the player is set to purely be the wins. This limits the solution as specimens that was very close to winning is seen as bad, even though they might have some good chromosomes for mating. Likewise does a specimen that wins overwhelming get judged with same fitness as a specimen that barely won. This isn't ideal and would be one of the improvements that could likely enhance the performance, by using a different fitness function.

Based on these statistics it can be concluded that the trained player in this paper performs significant better than the tested opponents. The reason for not getting a higher win-rate than what was obtained is believed to be in the representation of the problem. The GA worked as expected but converged at around 66% against semi-smart players. If the representation of the problem had been enhanced so the states and weights had been better at adapting as the game progressed it is likely able to reach a higher win-rate.

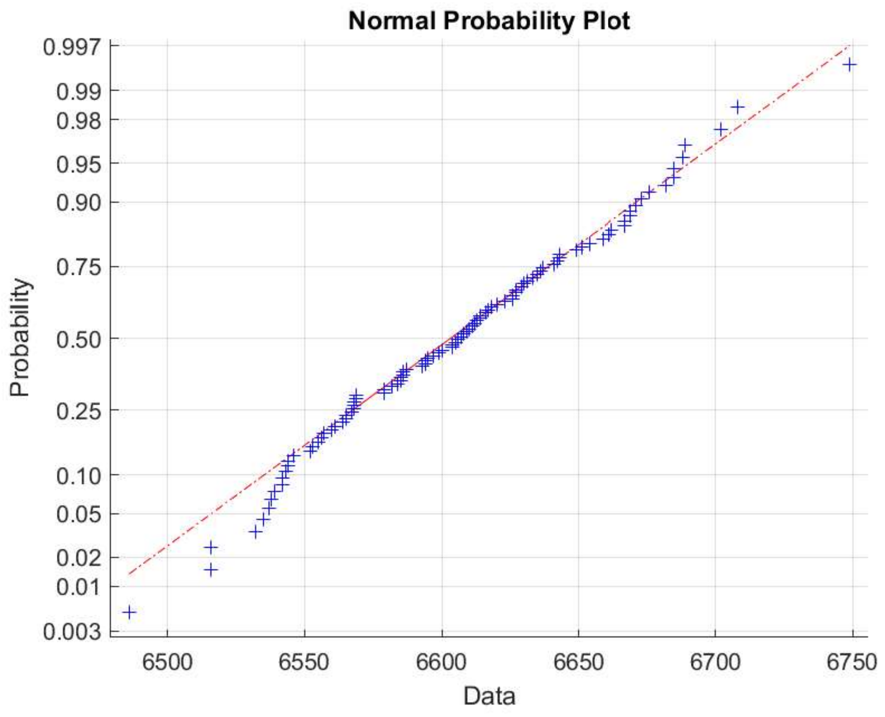

Fig. 3. Normality probability plot showing the results is reasonable normal distributed.

Opponents	Hypothesis Result	p-value
Against Randoms	strongly reject H_0	$3.0854e^{-207}$
Against Semi-smart	strongly reject H_0	$2.2668e^{-186}$
Against Competitor	strongly reject H_0	$2.1455e^{-197}$

Table 5. Table showing results of performance testing against different opponents.

5 Conclusions

It can be concluded that the trained player with the implemented GA had better performance than the opponents it was up against. It was proven to have a mean win-rate of 66.06% against the best opponent, the semi-smart player. It can be concluded that the training method of the GA gives an high convergence fitness. To reach higher level of win-rate a better representation of the problem is seemed necessary.

The next step would be to investigate how well the diversity is kept throughout the generations and if the convergence rate can be fasten while using fewer generations. It would also be worth looking into a better representation of the problem that could enable reaching higher win-rates. Investigating what influences who starts a game would also be interesting as it quite possible have a influence which player starts a round. Investigating how the fitness function can be improved to distinguish between overwhelming wins and losses could also prove to enhance the results, as that would label specimens more accurately and help make combinations of good chromosomes.

6 Acknowledgements

I would like to thank Per Breum for providing this player based on Q-Learning as a competition to test against. I will also like to thank Kristian Mller Hansen for proof reading this paper.

References

1. A.J. Umbarkar, P.D. Sheth: Crossover operations in genetic algorithms: A review: ICTACT journal on soft computing, October 2015, Vol. 06, Issue: 01: https://pdfs.semanticscholar.org/6888/7a8e96440f9e69c015c923a83455478f0290.pdf
2. Marek Obitko, Walter Ptzold: Genetic Algorithms Recommendations: Hochschule fr Technik und Wirtschaft Dresden (FH), 1998. http://www.obitko.com/tutorials/genetic-algorithms/recommendations.php
3. Miller, Brad; Goldberg, David: Genetic Algorithms, Tournament Selection, and the Effects of Noise: Complex Systems. 9: 193212 (1995)
4. S. Gotshall, B. Rylander: Optimal Population Size and the Genetic Algorithm: https://pdfs.semanticscholar.org/0a7b/63cfe4b40741452692366e2047db43b467c7.pdf
5. Wikipedia: Kolmogorov-Smirnov Test. https://en.wikipedia.org/wiki/Kolmogorov-Smirnov test (2018)