AN EVOLVING LUDO PLAYER BASED ON NEURAL NETWORKS, BACKPROPAGATION AND GA

Kasper Rytter Sæderup
The Mærsk Mc-Kinney Moller Institute (MMMI), University of Southern Denmark (SDU), Odense, Denmark
ksaed06@student.sdu.dk

Keywords: Genetic Algorithm, Neural Network, Ludo, Backprobagation, AI, Automated game player

Abstract: To compete in a tournament of Ludo, an automatic evolving Ludo player has been developed. The Ludo player is based on a neural network trained with backpropagation and uses a genetic algorithm to evolve and optimize the chromosome of the Ludo player during generations. The Ludo player is compared to a Ludo player using a combination of reinforcement learning and a genetic algorithm.

1 INTRODUCTION

This paper has been written as an exam paper for the course AI2 on the University of Southern Denmark. There is to be developed an automatic player for the game of Ludo. The player should be based on some learning tools used in the field of artificial intelligence covered in the AI2 course, such as supervised learning, e.g Artificial Neural Networks and unsupervised learning in terms of self organizing networks, e.g. Kohonen Maps or Growing Neural Gas.

This paper presents an evolving Ludo player based on a neural network, backpropagation and a genetic algorithm.

To compare the player's ability to perform in a game of Ludo, the player is empirically compared to a simple Ludo player based only on weighted inputs and another Ludo player based on artificial intelligence; reinforcement learning using Q-learning and a genetic algorithm.

1.1 Ludo

Ludo is a simple board game where 2-4 players have 4 bricks each. The players are doing turns at throwing a six sided die and moving a brick if possible. All of the players' bricks start in the players' respective start area, see figure 1. The winner is the first player who manages to get all four bricks into the goal in the middle of the board.

During the game different factors determine whether a move is good or bad. The most important rules of the games are: To get one brick out of the start area, into play and start moving around the board, the player has to get a 6 when rolling the die, if a brick is already in play, it can be moved the number of fields that the die shows.

If a single brick is placed on a field which is not a globe, the safe area or the goal, there is a risk of being hit home by an opponent, meaning that the brick has to move back into the start area. If a brick lands on a star, the brick moves to the next star, if not the star is located next to the player's goal zone, in which case the brick moves directly into goal.

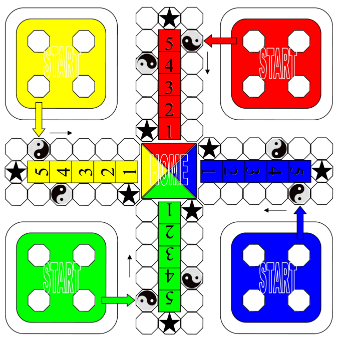

Figure 1: The Ludo board as it is illustrated in the simulation.

A simulation environment for the Ludo game is available for the course on which the implementation of the Ludo player and the experimentation results are based. In the simulation the three best players receives points after the end of a game. The points system in the simulation of the game is that the winner is rewarded 3 points, 2nd place gets 2 points, 3rd place gets 1 point and the looser gets 0 points.

The main points of developing an automatic Ludo player is to represent the state of the game in a way not too complex but still useable, analyze the brick situation using the state representation and chose which brick to move according to that input.

2 A Ludo player based on neural network and a genetic algorithm

The automatic Ludo player developed in this paper is based on a neural network where the desired outputs are chosen using a genetic algorithm (NNGA player).

2.1 The neural net

The NNGA player is using a feedforward neural net with 9 input neurons, a single hidden layer with 3 neurons and one output neuron, implemented using a weight matrix (Heaton, 2008). The 9 inputs represents the state of the game for one brick and the same neural network is used for all four bricks.

The neural network uses the sigmoid activation function, see equation 1, hence the output of the network is a floating point number between 0 and 1.

$$P(t) = \frac{1}{1+e^{-t}} \tag{1}$$

The neural network is trained with a backpropagation algorithm that runs, until a defined maximum error for the outputs of the neural network has been reached.

The neural network inputs:

The state representations consists of 9 inputs, the meaning of which are described in table 1.

Table 1: The inputs for the neural network

Input	Description
1	The brick can get home
2	The brick is able to hit an opponent home
3	If the brick moves, it will be hit home
4	The brick will land on a star
5	The brick will land on a globe
6	The brick can get out of the start area
7	If the brick moves, it will have an enemy near
8	If the brick moves, it will get into the safe area
9	The brick is in the safe area

Backpropagation

During the game 19 different combinations of the 9 inputs can occur, i.e. landing on a star and hit an opponent home. A truth table for the neural network is implemented to define the 19 states and an output for each as a floating point number between 0 and 1. The truth table is then used in the backpropagation algorithm to analyze and modify the weight matrix accordingly, so the wanted outputs, becomes the actual outputs, eventually with a small error. For every iteration of the backpropagation algorithm the error is found, and stops when a defined maximum error of 0.01 is found. That the algorithm stops at a desired maximal error, minimizes processing time compared to having a fixed number of iterations to perform.

Determining which brick to move

During the game, the Ludo player should choose which brick to move. This is done by finding the inputs and computing the output of the neural network for each moveable brick. The brick which get the highest output is the brick to move. If more than one brick has identical outputs, moving the one instead of the other should result in an equally good situation and therefore which brick to move is chosen randomly.
The code for determining which brick to move, works like this:

1. For each moveable brick.
   (a) Find the state using the 9 inputs from table 1.
   (b) Compute the output of the neural network.
2. Choose to move the brick which got the largest output.

The desired outputs for the neural network was manually chosen before the genetic algorithm was implemented (NN player). The NN player is compared to the NNGA player during the experiments in sections 4 and 5 to see that the evolved Ludo player actually learns.

2.2 The genetic algorithm

To optimize the Ludo player a genetic algorithm is implemented.
The genetic algorithm aims to find the optimal solution to a problem based on Charles Darwin's ”*Sur-*

vival of the fittest", in this case the fittest players is the Ludo players who is rewarded the most points after a series of games.

The chromosome

The genetics of the Ludo player is determined by a chromosome, the chromosome for the NNGA player is the desired outputs of the neural network in the 19 possible states, thus the chromosome has 19 genes. The genetic algorithm starts the evolution by initiating four players based on random numbers in every gene, as a floating point number between 0 and 1, in the chromosome. The four players plays Ludo for 1000 games. The "suitability" and privilege to mate is given to the two players having the most points, and the two players becomes parents to a new Ludo player, who substitutes the overall loser.

Mating two Ludo players

Mating of the Ludo players consist of two processes; crossover and mutation (Heaton, 2008).
The crossover method, creates a new chromosome with 19 genes, then finds a cut point randomly, and takes the genes from the winner that is before the cut point. The genes from the player who got the 2^{nd} most points is placed in the offspring's chromosome after the cut point, as illustrated in figure 2.

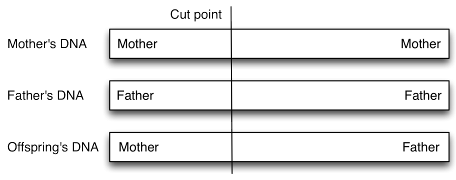

Figure 2: Crossover function used when mating two chromosomes to produce a single child

Mutation of a chromosome

After the crossover, 40 percent of the offspring's genes is mutated with a random amount between -0.4 and 0.4. Increasing the mutation percentage and the gene mutation rate could speed up the evolution of the Ludo player, on the contrary the offspring also differs significantly more from its parents and therefore smaller mutation percentage and gene mutation ensures greater possibility that the offspring becomes equally good or better than its parents. Whether or not the offspring becomes a better Ludo player than its parents is unknown due to the mutation, but natural selection will determine the faith of the offspring.

Pseudo code for the genetic algorithm:

```
1. Initialize four players with random values in the
   19 genes of the chromosome.
2. For every iteration.
   (a) Play 1000 games of Ludo
   (b) Mate the two players with the most points
       (crossover and mutation).
   (c) Train the offspring with the backpropagation
       algorithm.
   (d) Substitute the player with the fewest points,
       with the newborn Ludo player.
3. Continue with (2) until the wanted number of generations is reached.
4. Play 1000 games of Ludo.
5. Declare the overall winner as the player winning
   the last 1000 games and print the chromosome.
```

The specific NNGA player to be used for the player comparisons and the all against all tournament was evolved over 16500 generations and has the chromosome: {0.97, 1.0, 0.81, 0.06, 0.67, 0.36, 0.23, 0.39, 0.77, 0.29, 0.41, 0.32, 0.95, 0.91, 0.51, 0.69, 0.63, 0.18, 0.51}. As described before the genes in the chromosome means how much it want to move the brick if it is in specific state, i.e. the second number is the highest meaning that if the player can hit an opponent home, that is what the NNGA player is going for.

3 Ludo players for comparison

To analyze the performance of the NNGA Ludo player it will be compared to the pre-implemented SemiSmart player and to a Ludo player using Q-Learning.

3.1 SemiSmart player

The implementation of the SemiSmart player is delivered with the Ludo game. The SemiSmart player uses a simple representation of the game for each brick, comparable to the state representation used in the NNGA player. The SemiSmart player uses 5 inputs which represents 5 states, not considering combinations of the input. The inputs are weighted, and the brick having the highest input, is the one the SemiSmart player moves.

3.2 Ludo player based on Q-learning and a genetic algorithm

Another Ludo player based on artificial intelligence is used for comparison of the NNGA player. The Ludo player is using the reinforcement learning method Q-learning and a genetic algorithm (QLGA player).

The QLGA player uses a map of Q values which is used to decide which brick to move. The map contains all possible states during a game, each defining Q values for four actions; one action per brick to move. The brick to move is the one whose respective Q value is the greatest.

The state representation used in the QLGA player is based on the state representation of the NNGA player, shown in table 1, but is extended to contain all four bricks, hence being a 36 bit binary number, due to the fact that all inputs are either 0 or 1. The representation is therefore that brick 1 is the first 9 bits, brick 2 is bits 10-19, etc.

Initially the QLGA player's map of Q values is filled with zeroes, and the player has to explore the game during a significant number of games. To update the Q values, the player is getting a reward for the action taken. For the QLGA player, the reward is delayed until the player has its next turn, then depending on the new situation the player is either rewarded or punished and the Q value of the chosen action is updated according to the Q-learning rule (Kaelbling et al., 1996). The values of the rewards is decided using a genetic algorithm, thus the rewards or punishments to give, is the chromosome for the QLGA player. The genetic algorithm is then used similar to the one described for the NNGA player in section 2.2 using natural selection. Because the QLGA player is more complex than the NNGA player, the QLGA player also takes significantly more time to develop, therefore the QLGA player used is based on no more than 81 generations (Derakhshan, 2010).

The QLGA player has more information about the state and is therefore a potentially greater Ludo player than the NNGA player. The experiments shows though, that the NNGA player is the better player of the two.

4 EXPERIMENTAL SETUP

This section describes four experiments:

- an experiment concentrating on the processing time for the backpropagation and the genetic algorithm for the NNGA player.
- comparisons between the NN player and the NNGA player, the NNGA player and the SemiSmart player, and the NNGA player compared to the QLGA player.
- evaluation of the fitness of the evolving NNGA player along the generations in the genetic algorithm.
- the last experiment has been provided and consists of an all against all tournament with all the Ludo players developed by the participants of the course of AI2.

4.1 Processing time

The processing time is very dependent on the hardware on which the program has been tested, therefore the results of this experiments can only be used to get an idea about the performance of the implementation. The experiments is performed by writing out the amount of milliseconds the backpropagation algorithm uses when a new Ludo player is born, this is done for every generation over 100 generations and the mean is computed. The same setup is performed to calculate the average processing time for every generation in the genetic algorithm while evolving a NNGA player.

The computer on which the experiments is performed is an:

- Apple MacBook Pro laptop running Mac OSX 10.6.2
- CPU: Intel Core 2 Duo, 2.53 GHz
- RAM: 4 Gb
- Java version: 1.6.0

4.2 Player comparisons

To find out whether or not the NNGA player is a better Ludo player than the NN player, for which the output of the neural network has been chosen manually, a comparison between the NNGA player and the NN player will be performed. To evaluate the NNGA player's fitness of playing Ludo, the NNGA player will be compared to both the SemiSmart player and the QLGA player.

All player positions are not equally good, the player who starts the game has the advantage, which in this implementation is the yellow player. The blue position has the hardest case of winning the game, due to being the last one to roll the die. Therefore when comparing two players, the players play against each other with two instances of each player, the one player plays as the yellow and the blue player, the other player plays as the green and the red player. Playing 10.000 games was found not to be reliable, therefore the comparisons is based on 10x10.000 games, and the results of the 10 rounds consisting of 10.000 games are then used as samples for a statistical t-test (Cohen, 1995), to determine which Ludo player is the best.

4.3 The evolution of the NNGA player

To evaluate the evolution of the NNGA player, the fittest player in each generation have been playing 10.000 games of Ludo against 3 SemiSmart players, where the fittest player was playing at the yellow position. In this evaluation the total score after the 10.000 games of Ludo is plotted for each generation. The parameters used for the plot is 40 pct. gene mutation and mutation of the genes inside the interval (-0.4, 0.4).

4.4 All against all tournament

The all against all tournament consists of a tournament between 22 automatic Ludo players, most of them being Ludo players developed by the participants of the course AI2, including the NNGA and the QLGA players, and some of the Ludo players being pre-implemented, such as the SemiSmart player.

All players have played 42 matches each, each match consists of 1000 rounds of Ludo. Due to the fact that not all player positions are equally good, the players have played twice against all of the other players, being positioned at both yellow-blue positions and red-green positions.

5 RESULTS AND DISCUSSION

5.1 Processing time

The results for the simple experiment of measuring the processing time of the backpropagation and the genetic algorithms is shown in table 2. The table shows the average, minimum and maximum processing time in ms for each algorithm.

Table 2: Results for the average processing time of the backpropagation and the genetic algorithm

Algorithm	Processing time	Min/Max value
Backpropagation	499 ms	18 / 1637 ms
Genetic Algorithm	1748 ms	1168 / 2968 ms

Since the backpropagation algorithm varies in the amount of iterations, because it stops when it reaches a wanted maximum error, the processing time of the algorithm varies too. Because the backpropagation algorithm also runs in every generation of the genetic algorithm, this variation of processing time is also influencing the measurement of the processing time of the genetic algorithm. With this implementation it is possible to perform about 35 generations per minute.

5.2 Player comparisons

The NN player in which the ”chromosome” has been chosen manually by the developers, is compared to the NNGA player which has been evolved over 16500 generations using a genetic algorithm. The results is shown in table 3.

Table 3: The NN player against the NNGA player

NN player	NNGA player
2694	3300
2652	3348
2708	3292
2718	3282
2711	3289
2752	3248
2673	3327
2699	3301
2663	3337
2729	3271

The paired t-test gives a t-value of $t = 30.9118$. Since t > 3,250 it’s 99,5% certain that the NN player is not better than the NNGA player.

The comparison between the NNGA player and the SemiSmart player is shown in table 4.

Table 4: The SemiSmart player against the NNGA player

SemiSmart player	NNGA player
2349	3645
2455	3545
2463	3537
2428	3572
2444	3556
2403	3597
2472	3528
2407	3593
2413	3587
2416	3584

The unpaired t-test gives a t-value of $t = 72.5585$. Since t > 3,250 it’s 99,5% certain that the SemiSmart player is not better than the NNGA player.

The comparison between the NNGA player and the QLGA player is shown in table 5.

Table 5: The QLGA player against the NNGA player

QLGA player	NNGA player
2728	3266
2731	3269
2742	3258
2697	3303
2673	3327
2750	3250
2695	3305
2747	3253
2741	3259
2663	3337

The unpaired t-test gives a t-value of $t = 39.3163$. Since $t > 3{,}250$ it's 99,5% certain that the QLGA player is not better than the NNGA player.

The NNGA player has been compared to the NN player, the SemiSmart player and the QLGA player. The performed t-tests have shown that none of the players has proven to be better than the NNGA player.

5.3 The evolution of the NNGA player

The figure 3 shows a plot of the score the fittest player in every generation obtained in 10.000 games of Ludo against three SemiSmart players. Figure 4 shows a close-up of the first 150 generations of the same experiment.

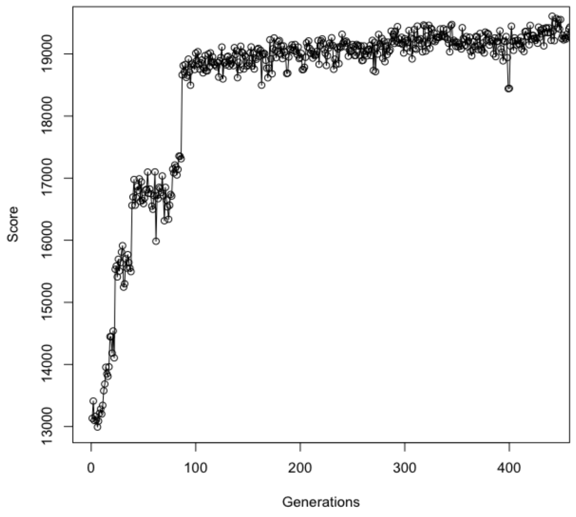

Figure 3: The evolution of the NNGA player over generations

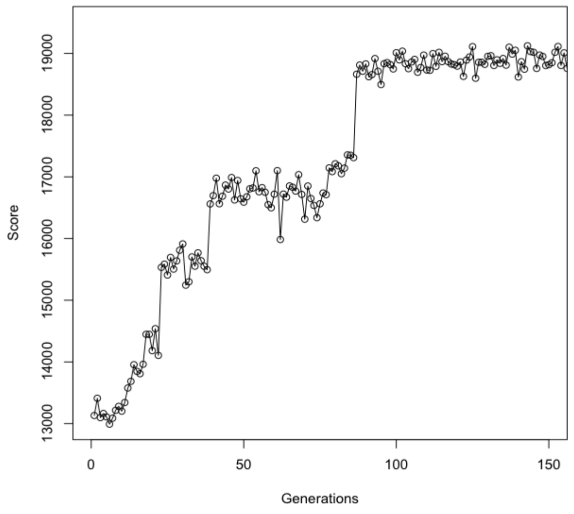

Figure 4: A close-up of figure 3

The plot shows that the Ludo player gets significantly better during the first 100 generations, and afterwards becomes more stable, but still slightly improving. The evolution of the Ludo player looks like happening in steps, which is due to the fact that not all of the new-born Ludo players in each generation is a better Ludo player. In the cases where the fitness value drops, the reason is that luck has a certain amount of effect on the players' score, i.e. if a player gets a significant amount of 6's, the player's score will most likely go up.

5.4 Tournament results

Table 6: Results of the all against all tournament, the NNGA player is placed 3^{rd} (player 5), the QLGA player got the 12^{th} place (player 3).

Player	No. of matches won
14	42
9	40
5	37
10	35
15	34
1	32
8	29
12	28
13	27
4	24
7	21
3	21
16	18
6	15
2	14
17	8
11	6

The NNGA player was player number 5 in the tournament, resulting in a 3^{rd} place. The QLGA player was player number 3 in the tournament, being placed number 12 in the tournament.

This shows that the NNGA player is a fairly good Ludo player, on the contrary there is still room for improvements due to the fact, that two other players in the tournament was placed higher than the NNGA player.

The QLGA player has a more describing state representation, being able to chose which brick to move based on combinations of how the other bricks are located, and should therefore in theory be able to beat the NNGA player. The reason this is not the case, is probably due to the fact that the QLGA player is more complex and it takes significantly more time to train and evolve than the NNGA player. A probable reason that the QLGA player is not able to beat the NNGA player is because of the limited amount of generations the QLGA player has been through.

5.5 Discussion about the implementation of the NNGA player

The chromosome for the NNGA player was chosen because the chromosome is itself meaningful because if one knows which gene means what, one can imagine how well the player is at playing Ludo. A negative effect of the chosen chromosome is that the neural network has to be trained every time a new Ludo player is born which is not the case if the chromosome was the weight matrix instead. Therefore the time it takes to train the neural network could have been spared in every generation when evolving the NNGA player.

The used chromosome is only applicable because the inputs are either 1 or 0. In case e.g. the inputs were floating points between 0 and 1, the chromosome would be too large and therefore not having the advantage of being easily deciphered. This would be the situation if e.g. the input of the enemy near gene, should indicate how close the enemy is.

In this implementation the neural network is held static, but if the neural network should also be generated during the genetic algorithm, the chromosome should also contain an representation of the neural network.

The population of the genetic algorithm is only four players, therefore the genetic algorithm possibly finds a local maximum. To explore further the algorithm can be run in three different ways; for many generations with a high rate of gene mutation, several times and compare the results or a kind of tournament could be implemented e.g. using islands where populations of four could develop a player, then through tournaments the best overall player could be found.

In each generation during the genetic algorithm Ludo is played 1000 times. Due to the significance of luck affecting the outcome of the game, 1000 games is possibly not enough to ensure the winner is actually the better Ludo player. Therefore the reliability of the genetic algorithm would probably be better if more games is played during every generation.

6 CONCLUSION

A Ludo player using techniques from the field of artificial intelligence has been presented. The Ludo player uses a combination of a neural network trained with backpropagation and a genetic algorithm to optimize the chromosome of the Ludo player. The genetic algorithm has proven its worth and has optimized the presented Ludo player to beat both a SemiSmart player, a player developed using reinforcement learning with a genetic algorithm and the presented Ludo player got a 3^{rd} place in the all against all tournament of the AI2 course.

ACKNOWLEDGEMENTS

The NNGA and QLGA Ludo players has been developed in companionship with Daroysh (Danni) Derakhshan.
Thanks to Rasmus Rosenqvist Christensen for performing the all against all turnament. We are happy to have come at a third place in the turnament with the NNGA player.

REFERENCES

Cohen, P. R. (1995). *Empirical Methods for Artificial Intelligence*. The MIT Press, 1st edition.

Derakhshan, D. (2010). *APPLIED REINFORCEMENT LEARNING WITH USE OF GENETIC ALGORITHMS*.

Heaton, J. (2008). *Introduction to Neural Networks for JAVA*. Heaton Research, Inc, 2nd edition.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. Journal of Artificial Intelligence Research 4 (1996) 237-285.