Co-evolutionary Backgammon Player: Learning Through Self-play

Han BAO

April 24, 2017

6514879

School of Computer Science
University of Nottingham

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature

Abstract

Backgammon is known as one of the oldest board games which has simple rules while require lucks and strategies to operate. Artificial Neural Network (ANN) is a widely used techniques in machine learning which could make game players more intelligent in a game. It works through modify the weights of ANN gradually through self-play until reaching the optimum condition, such refining process is called co-evolution.

This dissertation will present, design, and implement a Backgammon self-player in Java to obtain the best strategy representation with millions of tests. The aim is to generate an ANN capable of playing backgammon, using the performance of other backgammon game players that adopt a different strategy for comparison.

The outcome is that ANN player kept making significant progress when fighting with other players after million times of self-plays. Although it haven't reach 100 percent successful rate as TD-Gammon, it manifest itself that one can learn and make progress through self-play, which also demonstrates hill-climbing method proposed by Pollack & Blair is effective.

Acknowledgements

I would like to extend thanks to my supervisor of this project, John Cartlidge, for his support and for his highly efficient replies to my emails and questions.

I am grateful to Eugene Ch'ng for his summer research that I had the opportunity to learn more knowledge in Machine Learning. Also thanks to my parents for their support along the way. Thanks to my roommates and classmates for being there. Without these people, this work would not have been possible.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Project Description

The intention of this project is to create an automatic backgammon player through self-learning which can make better interactive intelligent moves. In traditional board games with AI, the programmers set up the rules and provide strategy algorithms with a standard to judge the whole board. However, the self-learning player could learn the strategies during practice, so this approach does not require humans to provide additional knowledge, just the rules of the game.

1.2 Project Aim & Objectives

The aim of the project: This project will attempt to figure out how to design a backgammon player that has been developed by Pollack and Blair (1997)[12], using an ANN trained through self-learning, beginning with a simplified version of backgammon, and moving on to a more realistic version of the game.
The main of objectives of the project are as follows:

- Design and implement Backgammon board game in Java and document.
- Do research on former Backgammon players and Artificial Neural Networks (ANNs).
- Create an ANN as described in the requirements specification mentioned in the project description.
- Fully test the system against other Backgammon players using a random algorithm or a little artificial intelligence.

Firstly, I will write a demo in Java to become familiar with backgammon and read related papers on backgammon and ANN. Secondly, I will apply the optimal doubling written by Keeler and Spencer (1975)[8] and the optimal strategy written by Thorp (1988)[20], to the demo. Then the demo should use self-learning with ANNs. Finally, I will test this Player and discuss the results. How to progress through self-play is the key point.

1.3 Motivations

Tesauro[17] reported that TD-Gammon, its exclusive training through self-play rather than tutelage, helps it explore strategies itself. The unorthodox strategies had a significant influence on backgammon that provided computer scientists a thought-provoking solution. Pollack and Blair[11] were against Tesauro's conclusion that the hill-climbing approach (no reinforcement learning necessary) could also be successful in backgammon training. Tesauro' comments in 1998[19] answered this question and pointed out the lack of value network. So based on this debate, I want to build an ANN player like Pollack and Blair's, using co-evolutionary algorithm during self-play. More details could see the Chapter 2 Background.

1.4 Document Content & Scope

This project will adopt the Waterfall engineering methodology and will thus proceed in a linear fashion. More specifically, the project will proceed with sequential steps, where each step will solve a particular problem in the process. The structure will be as follows:

Background:

Introduce Backgammon and the debate between Tesauro and Pollack and Blair.

a) Backgammon History: Introduce the history and how intelligent computer player involved in Backgammon.

b) Debate between Tesauro and Pollack and Blair : Introduce their main points in each report.

c) Methodology: Introduce Self-Play, Artificial Neural Network and some research referring to ANN.

e) A Short Summary of Research: Summarize the technology, ideology and working methodology.

Requirements & Specifications:

Gives more detailed information on this model and on what needs to be prepared before training.
a) Feasibility Study: Analyse the problems based on the suitability of previous technologies.

b) Non-functional Requirements: Analyse the compatibility, user-friendly, feasibility and playability.

Backgammon Game and Player Design:

A section to introduce the design of different kinds of backgammon player and their specific components.
a) Game Design: The frame design and functions design of backgammon game.

b) Human Player Version: User plays games through the input and there is no strategy.

c) Random Machine Player Version: The design of a random and automatic player.

d) Simple Machine Player Version: It is an automatic player with the more intelligent strategies.

e) ANN Player Version: A high level overview of the whole ANN design and its strategies. Low level attention paid to how each individual component in the system will work.

Implementation:

Describe how game and each player implement.
a) Game: Implement the backgammon game based on the design and rules.

b) "human": According to design, implement Human Player.

c) "random": According to design, implement Random Machine Player.

d) "simple": According to design, implement Simple Machine Player with intelligent strategies.

e) "ANNP": According to design, set up ANN model and get the best ANN Player after millions of training.

e) "Competition (A VS B)": The implementation of a competition and the reason why choose A VS B. In addition, it talks about how to solve the predominance of first player.

Evaluation:

Testing the system and analysing and discussing the results.
a) Testing: The testing of each player and the result.

b) Reflection: The chapter talks the review of whole project focusing on the management and implementation, the reflection on self evaluation and improvement and the further work in the future.

Conclusion:

Based on the testing result and ANN model, give a simple conclusion about the debate between Tesauro and Pollack & Blair. Reflection on the process and project. Finally, summarize the advantages and disadvantages of ANN model and co-evolutionary self-play.

Appendices:

Additional information about testing process and complimentary works completed for this project.

2 Background

This section focuses on details of backgammon game, the debate between Teasuro, Pollack and Blair. In addition, I would introduce the methodology mentioned in other researcher' report or my project.

2.1 Backgammon History

Backgammon is one of the oldest board games known. It has simple rules and elements of luck and elements of strategy. Commentators, Chuck for instance, have come to regard the 1970's as the "heyday of backgammon". In addition, people found more ways for computer players to learn intelligent strategies in the 1990s (Driver, 2017)[10]. This board game not only depends on rolls of the dice, but also needs a global view of the whole board rather than on only each small movement (Rosenfeld, 1974)[2]. Due to the relatively simple rules compared to other board games and the less complex implementation of algorithms, since backgammon is the perfect game for machine learning as it rewards skill and contains an element of luck. Consequently, the outcome of the game is in doubt until the last roll of the dice. Before exploring more detail of backgammon self-play, I would like to first introduce the special names involved and the basic rules of Backgammon.

2.1.1 Names in Backgammon Game

There is one board and two players play with two dices. In general, the board is divided into four parts, including outer board, home board, bar and home. Firstly, we need to familiarize ourselves with the terminology:

- Stones: pieces on the board.

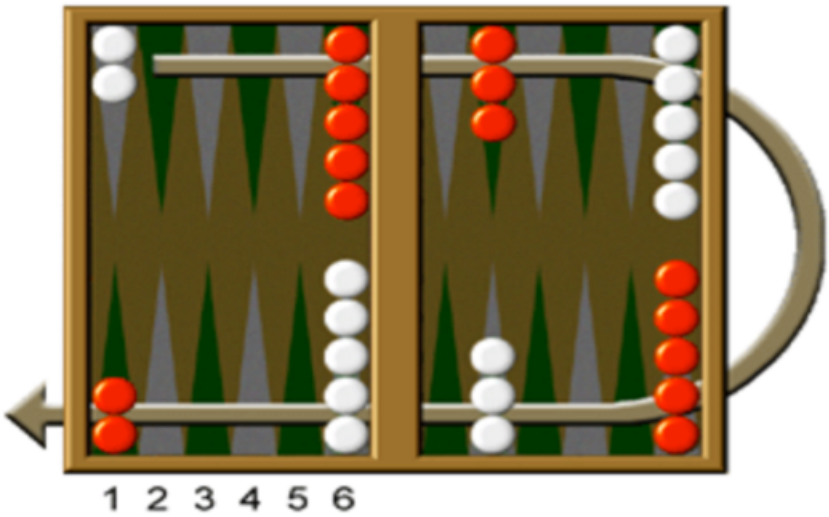


Figure 1: White Part and Black Part on the board. White's direction is arrow's direction and Black's direction is opposite.[0]

- Board: The whole board is split into a left part and a right part, and also split into a front part and a bottom part. The northern two parts are the Black Part (this depends on the stones' initial location) and the southern two parts are the White Part (see Figure 1).
- Bar: Located in the centre of the board (see Figure 2).

[0]This figure resource is www.bkgm.com[1]

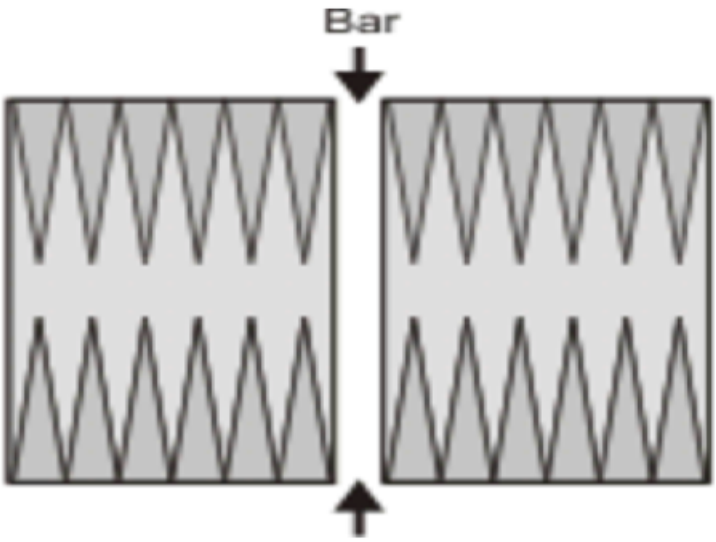


Figure 2: Captured stones are put in the “Bar”. [0]

- stonebar: Actually, the name should be “Point”. However, this is easily misunderstood as the stones’ location, so I use “stonebar” to mean the 24 columns in the whole board. Each board has six stonebars (see Figure 3).

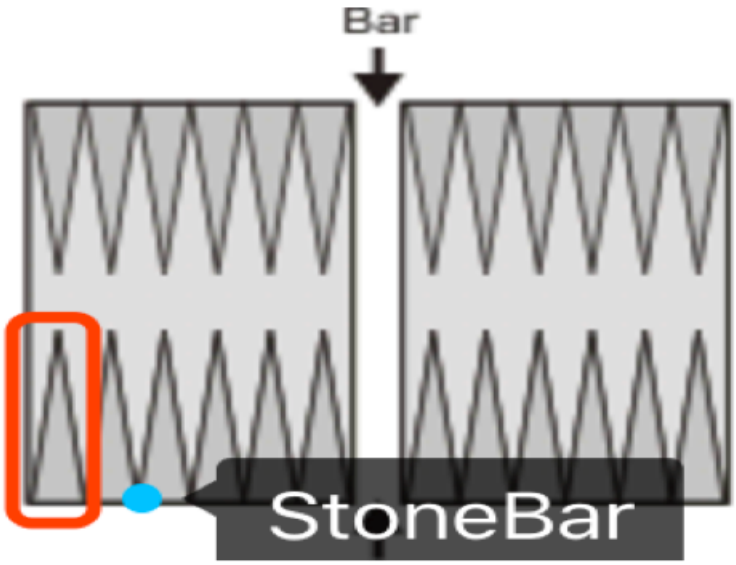


Figure 3: A board comprises 24 “stonebars”.[0]

- Blot: When there is only one stone on the stonebar, this stone is “Blot” (see Figure 4).

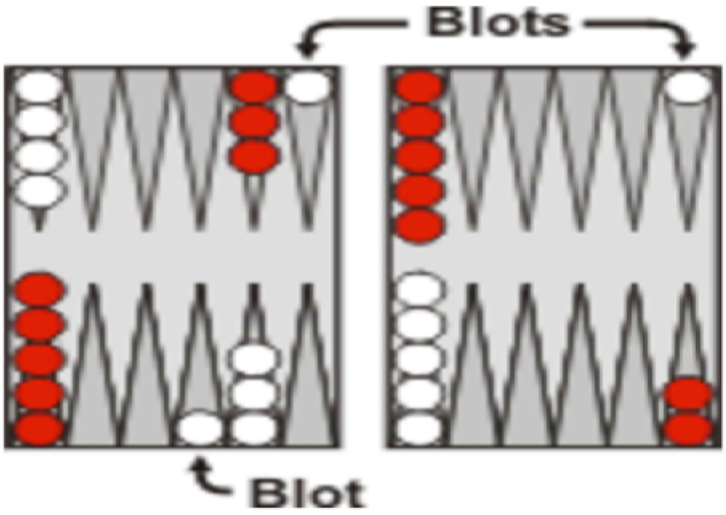


Figure 4: "Blot" means only one stone on "Stonebar"[0]

- Hit: When an opposing stone is moved to a "Blot", this movement is named "Hit". Then this blot would be moved to the Bar.
- Open: If there is no stone on a stonebar, this stonebar is open for both colours. If there are more than two stones of one colour located on a stonebar, this stonebar is open for this colour but not open (or closed or made) for the other colour. Only the destination is open for the stone, the stone could move to this destination.
- Closed or Made: Opposite to "Open".
- Bear off: Stones move from the inner board and get home.
- Shut out: If there are no "open" places for the stones, the turn is over.
- Start the game: Each player has 15 stones placed on four bars. Then, each player has two dice and rolls them to choose the colour that moves first. If the Black dice sum number is bigger than that of the White dice, Black moves first. Otherwise, White moves first.
- Move the stone: If the numbers on the rolled dices are same, the player needs to move four times. Otherwise, he only needs to move two times.
- Hit the stone: If only one stone is left in a bar and the other colour player could move his stone to hit this stone, this stone would be moved to the centre of bar. Next time, the player has to move this stone first from opposite the inner home board.
- Bear off: Only when all a player's stones are located in inner board, can the player move the stones to bear off.
- End the game: Once a player's stones are all beared off, this player wins the game and the game ends.

2.1.2 Rules of Backgammon

The backgammon rules, collected from the website (bkgm, 2017)[1] are not complex, but there are many particular names. If there is any misunderstand terminology, please refer to Chapter 2.1.1, Names in Backgammon .

1. At the beginning of the game, each player throws a single dice. This determines who plays first. If they have equal numbers, the two players roll again until they roll a different number. The player who throws the higher number then moves the pieces according to the numbers shown on the two dice. After this opening procedure, players should throw two dice in turn until the game ends.

2. The stone can only be moved to an open point that is not occupied by two or more opposing stones.

3. The numbers on the two dice constitute separate moves. For example, if a player rolls 1 and 2, he can move a stone two spaces forward to open stonebars and another stone to an open space for one space, otherwise he might move three spaces (1+2 = 3) with one stone.

4. A player who rolls doubles plays the numbers shown on the dice twice. For example, if a player rolls 3 and 3 he can move his stones 4 times and each time he can move three spaces.

5. If there is no such invalid movement for player, player can give this turn up. However, if there are valid movements in the board, the player must play as many numbers as he can.

6. If there is any stone in bar, it has to be moved first. If this move is invalid, the player should abandon his turn and it is the turn of the other player.

7. Only when all stones are placed on the inner board, can a player get his stones home one by one.

8. If a stone hits another player's stone, the hit stone should be put in the bar, and this stone has to be moved out first during the owner's next turn.

9. Once one player wins the game, by having all 15 stones home, the game ends.

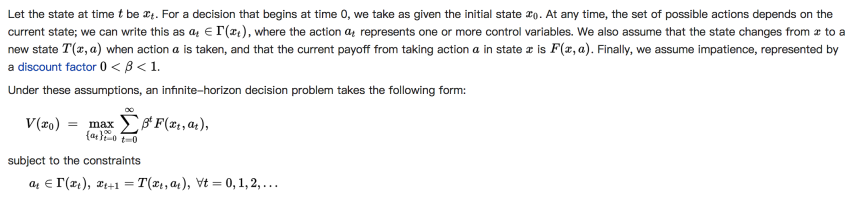
The topic of machine learning in Backgammon had been discussed and solved by the1990s and different people had various solutions. Mentioning ANN in Backgammon, Tesauro' work is one of the most significant, and he has published dozens of papers about Backgammon, especially about his TD-Gammon program. Another work that has influenced my work deeply concerns the co-evolutionary self-play approach of Pollack and Blair(1998) [11]. With reference to the TD-Gammon program, Pollack and Blair held a different opinion that hill-climbing could also train a backgammon master. Tesauro (1998)[19] gave his comments on Pollack and Blair's report and answered the questions they referred to. What follows is the main content of three significant reports.

2.2 Debate between Tesauro and Pollack and Blair

2.2.1 TD-Gammon

In 1992, Gerald Tesauro[17] developed a computer backgammon program, "TD-Gammon", using artificial neural networks. Primarily, he proved the ability of TO nets to discover features on their own in computer games research. Furthermore, according to the scaling of training time with the length of input sequences, and with the size and complexity of the task, he trained simplified endgame situations which could then be used to train in a full-game situation. He also concluded that this approach would work well in practice. The performance of TD-gammon was at a level not far below that of the best human players of the time, and the strategies explored by TD-gammon led to advances in the theory of backgammon play.

Temporal Difference Learning is an important branch of Reinforcement Learning based on the Bellman equation (see figure 5). This learning method estimates the value using the Bellman equation, and then use the estimated value as the target value (see Figure 6).It is a Model-free and on-line method. The training process is updating Step-by-step value and making the estimates consistent.


Let the state at time t be x_t. For a decision that begins at time 0, we take as given the initial state x_0. At any time, the set of possible actions depends on the current state; we can write this as $a_t \in \Gamma(x_t)$, where the action a_t represents one or more control variables. We also assume that the state changes from x to a new state $T(x, a)$ when action a is taken, and that the current payoff from taking action a in state x is $F(x, a)$. Finally, we assume impatience, represented by a discount factor $0 < \beta < 1$.

Under these assumptions, an infinite–horizon decision problem takes the following form:

$$V(x_0) = \max_{\{a_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t F(x_t, a_t),$$

subject to the constraints

$$a_t \in \Gamma(x_t),\ x_{t+1} = T(x_t, a_t),\ \forall t = 0, 1, 2, \ldots$$

Figure 5: This is Bellman equation.[1]

[1]This equation figure comes from wiki and the link is https://en.wikipedia.org/wiki/Bellman_equation.

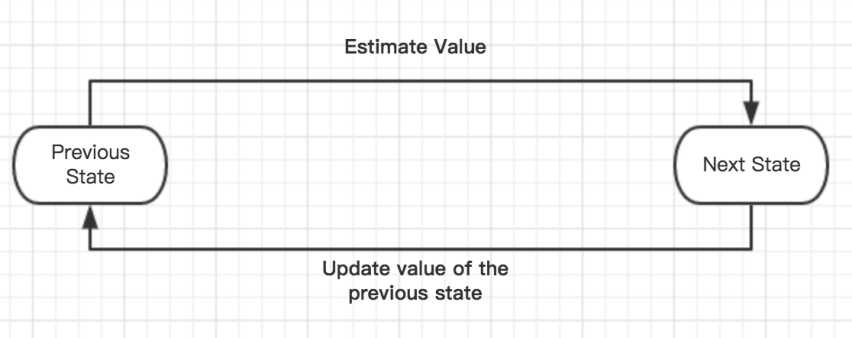


Figure 6: TD Learning Process.

2.2.2 A Co-evolutionary Self-play Approach

In 1997, Pollack and Blair[12] demonstrated that a co-evolutionary self-play approach to developing backgammon strategies could be successful using ANNs and hill-climbing (no reinforcement learning necessary).

Hill-climbing is a co-evolutionary learning method in which the task dynamically changes as the learning progresses through the co-evolutionary setup of the training. They used the same standard feed-forward neural network with two layers, and the sigmoid function, and set up in the same fashion as Tesauro (1992),[17] with 4 units showing the stone number on the board, and then added 2 units each to indicate how many stones are on the bar and are back home. Weights were all from zero Figure 7. They adjusted a little used "champion := 0.95*champion + 0.05*challenger"[2]. Then they succeed in 1998 using this formula. When the challenger wins the contest, rather than just replacing the champion by the challenger, they instead make only a small adjustment in that direction in their experiment. As seen in figure 8., the result demonstrated that the dynamics of backgammon are particularly suited to co-evolutionary learning through self-play (unlike many other games, such as chess). In 1998, they published "Co-evolutionary in the successful learning of backgammon strategy."[11]. They initialized weights and all weights were trained from zero instead of back-propagation, reinforcement or temporal difference. The result was good.

[2]This formula is from the report written by Pollack and Blair[12]

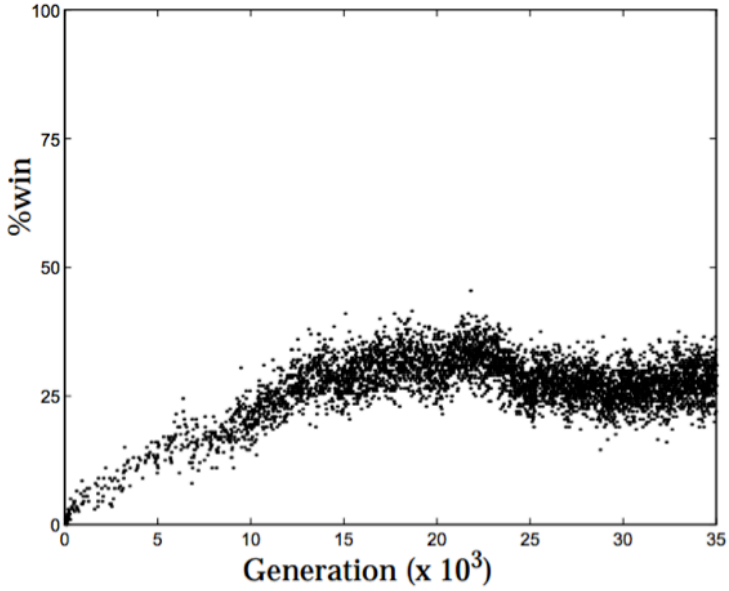


Figure 7: Percentage of wins of our first 35,000 generation players against PUBEVAL. Each match consisted of 200 games. [3]

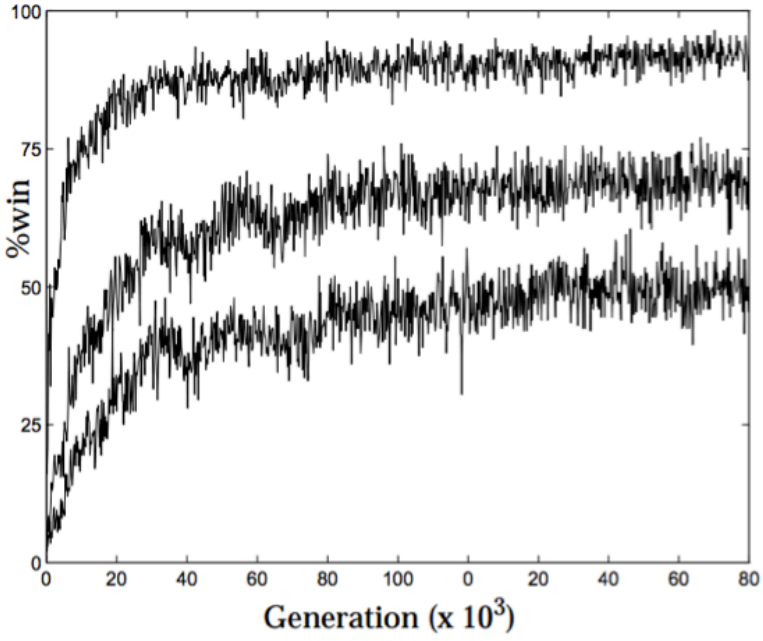


Figure 8: Percentage of wins against benchmark networks 1,000[upper], 10,000 [middle] and 100,000 [lower]. This shows a noisy but nearly monotonic increase in player skill as evolution proceeds. [3]

2.2.3 Tesauro's Comments

After Pollack and Blair's paper was published, Tesauro[19] wrote a short piece to defend his work, arguing that hill-climbing alone is not sufficient to achieve successful learning. In 1998, Tesauro also commented on the conclusion of Pollack and Blair that he analyzed the differences between two totally different methods and proved the correctness of TD learning networks. Firstly, hill-climbing is adjusting the policy network directly without any value networks of prediction. People could

[3]This figure is from the report written by Pollack and Blair in 1998[11]

improve the possibility of winning the game when they have similar situations, but if there is a mutation operation, or without a population of learners, or mutation operates directly on network weights, these possible situations would affect the final result a lot. Secondly, the hill-climbing based on the performance about 40% (A VS Pubeval, and then the percentage of A is about 40%). This cannot be evidence that hill-climbing could beat TD-gammon (or Pubeval). Most importantly, he pointed out the weakness of hill-climbing is that the learning algorithm cannot discover nonlinear solutions, but TD clearly can (see the difference in Figure 9).

Hidden units	Gam	Pub-1	Pub-2
10	0.564	0.527	0.507
20	0.619	0.571	0.557
40	0.655	0.611	0.602

Figure 9: Tesauro's experiment result [4]

2.3 Other Research and Methods

In the following part, I will introduce some other significant points mentioned by researchers and detailed information about backgammon and the research methods and why I chose self-play and ANN as the methods.

Firstly, I will introduce the rich research history in self-play and the co-evolutionary method. Then I would like to introduce ANN and some previous researches on ANN. Finally, I will summarise the research. Firstly, I would introduce the rich history by researchers in Self-Play and how it going of co-evolution.Then I would like to introduce what is ANN and the past researches on ANN. Finally, summarize the research.

2.3.1 Self-Play

Samule (1959)[14] wrote "Some studies in machine learning using the game of checkers" and introduced machine learning and defining it as giving "computers the ability to learn without being explicitly programmed." He ran checkers-learning programs with this method successfully. Self-play, as a new strategy, learned how to play checkers based on results from the game and making movement improvements. In recent decades, Go, chess, checkers and backgammon have all been used as testing grounds for new approaches to machine learning. Then Tesauro (1995)[18] mentioned temporal-difference (TD) learning with self-play as a method successfully used to derive the value approximation function. Wiering(2010)[22], Kotnik and Kalita (2003)[9] also provided a detailed report and summarised that co-evolution produced superior results through such a learning method. Researchers such as Campbell et al. (2002)[5], Gelly et al. (2012)[7], have provided their solutions to board games such as chess and Go using self-play. Thus, self-play is suited to backgammon and also provides superior results through co-evolution.

2.3.2 Co-evolution

Stochastically removing less desired solutions, and introducing small random changes produces each new generation. Then new generation repeats the same process. The cycle is repeated

[4]This table is from Teasuro's Comments in 1998[19]

millions of times to get a final result. This whole process is considered as evolutionary (Poncela, J et al., 2009)[13]. When two players use evolution to generate their new algorithms it is called co-evolutionary. Co-evolutionary algorithms are a class of algorithms used for generating artificial life as well as for optimization, game learning and machine learning. For example, Brown[4] wrote an article about co-evolution in game learning and introduced theoretical models and knowledge that would prove useful in actual application. The co-evolutionary method is widely used in self-learning that help machines to behave with human-like intelligence.

2.3.3 ANN

Figure 10 shows the standard ANN.

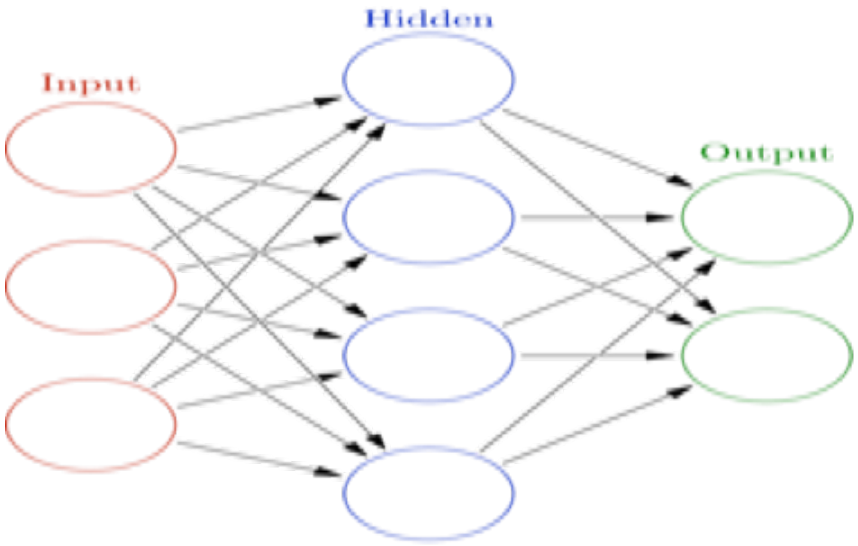


Figure 10: Standard ANN Model[5]

a) Input: The input should be a list of numbers, Booleans or other symbols that could be transformed into numbers. For example, input in my project will be 24 double numbers. The black stone number in each stonebar divided by 15 (the total number of stones) would be positive and the white stones would be negative. Others would be zero.

b) Hidden Layers: Programmers could design the size of hidden layers. It is not necessary for them to be very large and each layer only has several neurons.

c) Output: Input data through the hidden layers would get a result, and the result is the output.

d) Weight: Weight is in range from -1 to 1 and represents the connection of neurons. Besides, ANNs do not need to connect every neuron between different layers. Even neurons in the same layers could be connected. The neurons in the same layer also could be connected but the trend should be in the same direction (See the example in Figure 11).

[5]This ANN Diagram downloads from wiki page: en.wikipedia.org/wiki/Artificial_neural_network

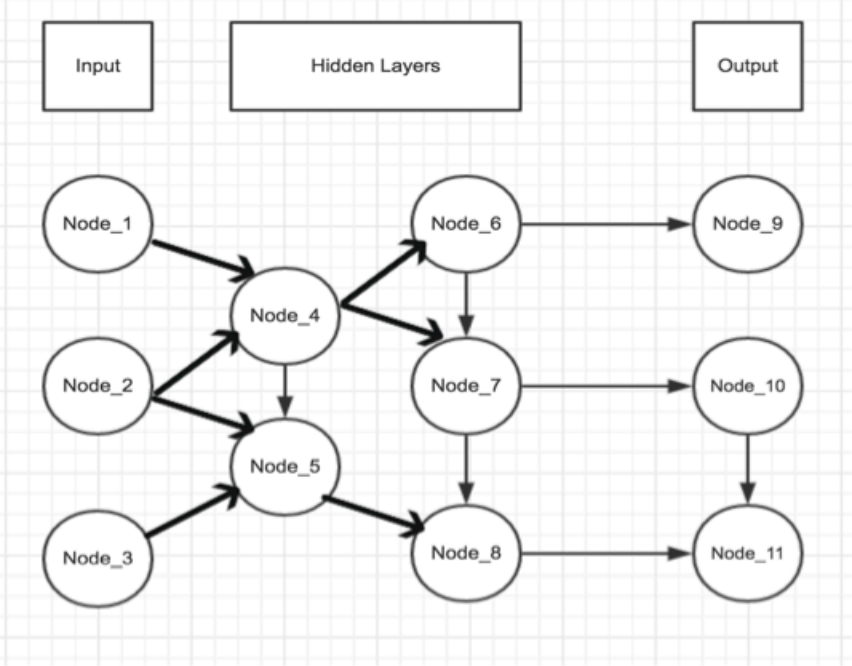


Figure 11: Another kind of ANN (fewer connections)

e) Bias: Usually some quite large or quite small numbers would affect the result and bias numbers could decrease the noise. Most ANNs add sigmoid to balance the noise.

ANNs contain multiple hidden layers with dynamic weights and get an output from its learning. The weights would be modified after each learning test so that the ANNs can be trained better gradually.

2.3.4 ANN Research

Regardless of board games or other puzzle games, programmers could implement self-learning instead of the traditional programming mode that set up engines as AI. In addition, with the continuous progress of its algorithm, it can also help humans to cultivate more talented people. ANN could not only be used in development of games, but also in other social areas. For example, ANN could be used in GPS navigation and always provide users with a better solution on different occasions and improve by self-learning. Furthermore, Gauci and Stanley (2010)[6] stated that the evolving ANNs could be obtained through increasing relevance to neural computation, and researchers in the field of neuroevolution have been developing evolutionary algorithms designed specifically to evolve ANN for at least the past 25 years. Recent developments in ANNs (in particular, deep learning methods) have generated international headlines through successful game playing implementations such as Alpha Go, capable of beating the best humans (Silver at el, 2015)[15].

Despite the TD and hill-climbing methods, there are some other significant knowledge points listed.

- Blair and Pollack[3] highlighted in 1997 that the result of a learning system, to a great extent, relied on the learning environment. Firstly, the probability of winning tends to be near 50% in the early period, and is then affected by some adjustments gradually. But the data, still in range from 15% to 85%, were low, as it was expected that the percentage of winning should be nearly 100% compared to the initial version. Finally, they continued to do research which facilitated co-evolutionary advancement by preventing collusive suboptimal equilibria in learning.

- Because the training is based on the competitor, and we could settle the competitor. Sklar, Blair and Pollack (1998)[16] mentioned in "Co-evolutionary learning: Machines and humans schooling together" that if students learn from their teachers, machines could also learn from some more intelligent human or machines. For example, Machine A knows nothing about chess. Then we train A to copy the same movement of B (Chess On-line Game). After more than 10000 times, the weights of ANNs have been settled and then train A VS A' (A' is slightly different from A). Repeat A VS A' thousands of times for training. This process could not only save a lot time but also benefit the training.

- Waston and Pollack (2001)[21] pointed out that 'intransitive superiority' was the main difficulty in the co-evolutionary approach. For instance, A beats B and B beats C, then we cannot exclude the possibility that C beats A that causes cycle problems, even when each step has slight improvements. In this case, we expect the two players to be evenly matched.

2.3.5 A Short Summary of Research

TD-Gammon, its exclusive training through self-play rather than tutelage, helps computers explore strategies themselves. Unorthodox strategies have had a significant influence on backgammon that provided computer scientists a thought-provoking solution. Pollack and Blair also helped to explain the co-evolutionary self-play approach and explain why TD-Gammon works. Based on this research and new technology, self-play is one of the best methods to implement self-learning.
As for the language, after scanning some papers, most of them use Java to write the User Interface . They often use Matlab or Excel to present the distribution of data. Consequently, I will also use Java to build the whole system and the ANNs. The training process would be recorded in two files including the result of competition and each weight of each neuron of ANNs.

3 Requirements & Specifications

In this section, the requirements of the proposed solution will be listed and discussed, ans will be split into two parts, including functional and non-functional requirements.

These requirements are based upon the summary of the research in Chapter 2.3. This could work in Windows, Mac OSX or a Linux operating system and the implemented IDE is Eclipse. Besides, this work mainly divides into three parts the general system and backgammon players, including human backgammon player, machine player and ANN player.

3.1 Functional Requirements

3.1.1 General System Requirements

1. Operating System: This could work in Windows, Mac OSX or Linux operating system. Furthermore, this computer should allow at least 2G CPU run alone.
2. System Requirements: The project could be implemented in the interface – Eclipse.
3. Internet Connection: The project should be able to work without an Internet connection.

3.1.2 Backgammon Game

1. Only permitted by rules and give error messages or warning messages.
2. Provide the updated game board all the time.
3. Player can choose Competitor, including human player, machine player at three different difficulty levels and ANN player.
4. Cannot take back a move. Because the real rule bans this action.
5. Once a player moves 15 stones back home, this player would be the winner and win the game.
6. dice are totally unpredictable and random.
7. dice number should decide the first and the second. But in this system, white first, helping training of ANN so I simplified that.
8. If dice are same, the player could move four times which increase the unpredictable circumstances.
9. If there is no valid movement, player should give up this turn.
10. Human player needs input specified command, but machine and ANN players would move stones automatically.
11. The testing result should be record and then to be anyalised by users. It helps users understand the difficulty level of each kind of player.

3.2 Non-functional Requirements

As for the non-functional requirements, I pay more attention to four parts, compatibility, use-friendly, feasibility and Playability.

1. **Compatibility** This ANN model needs plenty of calculations so that the computer should be compatible with running the project. Large computation should be managed smoothly in this project.
2. **Use-friendly** The game should be easy to use friendly and everyone could play Backgammon even he is no experience on this game. Human and computer will both work in this game. Consequently, this should have the ability to provide high interaction.

3. **Feasibility** If user wants to train ANN player, only needs to click "run" then all algorithms would run by themselves. Users do not need to understand much knowledge about how it runs or why it works.

4. **Playability** After finishing training of ANN, the best ANN player could be player with human player. It makes this game more interesting and attractive.

4 Backgammon Game and Player Design

In this chapter, the main aim is to introduce the design of game and explain the strategies used in each player and reason why I design five different types as the Backgammon Player, including human player, random machine player, simple machine player and ANN player. The first is to ensure the correctness of whole game. Three machine players are made to compete with ANN player, which is a method of testing.
Firstly, all game should obey following rules firstly (see more details in Chapter 1.2.2 Rules2.1.2 in Backgammon). Then each player also has its own different policy.

4.1 Game Design

According to rules and requirements, I design a backgammon game including three main parts. Firstly, there are one board with 30 stones and two dice. Secondly, the game needs to control stone moved in the board and shows dice numbers. Lastly, it contains at least one function, for example, it needs to judge whether this game is over or not.
Consequently, the design of frame in Figure 12: Beyond the frame design, our training would be based on a large number of repeated game. But as the rules, the first one moves 15 stones back home who wins the game. Under this circumstance, the first player has more chances of winning. To balance this sort of avoidable error, we could swap places of two competitors.

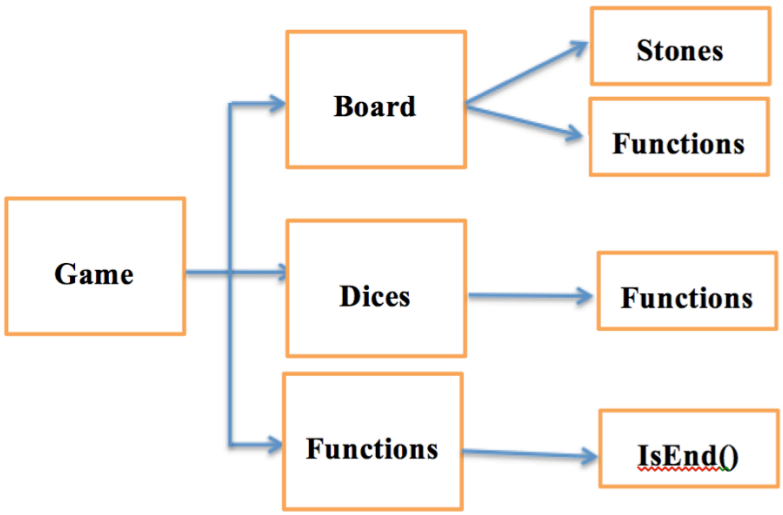


Figure 12: Game Frame Design

4.1.1 Board

Due to Figure 4, the board frame should be contains these following: 1. 15 black stones and 15 white stones. Besides, white stones would be put in stonebar 1, stonebar 12, stonebar 17 and stonebar 19. Black stones would be put in stonebar 24, stonebar 13, stonebar 8 and stonebar 6 (see Table 1).

White Stone Location	1	12	17	19
Numbers	2	5	5	3
White Stone Location	24	13	8	6
Numbers	2	5	5	3

Table 1: Stones location and how many stones would be put

2. Two Bars that prepare for hit stones. 3. Two Homes that prepare for stones back home. 4. Functions, including the initial board, move stone, put stone, bear off, get home and draw all above mentioned. See Figure 13.

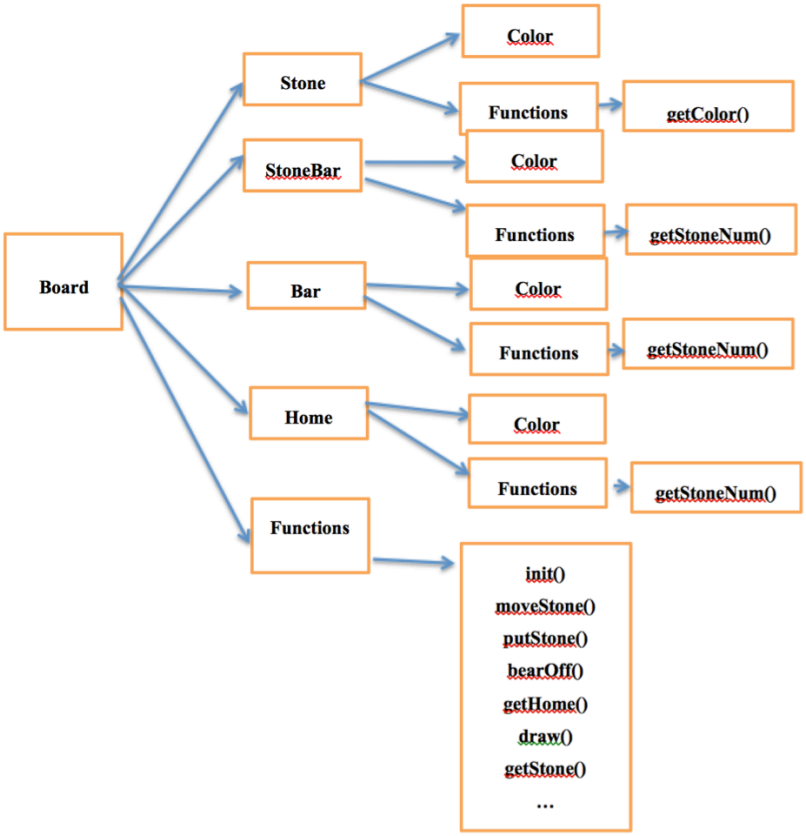


Figure 13: Board Frame Design, including components and functions

```
--1---2---3---4---5---6---BW----7---8---9--10--11--12--BLACKHOME
| 0 |   |   |   |   | # ||   ||   | # |   |   |   | 0 |
| 0 |   |   |   |   | # ||   ||   | # |   |   |   | 0 |
|   |   |   |   |   | # ||   ||   | # |   |   |   | 0 |
|   |   |   |   |   |   ||   ||   | # |   |   |   | 0 |
|   |   |   |   |   |   ||   ||   | # |   |   |   | 0 |
|   |   |   |   |   |   ||   ||   |   |   |   |   |   |
|   |   |   |   |   |   ||   ||   |   |   |   |   |   |
=================================================================
|   |   |   |   |   |   ||   ||   |   |   |   |   |   |
|   |   |   |   |   |   ||   ||   |   |   |   |   |   |
|   |   |   |   |   |   ||   ||   | 0 |   |   |   | # |
|   |   |   |   |   |   ||   ||   | 0 |   |   |   | # |
|   |   |   |   |   | 0 ||   ||   | 0 |   |   |   | # |
| # |   |   |   |   | 0 ||   ||   | 0 |   |   |   | # |
| # |   |   |   |   | 0 ||   ||   | 0 |   |   |   | # |
-24--23--22--21--20--19---BB---18--17--16--15--14--13--WHITEHOME
Please choose human or machine(enter human / machine):
```

Figure 14: Board UI Design

Then I design a the board in figure 14, "#" stands for black stone and "0" stands for white stones. Bars are located in the centre of the board and homes are on right side. "BW" stands for the place for white stones and "BB" presents the place for black stones. If the amount stones in one stonebar is more than 6, the stones would be shown using symbols and one digit.

4.1.2 Dice

Two dice are both random number from 0 to 6, so I design a "generator" to create random numbers. It also should record whether dice have been used or not. Each time dice have been used, dice

should have a boolean return whether this turn is over. As a result, the frame should be like in Figure 15.

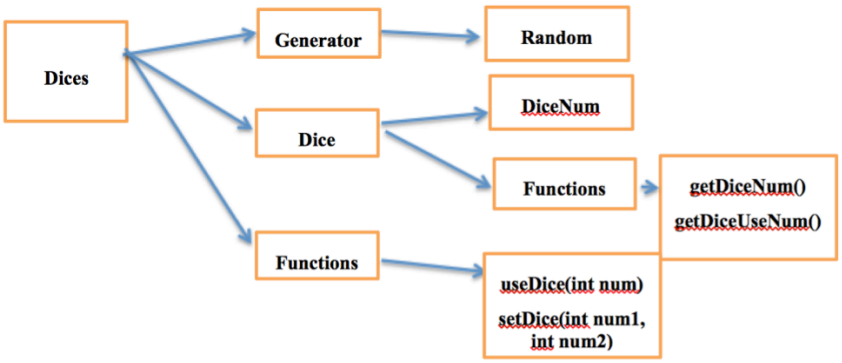


Figure 15: Dice Frame Design, including components and functions

4.2 Human Play Version

Hereafter "Human play version" will be abbreviated as "human". Human play this game should move the stone though the inputs and judge the whole board.
And this requires the human play version following:
a. Read the inputs and move the stone.
b. If the input is invalid or movement is invalid, give a response and ask human enter the input again.
c. Show the board after every movement.
d. Human player needs to judge the "Shut out" (there is no valid movement in this turn so player has to pass this turn).

4.3 Random Machine Play Version

Hereafter "Random Machine Play Version" will be abbreviated as "random". "random" is similar to "human", only the system would play it automatically.
And this requires "random" follow to these:
a. The Game provides all possible movements first, and then "random" pick one up for next step. If there is no valid movement, "random" gives the turn up automatically.
b. The movement is totally random.
Figure 10- Consequence of Random Machine Play Version

4.4 Simple Machine Play Version

Hereafter "Simple Machine Play Version" will be abbreviated as "simple". "simple" is also an automatic player but it would select the best movement from all possibilities.
The agent would follow these strategies:
a. If there is any stone capable of hitting another player, do this movement first.
b. If any movement could decrease the number of "Blot" in the board, do this movement.
c. If there is none of the above conditions are met, move the valid stone which is the nearest to home.

4.5 ANN Play Version

Hereafter "ANN Play Version" will be abbreviated as "ANNP". "ANNP" is totally different from above four players. Firstly, "ANNP" has no extra rules except for the basic rules in backgammon. Secondly, "ANNP" would learn how to play by itself without settings. Lastly, "ANNP" needs to play against with "human", "random" and "simple" to prove its progress. So the ANNP Frame Design is absolute different (see Figure 16):

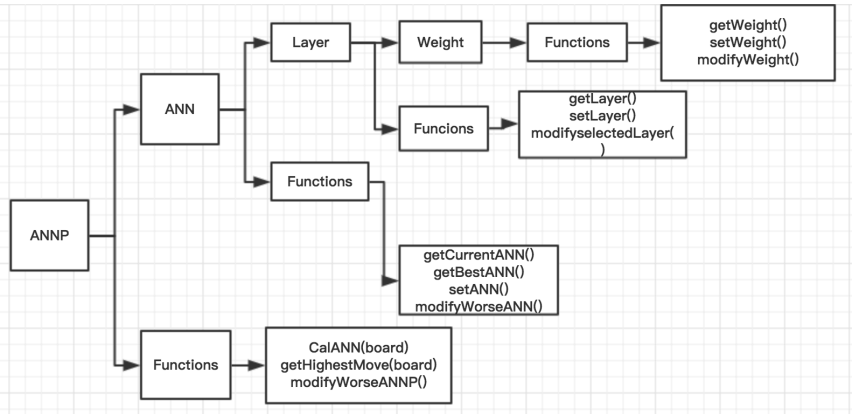


Figure 16: ANNP Frame Design

The most significant point in "ANNP" is the choice of next movement. "human" get command determined next step, and "random" and "simple" both machine players could make decisions by themselves because the policies. However, "ANNP" should decide its movement after comparing all possible situations. The board situation after one possible movement would be used as input, then users can set the number of layers and the amount of each layer in advance. All weights would be altered once A wins more than half times or loses more than half times during the competition between A and A' (A' is a copy of A but modifies one weight).

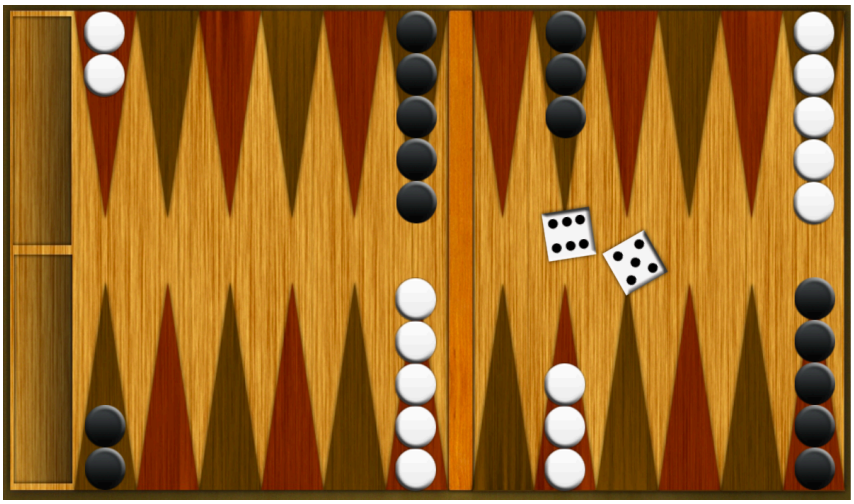


Figure 17: An example of Backgammon on-line game[6]

For example in Figure 17, white "ANNP" has "5" and "6" two dice, so here are seven possible movements for white player:

1. Move 12 to 17 and move 12 to 18
2. Move 12 to 17 and move 17 to 23
3. Move 1 to 7 and move 7 to 12
4. Move 1 to 7 and move 12 to 17
5. Move 1 to 7 and move 17 to 22
6. Move 12 to 18 and move 18 to 23
7. Move 12 to 18 and move 17 to 22

[6]This figure is a screen-shot of an on-line game, the link is http://www.4399.com/flash/71113.htm

"human" picks one valid to move by human-thinking, "random" picks one of them to move by random generator and "simple" picks the second movement according to its strategies. "ANNP" calculates the final score of the board after each possible movement and then picks the highest one. Other curial points are the pre-setting and the modification of ANN.

- **Pre-setting of ANN**
  The pre-setting includes the amount of neurons, the size of each layer and the number of hidden layers. Besides, it also contains the initial weights of each neuron and connections between each neuron or several neurons. I would introduce these details following and list the reason why I design such pre-setting.

Hidden units	Gam	Pub-1	Pub-2
10	0.564	0.527	0.507
20	0.619	0.571	0.557
40	0.655	0.611	0.602

Figure 18: The Amount of Hidden units in Pollack and Blair' experiment

1. **The Amount of Neurons**
   Tesuaro(1992)[17] did not mention this in his report and Pollack and Blair(1997)[12] only use "hidden units" to stand for the neurons. In addition, they made a table in Figure 18 that shows they use 10, 20 and 40 units as their ANN neurons. Consequently, the amount of neurons should be in a range of 10 to 50.
2. **The Size of Each Layer and The Number of Hidden Layers**
   It is related to the whole structure of ANN and the amount of neurons. If the amount of neurons is more than 20 or 30, dividing them into several layers may be better than single layer. So this should evaluate after experiments then conclude which ANN structure is better for this training.
3. **The Initial Weight of Each Neuron**
   In Pollack and Blair(1997)[12] report, they set the initial weights all from zero. Due to the inspiration of "random", initial weights could be in range of (-1,1) randomly. As a result, this part also needs to be considered in two ways. One is the initial weights should be 0. The another is the initial weights should be random numbers.
4. **Connections**
   According to "Connectionist Model" is the another name of ANN, the connections between neurons are essential for ANN. For example, BP (Back Propagation) neural network is a neural network learning algorithm. The hierarchical neural network is composed of input layer, middle layer and output layer. Besides, there is no connection between neurons in each adjacent layer. And all neurons in the adjacent layers are connected. Compared to BP, ANN also could add connections between in each adjacent layer and reduce the connections between neurons. In order to control variables, connections would be design like that there is no connections every neurons in the same layers but all neurons in next layers would be connect to former layers.

- **Modification of ANN**
  This design also depends on the structure of ANN and is related to the initial weights and threshold. Because this ANN does not need adjust the weight through the difference between

output and target, it does not need to consider the threshold. So there are three aspect should be considered:

1. **What is the situation that weight should be modified**
   Self-learning represents competitors learn from each other once they have better solution even they make little progress. For instance, if the competitor A ("ANNP") plays against the competitor A' ("ANNP") for 100 times, the result shows A has won 51 times and A' only has 49 times. Due to dice rules, it is hard to judge whether A is better than A'. But in the self-learning process, A' would be modified once A wins more games. Also, if A' wins more games, A would be modified.
2. **Which weight should be modified and how to do it**
   There are two ways to select weights. One way is that all weights should be modified. Another solution is choose one weight randomly during each training. Pollack and Blair (1997)[12] changed all weights once this ANN loses the game. However, in my design, I would only change one weight at one time. The weight would be selected randomly.
3. **The relationship between new weight and old weight**
   Pollack and Blair (1997)[12] use "champion := 0.95*champion + 0.05*challenger" to adjust their ANNs. From this formula, it can be concluded that the new weight is slight different to the old one. Whereas, I decide to keep the champion and make a new challenger which is slight different from champion. I create new random number in the range from -0.05 to -0.02 and from 0.02 to 0.05 that the weights in a small fluctuation range.

5 Implementation

This section utilizes the basis of game rules and the design section to implement the training of "ANNP". In the first priority, the game runs smoothly and obeys policy rules. Secondly, apply different strategies for each player version. Last, make the fierce competition between each players. Next, I would explain the implementation in details.

5.1 Game

Game contains two main objects, board and dice. Game has two player, black and white. White plays first and each turn this game system would check whether the dice are on rolling (the player does not finish his turn)or not. Run this player until he finishes his turn or he has no valid movement.

1. **Board**
   A board should have six parameters,including the stones number at black home, the stones number at white home,the stone number at black bar, the number of stones at white bar, an array of stones counts at every stonebar and an array of stonebar colors.
   Then, we should consider the functions about board:

   (a) Create new board. Then the stand map has 15 stones may be captured by the other competitor. However, in order to decease the difficulty of backgammon, I also make some new maps for beginners, users can modify following such code in Game.java. Besides, it can simplified the game that "ANNP" may learns quicker than that from a standard map.
   BoardMap.getMap15(stoneColors, stoneCounts);
   // Get Stand Map
   => BoardMap.getMap3(stoneColors, stoneCounts);
   // Get a Simplified Map

   (b) Check whether the selected stone be moved or not, and return a boolean. Stone cannot be moved until following three conditions apply simultaneously.
   1. The selected place is in the board that means the place cannot beyond the range of [0,23].
   2. There is no stone in the bar. But if it has stones in the bar, player must move the stone in bar firstly. Use a new function to implement.
   3. There is no more than one another color stone in the target place.
   Input from and target => Check the movement => Return boolean

   (c) Only player get the admission to move the selected stone, the board can move this stone from the selected place to target place.

   (d) Copy the old board before one valid movement and then recover board back when "ANNP" tries all possible movements. But the shallow cloning – "clone()" in Java package is not working when the board needs deep cloning. In this case, I apply Board implements Cloneable and then record the old board situation.

   (e) Add stone, delete stone, get current board...

2. **Stone**
   It is much clearly to see that a single stone has its own color."0" stands for White Player and "#" represents Black in the board. Besides, 30 stones comprise a game.

3. **Dice**
   In the real game, at the beginning, two players would roll the dice to decide the player who plays first. However, I choose the first player always is Black Player in order to simplified some unnecessary situations.
   During one turn, each player roll two dice and the number of dice are totally random number from 1 to 6. In addition, if two numbers of dice are same, player can move in double. Before one possible movement, the remaining times of dice would be checked and then it determines whether this turn is over or not. Most functions have been designed in the design part 4.1.2.

5.2 “human”

“human” only needs to type two integers (if there is no stone in the bar) or one integer (if there is no less than one stone in the bar) to complete the command.
System.out.println(“Please enter piece location using blankspace (like a b, a is the numer, b is move number): ”);
sc = new Scanner(System.in);
String input = sc.nextLine();
Then this command would be executed by “human” and game would give response on this command, including “Error Movement!” such error message. If “human” has no valid movement to choose, enter “no” to pass this turn.

5.3 “random”

This is written based on “human” but it runs automatically. According the current board, use an array to record all possible stonebars and then choose one randomly. In addition, if there is more than one stone on the bar, move this sort of stones firstly and then consider other possible movements. The hard point to implement is to decide which dice should be used in this turn. Because there are only two dice, here are three situations:

1. Only remains Die One. Die Two has been used.
   (a) Check there is any valid movement using Die One and return a boolean.
   (b) If the boolean is false, then reset the Die One Remaining Times zero and turn this round.
   (c) If the boolean is true, move one movement randomly and reduce the Die One Remaining Times.
2. Only remains Die Two. Die One has been used.
   Same as the former one situation but Die Two replaces Die One.
3. Die One and Die Two both can be used.
   However, due to this Player choose stone in random, I do not need to consider which one is used first. So “random“ only needs to consider same as the former two situations.

5.4 “simple”

“simple” has more intelligent strategies compared to “random” because it needs to consider which movement is better than others instead of move randomly. Besides, it should follow the basic rule that if there is more than one stone on the bar, move this kind of stones firstly. Next, pay more attention to the possible movements on the board. First, the strategies have been set and then “simple” should consider the dice usage. Similar to “random”, this also can be considered as three situations:

1. Only remains Die One. Die Two has been used.
   (a) Check there is any valid movement using Die One and return a boolean.
   (b) If the boolean is false, then reset the Die One Remaining Times zero and turn this round.
   (c) If the boolean is true, move the best choice due to the policies discussed in Chapter 4.4. If any stone can go back home, move this stone first. Then if this movement can hit some stone, choose this movement. Besides, if this movement can decrease the amount of “Blot”, do this movement. In addition, if it is possible, “simple” should avoid increasing the amount of “Blot”. Beyond all these cases, move the valid one which is nearest to home.
2. Only remains Die Two. Die One has been used.
   Same as the former one situation but Die Two replaces Die One.

3. Die One and Die Two both can be used.
   This situation almost the same as the previous two situations. Whereas, this situation should compare all possible movements. Then select the best one.
   So, this situation also could be divided into two possible cases. If Die One is equal to Die Two, perform the same as Chapter 5.4.1. or Chapter 5.4.2.. According to Chapter 5.4.1.(c), return different numbers standing for different degrees of importance. Due to the return numbers, "simple" can easily pick the best movement out.

5.5 "ANNP"

"ANNP" is more complex than "human", "random" and "simple" these players who could have command or polices to help making decisions. "ANNP" only can reply on the final result representing the better player. As a result, it is important to build up a appropriate ANN model. "ANNP" has two parts, ANN and functions. ANN part only contains the adjustment and initial functions. Then functions in "ANNP" call ANN which are similar to "simple".

In the light of the Design of ANN in Chapter 2.3.3, an ANN model should be set up like this:
Input => Hidden Layers => Output
Firstly, I would set the current board as the input. As for the array, it has 24 integers. 0 stands for the situation that if there is nothing in this stonebar. n stands for the number of stones in this stonebar and m stands for the number of all stones left on the board including in the bar.**n / m** is positive for the player who is playing in this turn and negative for other color player. Give one example to help explain (see Figure 19):

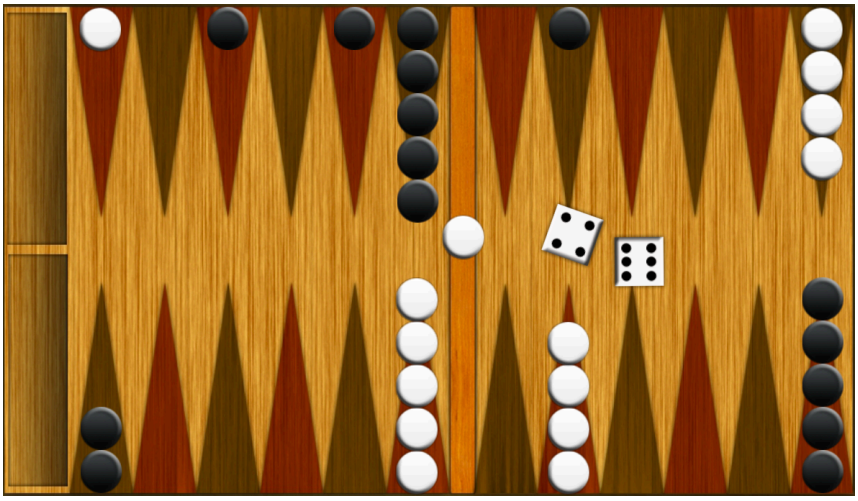


Figure 19: An example of Backgammon On-line Game. If there has stones in bar, player must move this stone from the bar.[6]

- White Player:

  If it turns to White, the current board should have only one possible movement because there is one stone in the bar. It must be moved firstly according to the basic rules. So the White player must move like this (see Figure 20):

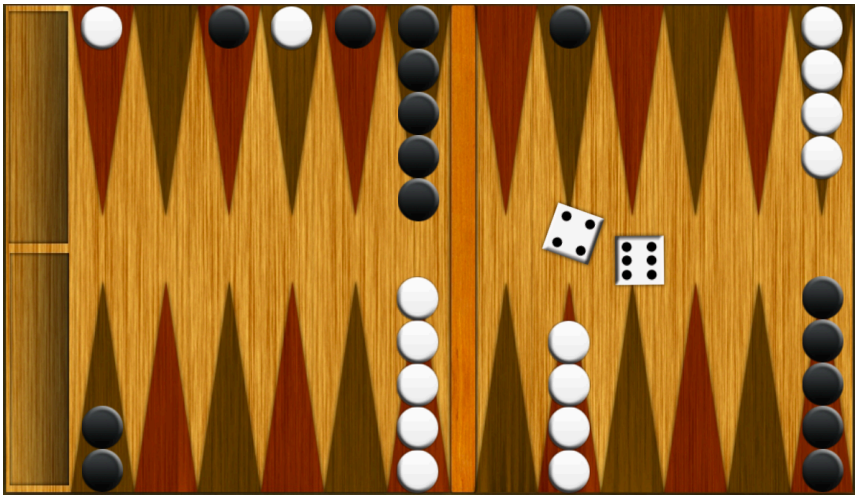


Figure 20: An example of Backgammon Game For White Player. If there is no stone in bar, player has 8 possible movements for next step .[6]

Then this board would be seen as the current board and be transformed as the input (see Table 2):

stonebar NUMBER	1	2	3	4	5	6	7	8	9	10	11	12
Input NUMBER	1/15	0	-1/15	1/15	-1/15	-5/15	0	-1/15	0	0	0	4/15
stonebar NUMBER	24	23	22	21	20	19	18	17	16	15	14	13
Input NUMBER	-2/15	0	0	0	0	5/15	0	4/15	0	0	0	-5/15

Table 2: Input Table for White Player. Red stands for white stones and blue stands for black stones. Green stands for the moved stone.

- Black Player:

  If it turns to Black, the current board should have only nine possible movements. So the Black player can move four steps or six steps as the first movement. Then this board would be seen as the current board and be transformed as the input (see Table 3 to Table 11):

stonebar NUMBER	1	2	3	4	5	6	7	8	9	10	11	12
Input NUMBER	-1/15	0	1/15	0	1/15	5/15	0	1/15	0	0	0	-4/15
stonebar NUMBER	24	23	22	21	20	19	18	17	16	15	14	13
Input NUMBER	1/15	0	0	0	1/15	-5/15	0	-4/15	0	0	0	5/15

Table 3: The First Possible Movement as the Input Table for Black Player. Red stands for white stones and blue stands for black stones. Green stands for the moved stone. Move 4 steps.

stonebar NUMBER	1	2	3	4	5	6	7	8	9	10	11	12
Input NUMBER	1/15	0	1/15	0	0	5/15	0	1/15	0	0	0	-4/15
stonebar NUMBER	24	23	22	21	20	19	18	17	16	15	14	13
Input NUMBER	2/15	0	0	0	0	-5/15	0	-4/15	0	0	0	5/15

Table 4: The Second Possible Movement as the Input Table for Black Player. Red stands for white stones and blue stands for black stones. Green stands for the moved stone. Move 4 steps.

stonebar NUMBER	1	2	3	4	5	6	7	8	9	10	11	12
Input NUMBER	-1/15	1/15	1/15	0	0	5/15	0	1/15	0	0	0	-4/15
stonebar NUMBER	24	23	22	21	20	19	18	17	16	15	14	13
Input NUMBER	2/15	0	0	0	0	-5/15	0	-4/15	0	0	0	4/15

Table 5: The Third Possible Movement as the Input Table for Black Player. Red stands for white stones and blue stands for black stones. Green stands for the moved stone. Move 4 steps.

stonebar NUMBER	1	2	3	4	5	6	7	8	9	10	11	12
Input NUMBER	-1/15	0	1/15	0	1/15	5/15	0	1/15	1/15	0	0	-4/15
stonebar NUMBER	24	23	22	21	20	19	18	17	16	15	14	13
Input NUMBER	2/15	0	0	0	0	-5/15	0	-4/15	0	0	0	4/15

Table 6: The Fourth Possible Movement as the Input Table for Black Player. Red stands for white stones and blue stands for black stones. Green stands for the moved stone. Move 4 steps.

stonebar NUMBER	1	2	3	4	5	6	7	8	9	10	11	12
Input NUMBER	-1/15	0	1/15	1/15	1/15	5/15	0	0	0	0	0	-4/15
stonebar NUMBER	24	23	22	21	20	19	18	17	16	15	14	13
Input NUMBER	2/15	0	0	0	0	-5/15	0	-4/15	0	0	0	5/15

Table 7: The Fifth possible Movement as the Input Table for Black Player. Red stands for white stones and blue stands for black stones. Green stands for the moved stone. Move 4 steps.

stonebar NUMBER	1	2	3	4	5	6	7	8	9	10	11	12
Input NUMBER	-1/15	1/15	1/15	0	1/15	4/15	0	1/15	0	0	0	-4/15
stonebar NUMBER	24	23	22	21	20	19	18	17	16	15	14	13
Input NUMBER	2/15	0	0	0	0	-5/15	0	-4/15	0	0	0	5/15

Table 8: The Sixth possible Movement as the Input Table for Black Player. Red stands for white stones and blue stands for black stones. Green stands for the moved stone. Move 4 steps.

stonebar NUMBER	1	2	3	4	5	6	7	8	9	10	11	12
Input NUMBER	-1/15	0	1/15	0	1/15	5/15	1/15	1/15	0	0	0	-4/15
stonebar NUMBER	24	23	22	21	20	19	18	17	16	15	14	13
Input NUMBER	2/15	0	0	0	0	-5/15	0	-4/15	0	0	0	4/15

Table 9: The Seventh possible Movement as the Input Table for Black Player. Red stands for white stones and blue stands for black stones. Green stands for the moved stone. Move 6 steps.

stonebar NUMBER	1	2	3	4	5	6	7	8	9	10	11	12
Input NUMBER	-1/15	2/15	1/15	0	1/15	5/15	0	0	0	0	0	-4/15
stonebar NUMBER	24	23	22	21	20	19	18	17	16	15	14	13
Input NUMBER	2/15	0	0	0	0	-5/15	0	-4/15	0	0	0	5/15

Table 10: The Eighth possible Movement as the Input Table for Black Player. Red stands for white stones and blue stands for black stones. Green stands for the moved stone. Move 6 steps.

stonebar NUMBER	1	2	3	4	5	6	7	8	9	10	11	12
Input NUMBER	-1/15	0	1/15	0	1/15	5/15	0	1/15	0	0	0	-4/15
stonebar NUMBER	24	23	22	21	20	19	18	17	16	15	14	13
Input NUMBER	1/15	0	0	0	0	-5/15	1/15	-4/15	0	0	0	5/15

Table 11: The Ninth possible Movement as the Input Table for Black Player. Red stands for white stones and blue stands for black stones. Green stands for the moved stone. Move 6 steps.

Then another essential point is the initial ANN. As the pre-setting design in Chapter 4.5, I would create a ANN that each neuron would connect to other neurons in the former layer and then add one neuron "1" to balance the error. The add up all results in the final layer as the output. Compare 8 outputs from 8 possible movements and select the highest one to move.

In order to implement all possible movements, the board would move the stone and then do the training through ANN and move the stone back that board could try other possible movements and avoid to influence the following game. Due to this reason, board needs deep cloning instead of shallow cloning. It not only helps moving stone back, also reducing some unnecessary coding.

5.6 Competition (A VS B)

The competition may happens between any players. So I provide users to edit the competitors choosing from "human", "random", "simple" and "ANN" these words to presenting the competitors. At the same time, the result of competition would be record in "txt" files.

1. Human VS Human => Prove the Correctness of Game
   Firstly, in order to prove the correctness of "human", I test "human" VS "human". Proving the correctness of "human" is to prove the completeness and exactness of the whole game program.
2. "random" VS "random" & "human" VS "random"=> Prove the Correctness of "random"
   Then, I need to prove the correctness of "random" and "simple". So I wrote one file that "random" VS "random". Each game would repeat 100 times in order to lift the reliability and accuracy of this testing. The result would be discussed in detail in Chapter 6 Evaluation.
3. "human" VS "simple" & "random" VS "simple" => Prove the Correctness of "simple"
   Later, run "human" VS "simple" and "random" VS "simple". "simple" should be better than "random" and the result could be recorded in Chapter 6 Evaluation.
4. "ANNP" VS "ANNP" => Train the ANN of "ANNP"
   The ANN weights all initialized to zero. Then do little change to one weight and apply this in new "ANNP". Repeat 100 times of this competition between the initial "ANNP" and the new "ANNP". According to the win-rate, keep the better one as the better "ANNP". Then copy the better "ANNP" and alter one weight randomly as the new "ANNP". Repeat this competition as many as I can and get the best "ANNP".
5. "ANNP" VS "random" & "ANNP" VS "simple"=> Prove the progress of "ANNP" and prove the correctness of co-evolution works in Backgammon
   Similar to the competition "random" VS "simple", I would not alter any weight of ANN and calculate the win-rate.

The last most important point is the predominance of first player. In the real game, the first player would be decided by the dice number. But the first player in this game is Black so that it is more convenient to statistical data. In order to avoid the predominance who plays first, the number of competition must be even and exchange their seats.

```
    if(index%2 == 1)
testAGame(index,"A" ,"B");
else
testBGame(index,"B" ,"A");
```

6 Evaluation

In order to evaluate the each part of this project, I divide this into three parts. The first is the results of testing of various players, the second part is the reflection of this project including the management and review and the last part is the future work what application this project could be applied in and the future development.

6.1 Testing

Because the results of testing are not only providing a method to prove the correctness of the whole game, but also helping me debugging the possible mistakes. As the design and implementations, this section is also divided into four parts because of four different players.

6.1.1 "human"

The first player is Human Player which is totally same as the real game in the word. Player should manage the stones by themselves and the program only provides error messages or warnings. This program would not provide any suggestion or any extra useful information. Whereas, the success of this player testing suggests that the whole game is working well without any regular mistakes. Here are some figures to help explain the basic rules and functions used in backgammon. The complete process can be played in "test2.java".

1. Move a stone to a blank stonebar, see Figure 21

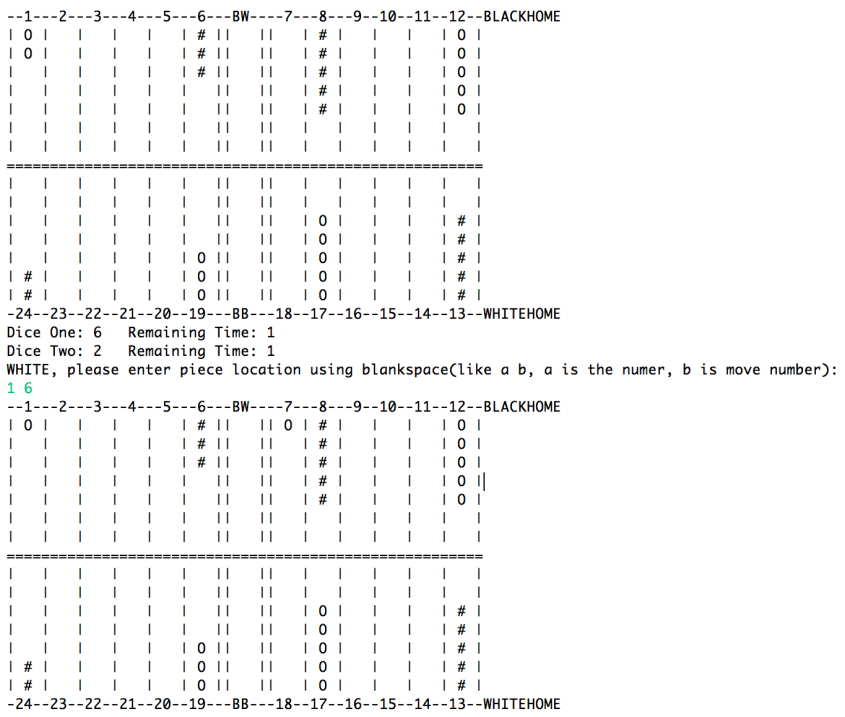


Figure 21: Human Player moves a stone to a blank stonebar

2. Move a stone to hit a stone, see Figure 22

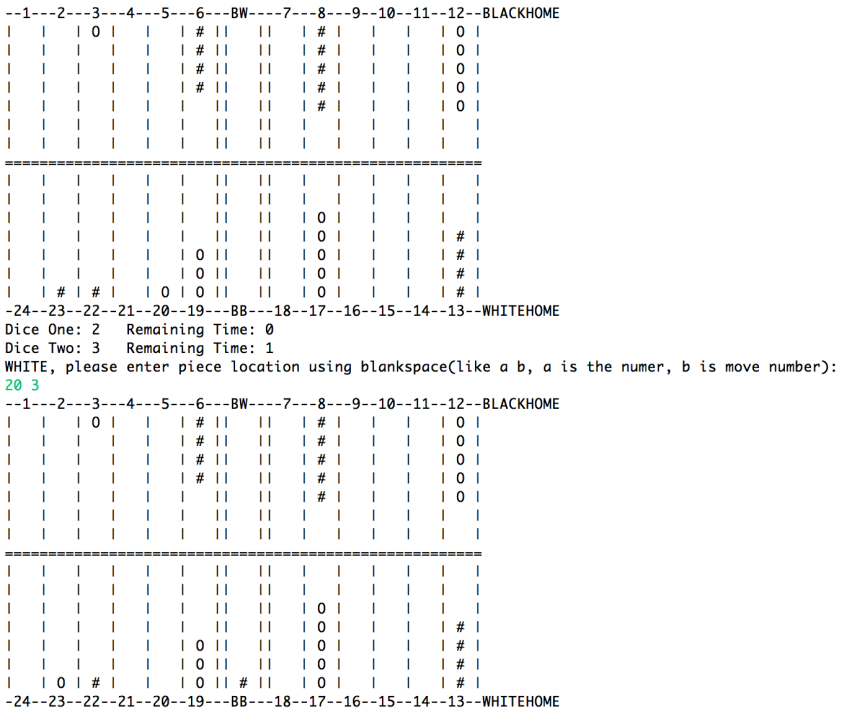


Figure 22: Human Player moves a stone to hit a stone

3. Move a stone to a blank stonebar see Figure 23

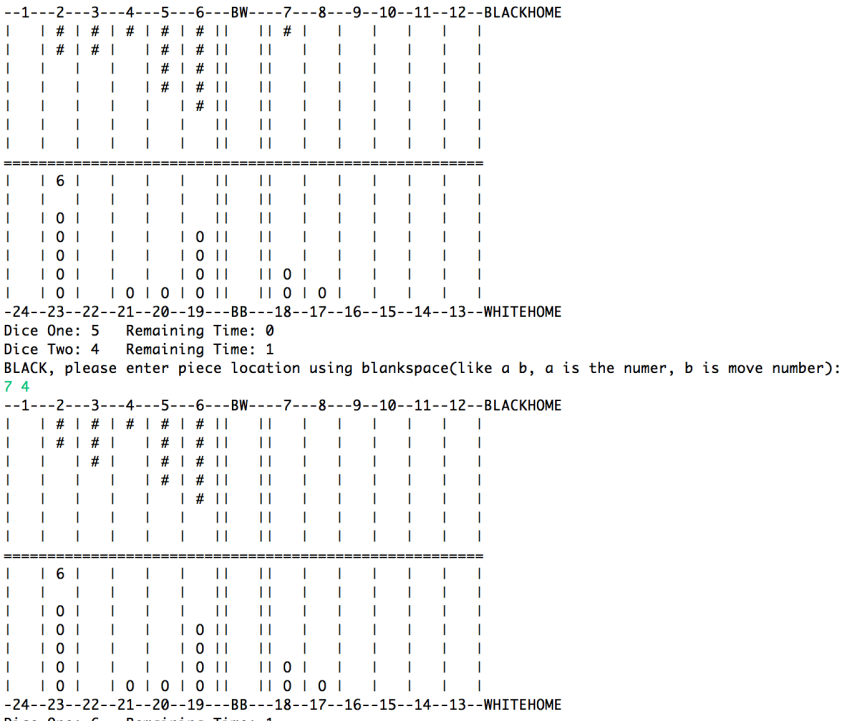


Figure 23: Human Player moves a stone to hit a stone

4. Move all stones to inner board then the player status is bear off (means player can move stones to return home) see Figure 24

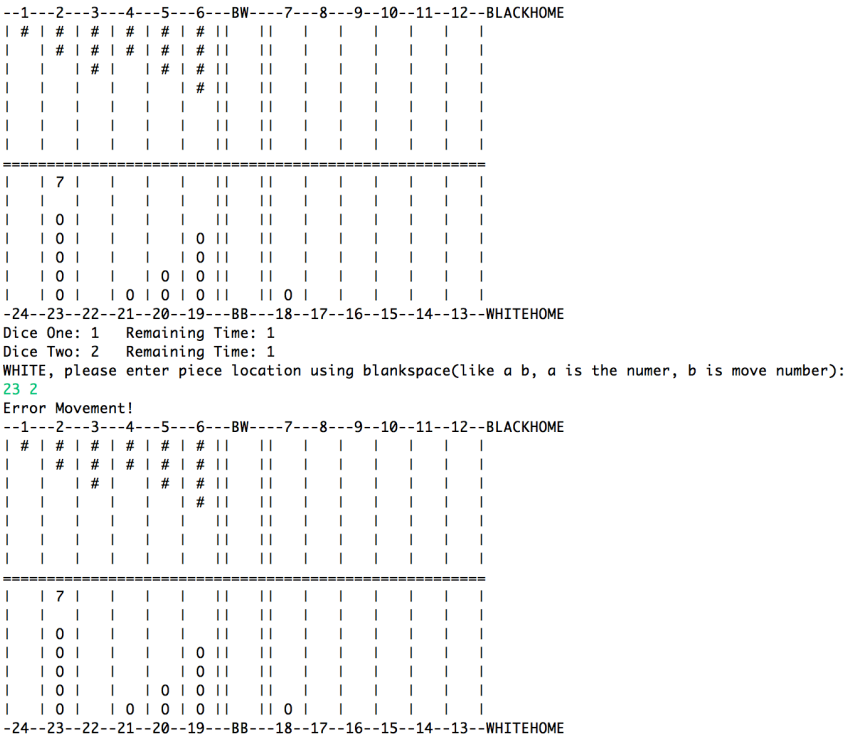


Figure 24: Human Player moves all stones to inner board

5. Player cannot move stone back home until bear off and get an error message. 25

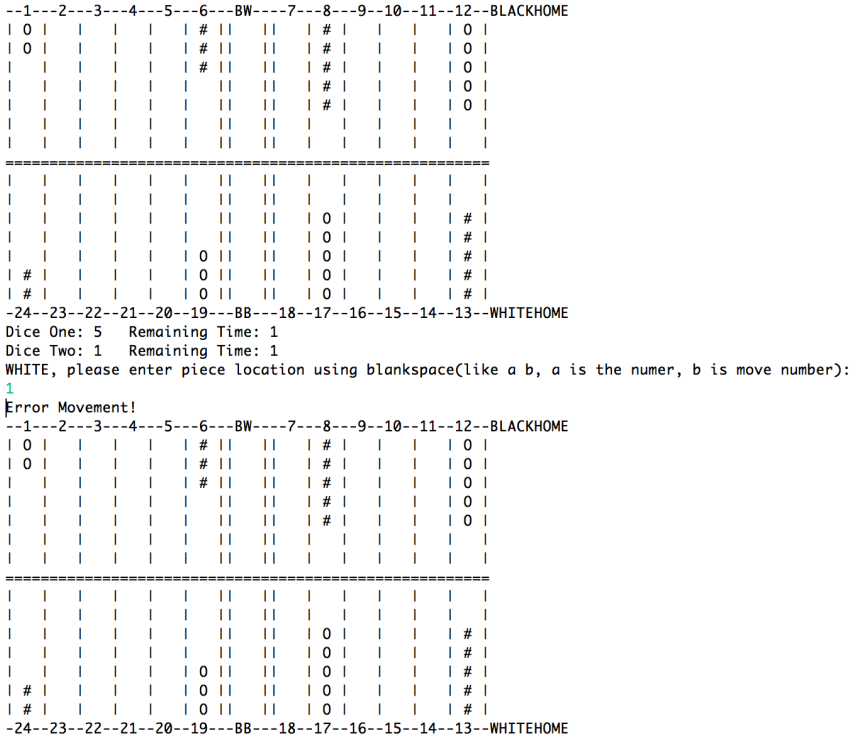


Figure 25: Human Player cannot move stone back home until bear off

6. If player enters error input, it returns an error message 26

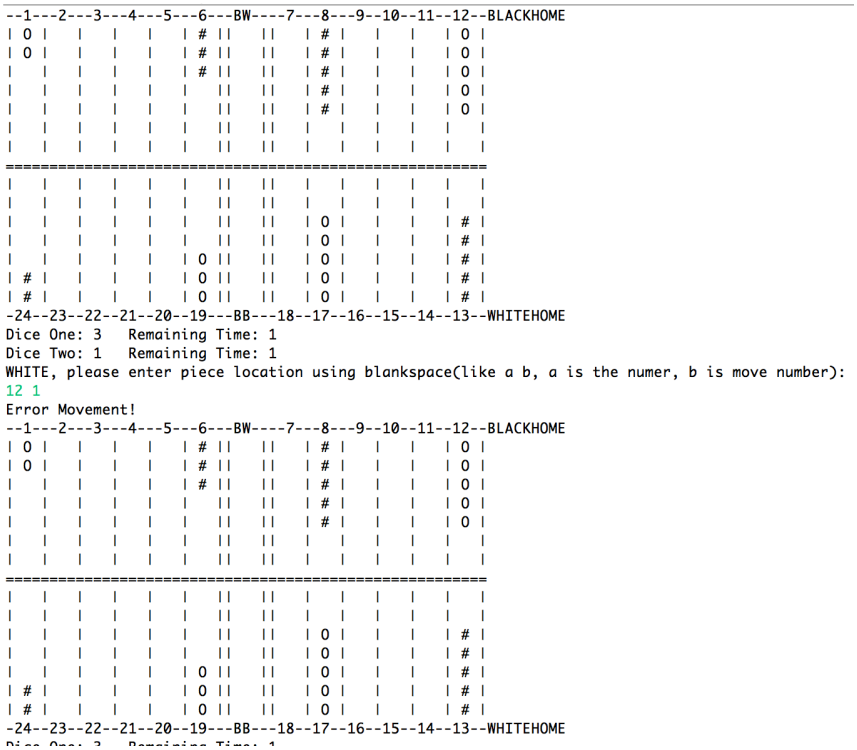


Figure 26: Human Player gets an error message when enters wrong input

7. If player has no valid movement to implement, please enter "no". However, if there exists at least one valid movement, player must move such stone. 27

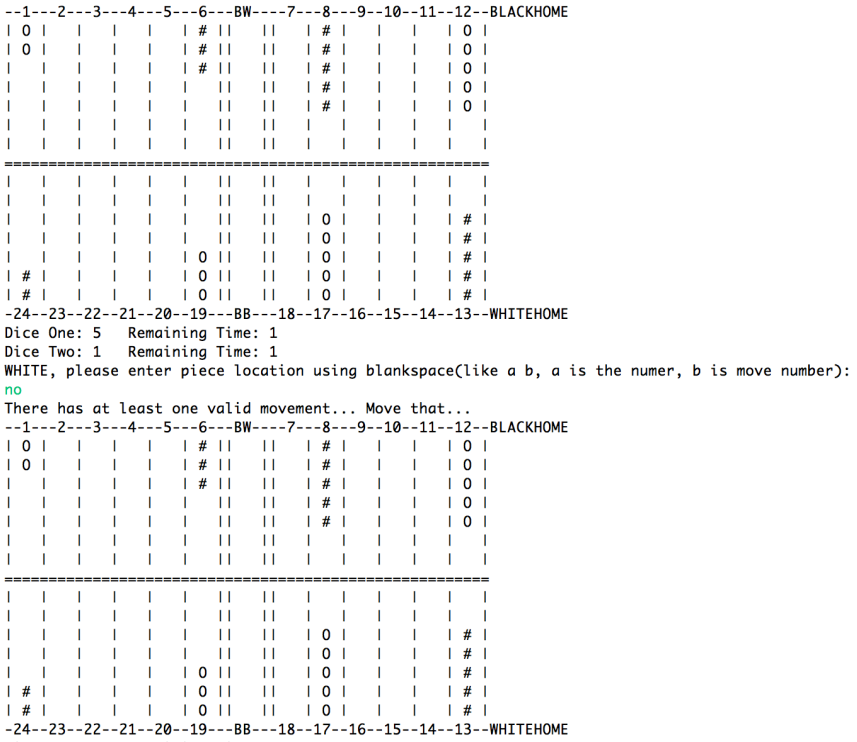


Figure 27: Human Player cannot give up this turn when player still has valid movements.Then player would get an error message.

Above all examples stand for the correctness of backgammon based on basic rules. "random",

"simple" and "ANNP" all would optimize the move methods based on this game. Due to this reason, this success of test is the fundamental of development of other players.

6.1.2 "random"

Then Random Machine Player should play against with "human" and itself.
The former one test is for checking the random selection. Most time, "human" plays better than "random" because human have a more holistic view of game but also more intelligence in small scale.
Another test is "random" VS "random". In this test, as I design eariler in Chapter 4 and implement in Chapter 5, the win-rate of "random" is approximately 50%. The results of "random" VS "random" for 100 times which contain the winner the left stones count and the value of the left stones. The first player is white so the left player always plays white stones. In addition,
Value = the amount of stone in one stonebar * the distance between the stonebar and home
Then calculate the average of these results. Due to the size of all outputs, I only copy some data from the original output. See Figure 28. The win-rate is 50.58%.

testID	winner	count	value
1	BLACK	14 308	
2	BLACK	15 229	
3	BLACK	2	48
4	WHITE	6	6
5	WHITE	15 141	
6	WHITE	15 78	
7	WHITE	15 83	
8	BLACK	12 278	
9	BLACK	15 276	
10	BLACK	6	132
.....			
9991	BLACK	15 240	
9992	BLACK	15 263	
9993	WHITE	15 117	
9994	WHITE	15 136	
9995	BLACK	15 250	
9996	WHITE	15 50	
9997	BLACK	15 260	
9998	BLACK	15 304	
9999	WHITE	8	15
10000	BLACK	15 314	
5058			

Figure 28: The results of "random" VS "random" for 10000 times and the final average, including the winner (white first), the left stones count and the value of the left stones. Value = the amount of stone in one stonebar * the distance between the stonebar and home.

6.1.3 "simple"

Due to Simple Machine player evolves from "random", this player only needs to play against with "human" and "random". The first one is almost same as the test between "human" and "random", so it does not make much sense to print the process of this testing.

The second test is to prove that "simple" is more intelligent than "random". The output is in Figure 29 including the winner, the left stones count and the value of the left stones. Finally, the win-rate is 67.02%.

testID	winner	count	value
1	BLACK	15	316
2	WHITE	4	9
3	WHITE	10	55
4	BLACK	3	63
5	BLACK	15	321
6	WHITE	15	43
7	WHITE	15	137
8	BLACK	4	90
9	BLACK	8	190
10	WHITE	11	19
.....			
9991	WHITE	15	75
9992	BLACK	8	171
9993	WHITE	6	32
9994	BLACK	2	43
9995	BLACK	11	252
9996	BLACK	15	249
9997	BLACK	11	251
9998	WHITE	5	6
9999	BLACK	15	356
10000	BLACK	15	293
3298			

Figure 29: The results of "simple" VS "random" for 10000 times and the final average, including the winner (white first), the left stones count and the value of the left stones. Value = the amount of stone in one stonebar * the distance between the stonebar and home.

6.1.4 "ANNP"

The tests referred to "ANNP" are "ANNP" VS "random" and "ANNP" VS "simple". Their competition result shows the ultimate ability of "ANNP" which learns from self-play all the time.

- "ANNP" VS "random" & "ANNP" VS "simple"
  This diagram 30 and 31 both show a up-towards trend which presents the ability of "ANNP" has been improved through the self-play and complete the aim of co-evolution, though the win-rate is not much (the optimal "ANNP" should have nearly 100% win-rate). The weights all begin with zero and only 5 neurons (3 layers: 2,1,2) comprise this ANN. In addition, more neurons would not have more efficient progress and the neurons location is related to the final result. So I make a table with different amount of neurons and locations. (see more details in the table 12 and Appendix A and B)

Amount of Neurons	Location	Polynomial Fitting	R^2	Figure Number
5	2,1,2	y = -0.0027x2 + 0.3483x + 58.675	0.21637	30
5	1,2,2	y = -0.0012x2 + 0.1419x + 54.888	0.04026	32
5	3,2	y = -0.0014x2 + 0.1838x + 54.66	0.07843	33
10	4,2,4	y = -0.0007x2 + 0.083x + 56.389	0.015	34

Table 12: Different Amount of Neurons and Locations

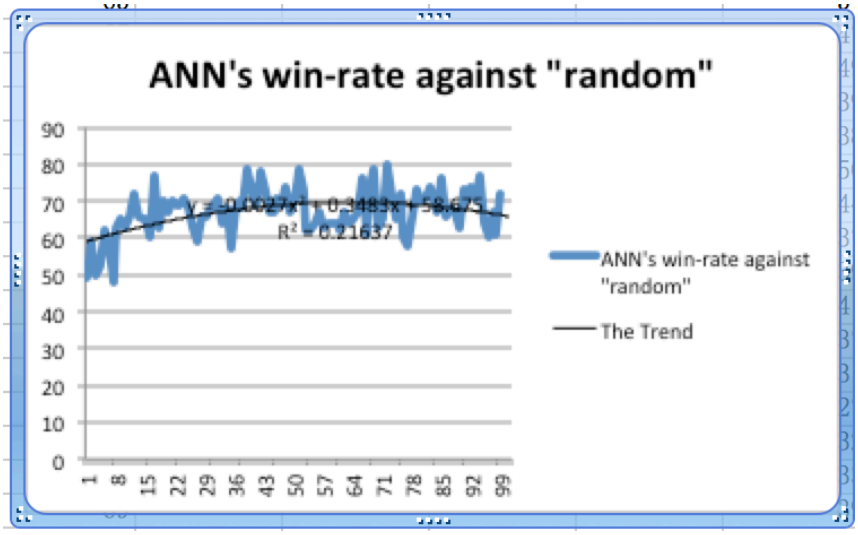


Figure 30: "ANNP" has been trained for 100 times first. Then use the best "ANNP" to test with "random". This cycle would be repeat for 100 times as well. This chart shows the trend of "ANNP" VS "simple" for 100 times and helps us to see the progress of "ANNP".

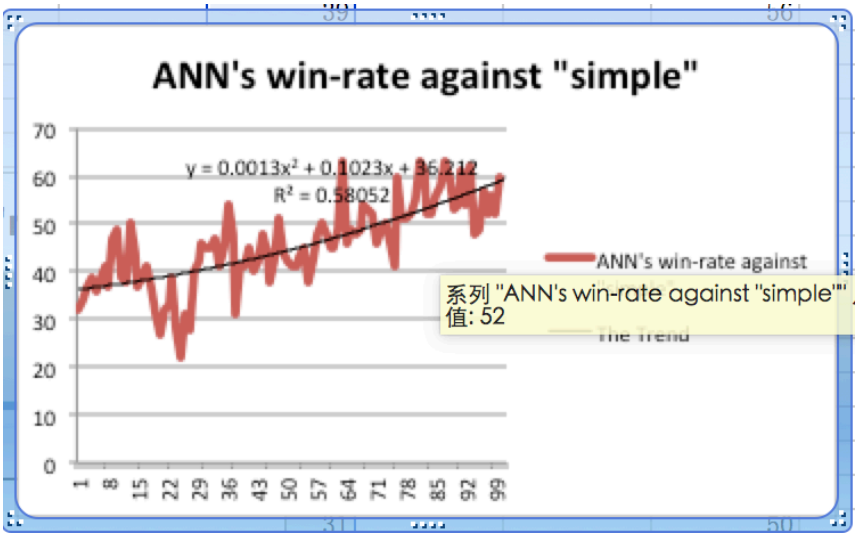


Figure 31: "ANNP" has been trained for 100 times first. Then use the best "ANNP" to test with "simple". This cycle would be repeat for 100 times as well. This chart shows the trend of "ANNP" VS "simple" for 100 times and helps us to see the progress of "ANNP".

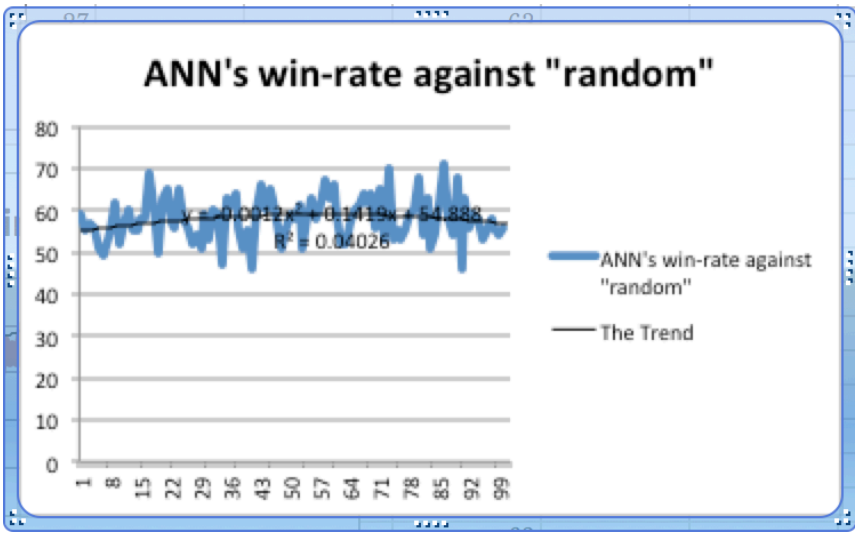


Figure 32: This ANN has three layers, the structure is 1,2,2. "ANNP" has been trained for 100 times first. Then use the best "ANNP" to test with "simple". This cycle would be repeat for 100 times as well. This chart shows the trend of "ANNP" VS "simple" for 100 times and helps us to see the progress of "ANNP".

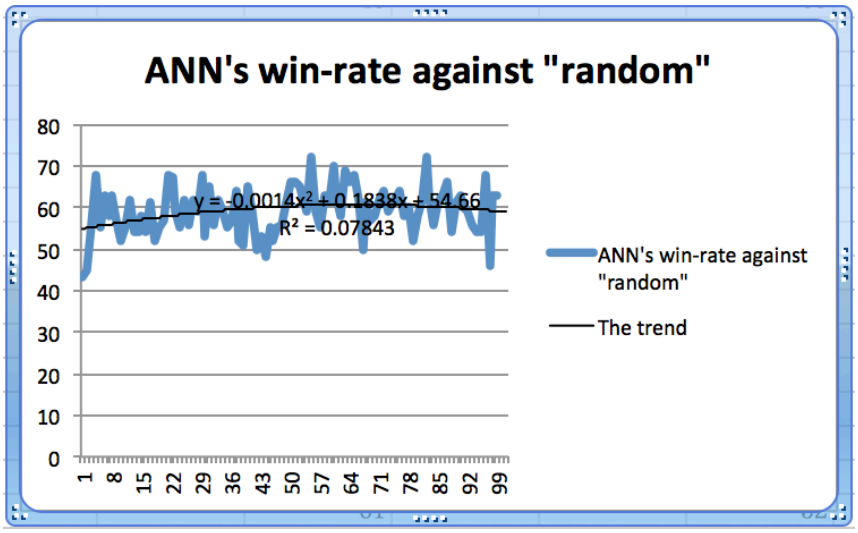


Figure 33: This ANN has two layers, the structure is 3,2. The trend of "ANNP" VS "simple" for 100 times"ANNP" has been trained for 100 times first. Then use the best "ANNP" to test with "simple". This cycle would be repeat for 100 times as well. This chart shows the trend of "ANNP" VS "simple" for 100 times and helps us to see the progress of "ANNP".

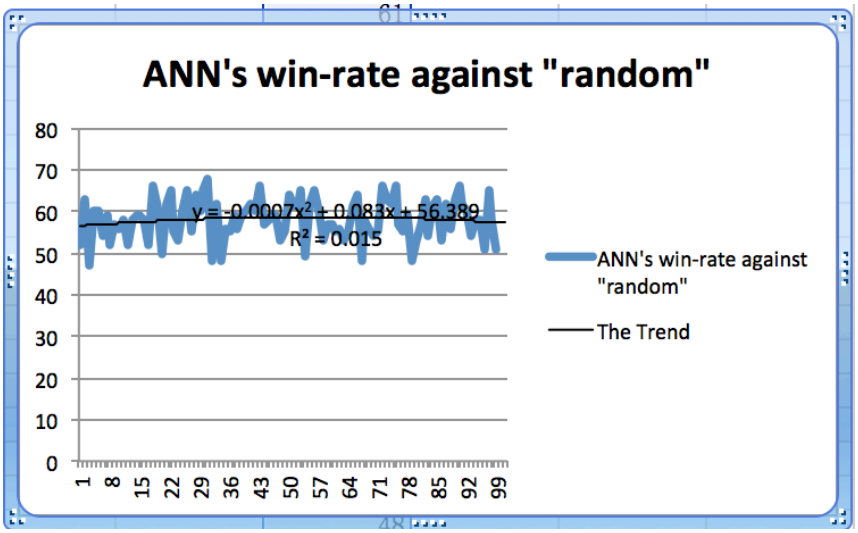


Figure 34: This ANN has three layers, the structure is 4,2,4. “ANNP” has been trained for 100 times first. Then use the best “ANNP” to test with “simple”. This cycle would be repeat for 100 times as well. This chart shows the trend of “ANNP” VS “simple” for 100 times and helps us to see the progress of “ANNP”.

From the table 12 and four figures I can get the current best ANN in ”ANN.txt”, the last several lines are the weights of ANN. So the best ANN model has 3 layers, 2,1,2 and each neuron connect to the formal layer. The best ANN model should be like figure 35 and the data should be in figure 36. The modification process of weights show in Appendix C.

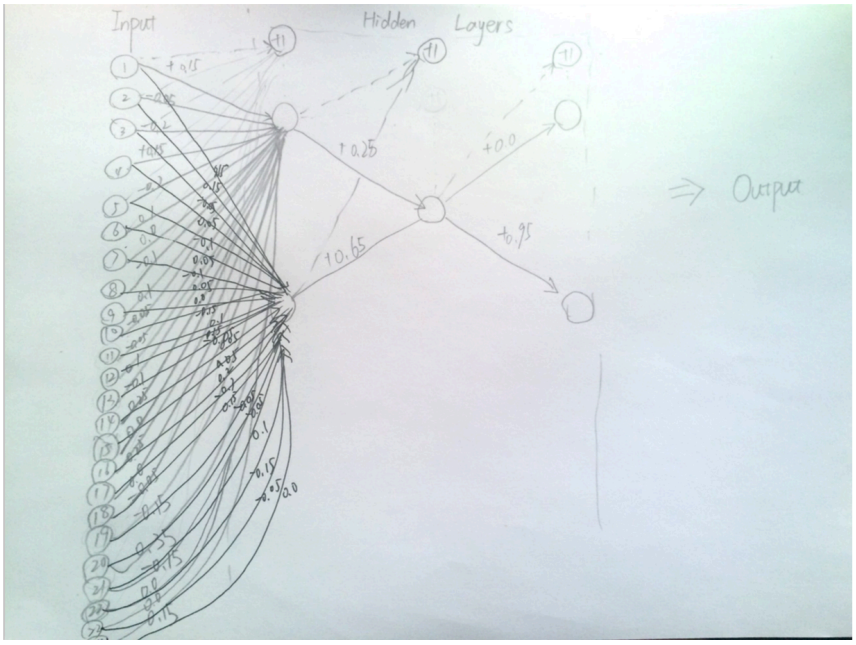


Figure 35: This is the structure of ANN model and the data of weights are in Figure 36

```
0.15  -0.05  -0.2  0.15  -0.2  0.1  0.0  -0.1  0.1  -0.05  -0.05  0.1  -0.1  0.15  -0.1  -0.25  0.0  -0.05  -0.15  0.35  -0.15  0.0  0.0  0.15
0.15  0.15  -0.05  0.05  -0.1  0.05  -0.15  0.0  0.0  -0.15  0.1  0.15  -0.05  0.05  0.05  0.2  -0.3  0.15  -0.05  -0.05  0.1  -0.15  -0.05  0.0

-0.25  -0.65

0.0
0.95
```

Figure 36: The data of weights of neuron in each layer

6.2 Reflection

This project is a whole-year project so I spent plenty of time in finishing this project as most as I can. This section is aiming at reflect on two objects including the whole project, self assessment on my own work and what I can improve and further work.

6.2.1 Project

The project reflection contains time management, the implementation of project and the final result.

1. Project Plan & Management
   Firstly, I plan my project into two parts at the beginning of project, "Backgammon" and "ANN". Then when I finished interim report, I found that I should reset the project plan and the final plan is Table 13.

Project Item Name	Start Day	End Day	Status
Proposal	03/10/2016	28/10/2016	Completed
ANN Reading	22/10/2016	05/11/2016	Completed
Demo in Java	22/10/2016	09/11/2016	Completed
Strategy Reading	3/10/2016	28/10/2016	Completed
Human Player	07/11/2016	25/11/2016	Completed
Random Machine Player	25/11/2016	05/12/2016	Completed
Interim Report	07/10/2016	17/12/2016	Completed
More Related Reading	11/02/2017	25/02/2017	Completed
Simple Machine Player	04/03/2017	04/03/2017	Completed
Reading on co-evolution	05/03/2017	20/03/2017	Completed
ANN Player	20/03/2017	23/04/2017	Completed
Final Report	10/04/2017	24/04/2017	Completed

Table 13: Project Management– items, start day, end day and the status

   Thanks to my supervisor of this project, John Cartlidge, he always appointments a short meeting when I feel confused on one subject or I need more suggestion leading to next stage. Owing to the project plan and under the supervision of my supervisor, I can finish this project on time.

2. Project Implementation
   However, the implementation is not easy during the process that I nearly did not finish my testing part. When I did the testing on "ANNP", I found my testing result is not same or similar to what I expect. So it must be error. When code goes awry, failing to produce some expected result or crashing outright, I want answers to the Four w's: where, what, why, and when. As a result, I spent much time than I presumed in the project plan. I have to delay the next work that it almost led to the failure.

3. Project result
   My aim is to find the trend that "ANNP" get progress named co-evolution through the self-play method like Pollack and Blair. And the trend should be rose up dramatically at the beginning, then grow slowly and reach a plateau. Finally, the best "ANNP" can beat other players easily. The actual results of experiments prove my estimations.

6.2.2 Self Evaluation & Improvement

Approximately seven months ago, I chose "ANN" as my final year project because I was intensely interested in Machine Learning and "ANN" is a branch of Machine Learning. As a result, I chose this theme even my supervisor had suggested that this project should be hard to complete. I am not regret to choose this project because it is a great opportunity for intellectual development and Capacity enhancement that it has gifted me throughout the past seven months.

Put aside these words, I meet many challenges and make numerous mistakes during this process. Due to the interim report, the first problem is the lack of understanding of related work. Consequently, I read more papers and reports even some papers are not what I want. The second problem is the testing result which delays the progress of whole project. Because the training result is not as same as I expect, I have to spend much time examining and reviewing code over and over. I waste a lot of precious time because of some unnecessary mistakes. The last problem is the report because I shift the focus and modify the structure for several times.

These three main problems make me have to delay the progress of project and waste a lot of time, so I should prepare much more time What if something unexpected happens and remind me all the time. Besides, improving techniques of writing reliable code is another solution.

6.2.3 Further Work

Mentioned in my aims, I want this work could be applied to a more realistic version of the game. Apart from backgammon game, I want to test other board games using the same method.Besides, the performance of the resulting optimal ANN can be further improved. Firstly, it needs much more experiments on different kinds of ANNs with different weights. After such experiments, it is possible that the best ANN will perform better than the current one. Secondly, the initial weights could learn from the existed intelligent players, such as "simple". It may improves the efficiency of training.

Except for these further work on backgammon project, the kind of ANN is another important point. There are more than 100 kinds of artificial neural network models used by researchers. Besides, with the development of application research, more and more new artificial neural network models are introduced to the public. More ANNs would be implemented in more realistic games rather than confined to the research on algorithms. Thus, changing this ANN may be the another solution to improve the result.

7 Conclusion

This paper has introduced the history of backgammon game and the debate between Teasuro and Pollack, Blair who had made substantial contributions to machine learning and hill-climbing algorithm respectively. Pollack and Blair found that hill-climbing, a typical kind of Artificial intelligence algorithm, is enough for training instead of adopting complex algorithm, temporal difference learning proposed by Teasuro. Hence the focus of this report is to demonstrate these thoughts by implementing co-evolution through self-play. The result has supportively proved the feasibility of co-evolution through self-play without temporal difference learning. In addition, the layer number and the amount of neurons and the weights all have influence on the final testing. The results show that the amount of neurons and their locations and connections would have effect on the process of training directly. Although the current ANN model for this question was not best, this evolutionary backgammon player can be worked in a realistic backgammon background.

References

[1] 4399.com. Rules of backgammon. Available at: http://www.bkgm.com/rules.html. Accessed April 11, 1017.

[2] bkgm.com. Backgammon game. Available at: http://www.4399.com/flash/71113_2.htm, 2011. Accessed April 20, 1017.

[3] Blair, A. D., and Pollack, J. B. What makes a good co-evolutionary learning environment. *Australian Journal of Intelligent Information Processing Systems 4*, 3/4 (1997), 166–175.

[4] Brown, J. S., and Vincent, T. L. Coevolution as an evolutionary game. *Evolution* (1987), 66–79.

[5] Campbell, M., Hoane, A. J., and Hsu, F.-h. Deep blue. *Artificial intelligence 134*, 1-2 (2002), 57–83.

[6] Gauci, J., and Stanley, K. O. Autonomous evolution of topographic regularities in artificial neural networks. *Neural computation 22*, 7 (2010), 1860–1898.

[7] Gelly, S., Kocsis, L., Schoenauer, M., Sebag, M., Silver, D., Szepesvári, C., and Teytaud, O. The grand challenge of computer go: Monte carlo tree search and extensions. *Communications of the ACM 55*, 3 (2012), 106–113.

[8] Keeler, E. B., and Spencer, J. Optimal doubling in backgammon. *Operations Research 23*, 6 (1975), 1063–1071.

[9] Kotnik, C., and Kalita, J. The significance of temporal-difference learning in self-play training td-rummy versus evo-rummy. In *ICML* (2003), pp. 369–375.

[10] M., D. A history of backgammon. backgammon. Available at: http://www.bkgm.com/articles/GOL/Nov00/mark.htm. Accessed April 11, 1017.

[11] Pollack, J. B., and Blair, A. D. Co-evolution in the successful learning of backgammon strategy. *Machine learning 32*, 3 (1998), 225–240.

[12] Pollack, J. B., Blair, A. D., and Land, M. Coevolution of a backgammon player. In *Artificial Life V: Proc. of the Fifth Int. Workshop on the Synthesis and Simulation of Living Systems* (1997), Cambridge, MA: The MIT Press, pp. 92–98.

[13] Poncela, J., Gómez-Gardeñes, J., Traulsen, A., and Moreno, Y. Evolutionary game dynamics in a growing structured population. *New Journal of Physics 11*, 8 (2009), 083031.

[14] Samuel, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of research and development 3*, 3 (1959), 210–229.

[15] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *Nature 529*, 7587 (2016), 484–489.

[16] Sklar, E., Blair, A. D., and Pollack, J. B. Co-evolutionary learning: Machines and humans schooling together. In *Workshop on Current Trends and Applications of Artificial Intelligence in Education: 4th World Congress on Expert Systems* (1998), Citeseer.

[17] Tesauro, G. Practical issues in temporal difference learning. *Machine learning 8*, 3-4 (1992), 257–277.

[18] Tesauro, G. Temporal difference learning and td-gammon. *Communications of the ACM 38*, 3 (1995), 58–68.

[19] Tesauro, G. Comments on "co-evolution in the successful learning of backgammon strategy". *Machine Learning 32*, 3 (1998), 241–243.

[20] Thorp, E. O. Backgammon: the optimal strategy for the pure running game. *Optimal Play: Mathematical Studies of Games and Gambling. Institute for the Study of Gambling and Commercial Gaming, University of Nevada, Reno* (2007), 237–265.

[21] Watson, R. A., and Pollack, J. B. Coevolutionary dynamics in a minimal substrate. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation* (2001), Morgan Kaufmann Publishers Inc., pp. 702–709.

[22] Wiering, M. A., et al. Self-play and using an expert to learn to play backgammon with temporal difference learning. *Journal of Intelligent Learning Systems and Applications 2*, 02 (2010), 57.

A ANN Player VS Random Machine Player & ANN Player VS Simple Machine Player

The first column shows the win-rate of “ANNP” in the competition with “random” and it is a upward trend. The average win-rate (67%) is much higher than the initial win-rate (49%). The second column shows the win-rate of “ANNP” in the competition with “simple” and it is a upward trend as well.

ANN's win-rate against "random"	ANN's win-rate against "simple"
49	32
59	34
50	37
52	39
62	36
57	39
48	41
63	37
65	47
63	49
66	39
72	38
66	50
65	44
65	37
60	39
77	41
63	37
70	31
67	27
70	32
69	33
69	39
71	28
68	22
63	31
59	28
66	40
66	42
67	46
69	45
71	45
64	47
68	41
57	45
68	54
70	48
68	31
79	41
73	41
69	45
78	40
73	42
67	48
67	44
71	38
69	42
74	51
67	44
70	42
79	41
73	41

62	43
62	45
64	38
67	42
63	48
64	50
64	48
64	45
62	45
67	49
64	63
65	46
67	49
76	48
61	49
73	54
79	53
63	52
65	46
80	49
72	50
65	46
72	41
60	60
58	51
65	51
73	52
70	55
71	63
74	61
72	52
67	52
76	56
66	58
68	63
67	60
63	53
73	54
70	61
74	54
71	62
77	48
64	49
60	57
67	52
61	56
72	52
==========	==========
66.97979798	45.47474747

B Different ANN Player VS Random Machine Player

The first column shows the win-rate of "ANNP" whose ANN structure is 1,2,2. The second column shows the win-rate of "ANNP" whose ANN structure is 3,2. The last column shows the win-rate of "ANNP" whose ANN structure is 4,2,4.

1,2,2	3,2	4,2,4
59	43	52
55	45	63
57	56	47
56	68	60
51	55	60
49	63	54
53	58	59
55	63	52
62	57	57
52	52	56
57	55	58
60	62	52
55	54	58
55	54	59
58	58	59
58	54	58
69	61	52
61	52	66
50	55	61
63	57	50
65	68	62
57	67	65
56	59	55
65	55	53
58	62	60
56	56	65
52	62	55
54	59	64
51	68	60
59	53	65
53	65	68
60	56	48
59	62	62
47	59	48
63	55	56
61	57	55
64	64	59
56	52	56
51	51	59
55	65	60
46	59	62
60	50	60
66	53	66
60	48	57
65	55	58
63	52	59
57	55	59
51	56	53
56	61	55
59	66	64
58	66	61
61	65	62
51	63	65

58	59	49
63	72	62
58	59	65
61	55	60
67	63	53
63	61	57
66	70	57
59	63	55
52	58	56
52	69	53
57	66	55
60	68	61
61	62	64
64	50	48
62	61	58
64	57	56
56	58	54
65	62	55
53	64	66
70	59	63
53	61	62
55	63	66
53	64	57
55	58	55
59	60	59
60	52	48
68	58	53
54	62	57
63	72	63
51	63	54
54	56	61
62	61	63
71	63	53
58	66	62
54	54	56
68	61	63
46	63	66
63	60	62
56	59	60
59	56	54
58	54	58
53	54	58
55	68	51
58	46	65
56	63	58
54	63	51
59	60	56

C The modification process of weights in each layers for 100 times

After 100 times competition between “ANNP” and another “ANNP”, the best ANN would be kept and then modify one weight based on the best ANN. In addition, the best “ANNP” would have competition with “random” and “simple” that could test the ability of “ANNP”. Then repeat the training. After finishing the cycle, the final weights would be seen as the best “ANNP”.

```
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

0.0  0.0

0.0
0.0

Time: 0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.05  0.0  0.0  0.0  −0.05  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.0  0.05
0.05  0.0  0.0  0.0  0.0  0.0  0.0  0.0  −0.05  0.05  0.0

0.0  −0.05

−0.05
0.0

Time: 1
0.0  −0.05  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.1  0.0  0.0  0.0  −0.05  0.0
0.05  0.0  0.0  0.0  0.0  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.0  0.05
0.05  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  −0.05  0.05  0.0

−0.1  −0.05

−0.05
0.0

Time: 2
0.0  −0.05  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.1  0.0  0.0  0.0  −0.05  0.0
0.05  0.0  0.05  0.0  0.0  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.0  0.05
0.05  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  −0.05  0.05  0.0

−0.1  −0.05

−0.05
0.0

Time: 3
0.0  −0.05  0.05  −0.05  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.1  0.0  0.0  0.0  −0.05  0.0
0.05  −0.05  0.05  −0.05  0.05  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.0
0.0  0.05  0.0  0.0  0.0  0.0  0.0  0.0  0.0  −0.05  0.05  0.0

−0.15  0.0

0.1
0.1

−0.25
0.15

Time: 14
0.0  0.0  0.05  −0.05  0.05  −0.05  0.05  0.1  0.05  0.0  0.0  0.0
−0.05  0.05  0.0  0.0  0.0  −0.05  0.1  0.0  0.0  0.0  −0.1  0.0
0.05  −0.1  0.05  −0.1  0.05  −0.05  −0.05  −0.05  0.0  0.0  0.0
0.0  0.05  0.05  −0.1  −0.05  0.0  0.0  −0.05  0.0  0.05  0.0  0.05
0.05

−0.2  0.1

−0.15
0.25

Time: 15
0.0  0.0  0.05  −0.05  0.05  0.0  0.05  0.1  0.05  0.0  0.0  0.0
−0.05  0.05  0.0  0.0  0.0  −0.05  0.1  0.0  0.0  0.0  −0.1  0.0
0.05  −0.1  0.05  −0.1  0.05  −0.05  −0.05  −0.05  0.0  0.0  0.0
0.0  0.05  0.05  −0.1  −0.05  0.0  0.0  −0.05  0.0  0.05  0.0  0.05
0.05

−0.2  0.1

−0.15
0.2

Time: 16
0.0  0.0  0.05  −0.05  0.05  0.0  0.05  0.1  0.05  0.0  0.0  0.0
−0.05  0.05  0.0  0.0  0.0  −0.05  0.1  0.0  0.0  0.0  −0.1  0.0
0.05  −0.1  0.05  −0.1  0.05  −0.05  −0.05  −0.05  0.0  0.0  0.0
0.0  0.05  0.05  −0.1  −0.05  0.0  0.0  0.0  0.0  0.05  0.0  0.05
0.05

−0.2  0.1

−0.15
0.3

Time: 17
0.0  0.0  0.05  −0.05  0.05  0.0  0.05  0.1  0.05  0.0  0.0  0.0
−0.05  0.05  0.0  0.0  0.0  −0.05  0.1  0.0  0.0  0.0  −0.1  0.0
0.05  −0.1  0.05  −0.1  0.05  −0.05  −0.05  −0.05  0.0  0.0  0.0
0.0  0.1  0.05  −0.1  −0.05  0.0  0.0  0.0  0.0  0.05  0.0  0.05
0.05

−0.2  0.1

−0.15
0.3

Time: 18
0.0  0.0  0.05  −0.05  0.05  0.0  0.05  0.1  0.05  0.0  0.0  0.0

−0.05  0.05  0.0  0.0  0.0  −0.05  0.1  0.0  0.0  0.0  −0.1  0.0
0.05  −0.1  0.05  −0.1  0.05  −0.05  −0.05  −0.05  0.0  0.0  0.05
0.0  0.1  0.05  −0.1  −0.05  0.0  −0.05  0.0  0.0  0.05  0.0  0.05
0.05

−0.2  0.1

−0.15
0.35

Time: 21
0.0  0.0  0.05  −0.05  0.05  0.0  0.05  0.1  0.05  0.0  0.0  0.0
−0.05  0.05  0.0  0.0  0.0  −0.05  0.1  0.0  0.0  0.0  −0.1  0.0
0.05  −0.1  0.05  −0.1  0.05  −0.05  0.0  −0.05  0.0  0.0  0.05  0.0
0.1  0.05  −0.1  −0.05  0.0  −0.05  0.0  −0.05  0.05  0.0  0.05
0.05

−0.15  0.15

−0.2
0.35

0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

0.0  0.0

0.0
0.0

Time: 0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  −0.05  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.05  0.0  0.0  0.0  0.0

0.0  0.0

0.0
0.0

Time: 1
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  −0.1  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.05  0.0  0.0  0.0  0.0

0.0  0.0

0.0
0.0

Time: 2
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  −0.1  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.0  0.0  0.0

0.1  −0.05

−0.05
0.0

Time: 3
0.05  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  −0.1  0.0  0.0
0.0  0.0  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.0  0.0  0.0

0.2  0.1

0.0
0.0

Time: 4
0.05  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  −0.1  0.0  0.0
0.0  0.0  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.0  0.0  0.0

0.2  0.1

0.05
0.0

Time: 5
0.05  0.0  0.0  0.0  0.0  0.0  0.05  0.0  0.0  0.0  −0.1  0.0  0.0
0.0  0.0  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  −0.05  0.0  0.05  −0.05
0.0  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.0  0.0  0.0

0.15  0.15

0.05
0.1

Time: 6
0.05  0.0  0.0  0.0  0.0  0.0  0.05  0.0  0.0  0.0  −0.1  0.0  0.0
0.0  0.0  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  −0.05  0.0  0.05  −0.05
0.0  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.0  0.0  0.0

0.2  0.15

0.05
0.1

Time: 7
0.05  0.0  0.0  0.0  0.0  0.0  0.05  0.0  0.0  0.0  −0.1  0.0  0.0
0.0  0.0  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  −0.05  0.0  0.05  −0.05
0.0  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.0  0.0  0.0

0.25  0.15

0.05
0.1

Time: 8
0.05  0.0  0.0  0.0  0.0  0.0  0.05  0.0  0.0  0.0  −0.1  −0.05  0.0
0.0  0.0  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  −0.05  0.0  0.05  −0.05
0.0  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.0  0.0  0.0

0.3  0.2

0.05
0.1

Time: 9
0.05  0.0  0.0  0.0  0.0  0.0  0.05  0.0  0.0  0.0  −0.1  −0.05  0.0
0.0  0.0  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  −0.05  0.0  0.1  −0.05
0.0  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.0  0.0  0.0

0.3  0.25

−0.1

0.1

Time: 10
0.05  0.0  0.0  0.0  0.0  0.0  0.05  0.0  0.0  0.0  −0.1  −0.05  0.0
0.0  0.0  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.0  0.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  −0.05  0.0  0.1  −0.05
0.0  0.0  0.0  −0.1  0.0  0.0  0.0  0.0  0.0  0.0  0.0

0.35  0.25

−0.05
0.15

Time: 11
0.05  −0.05  0.0  0.0  0.0  0.0  0.1  0.0  0.0  0.0  −0.1  −0.05
0.0  0.0  0.0  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.05  0.0
0.0  0.0  0.0  0.05  −0.05  0.0  0.0  0.0  0.0  −0.1  0.0  0.1  −0.05
0.0  0.0  0.0  −0.15  0.0  0.0  0.0  0.0  0.0  0.0  0.0

0.35  0.15

−0.1
0.15

Time: 12
0.05  −0.05  0.0  0.0  0.0  0.0  0.1  0.0  0.0  0.0  −0.1  −0.05
0.0  0.0  0.0  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.05  0.0
0.0  0.0  0.0  0.05  −0.05  0.0  0.0  0.0  0.0  −0.1  0.0  0.1
−0.05  0.05  0.0  0.0  −0.15  0.0  0.0  0.0  0.0  0.0  0.0

0.35  0.15

−0.1
0.15

Time: 13
0.05  0.0  0.0  0.0  0.0  0.0  0.1  −0.05  0.0  0.0  −0.1  −0.05
0.0  0.0  0.0  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.05  0.0
0.0  0.0  0.0  0.05  −0.05  0.05  0.0  0.0  0.0  −0.1  0.0  0.1
−0.05  0.05  0.0  0.0  −0.15  0.0  0.0  0.0  0.0  0.0  0.0

0.3  0.25

−0.1
0.2

Time: 14
0.05  0.0  0.0  0.0  0.0  0.0  0.1  −0.05  0.0  0.0  −0.1  −0.05
0.0  0.0  0.0  0.0  0.0  −0.05  0.0  0.0  0.0  0.0  0.05  0.0
0.0  0.0  0.0  0.05  −0.05  0.05  0.0  0.0  0.0  −0.1  0.0  0.1
−0.05  0.05  0.0  0.0  −0.15  0.0  0.0  −0.05  0.0  0.0  0.0  0.0

0.3  0.25

−0.1
0.2

Time: 15
0.05  0.0  0.0  0.0  −0.05  0.0  0.15  −0.05  0.0  0.0  −0.1  −0.05
0.05  0.0  0.0  −0.05  0.0  −0.05  0.0  0.0  0.0  0.0  0.05  0.0
0.0  0.0  0.0  0.05  −0.05  0.05  0.0  0.0  0.0  −0.1  0.0  0.1
−0.1  0.05  −0.05  0.0  −0.15  0.0  −0.05  −0.05  0.0  0.0  0.0  0.0

0.35  0.3

−0.15
0.2

Time: 16
0.05  0.0  0.0  0.0  −0.05  0.0  0.15  −0.05  0.0  0.0  −0.1  0.0
0.05  0.0  0.0  −0.05  0.0  −0.05  0.0  0.0  0.0  0.0  0.05  0.0
0.0  0.0  0.0  0.05  −0.05  0.05  0.05  0.0  0.0  −0.1  0.0  0.05
−0.1  0.05  −0.05  0.0  −0.2  0.0  −0.05  −0.05  0.0  0.0  0.0
−0.05

0.3  0.2

−0.2
0.15

Time: 17
0.05  0.0  0.0  0.0  −0.05  0.05  0.15  −0.05  0.0  0.0  −0.1  0.0
0.05  0.0  0.05  −0.05  0.0  −0.05  0.0  0.0  0.0  0.0  0.1  0.0
0.1  0.0  0.0  0.05  −0.05  0.1  0.05  0.0  0.0  −0.1  0.0  0.05
−0.1  0.05  −0.05  0.0  −0.2  0.0  −0.05  −0.05  0.0  0.0  0.0
−0.05

0.2  0.2

−0.3
0.25

Time: 18
0.0  0.05  0.0  0.0  −0.05  0.05  0.15  −0.05  −0.05  0.0  −0.1  0.0
0.05  0.0  0.05  −0.1  0.0  −0.1  −0.15  0.0  0.0  0.0  0.1  0.0
0.1  0.05  0.05  0.05  −0.05  0.1  0.05  0.0  −0.05  −0.1  0.0  0.05
−0.1  0.0  −0.1  0.0  −0.2  0.0  −0.05  −0.05  0.0  0.05  0.0  −0.05

0.35  0.35

−0.2
0.25

Time: 19
0.0  0.05  0.0  0.0  −0.05  0.05  0.15  −0.05  −0.05  0.0  −0.1  0.0
0.05  0.0  0.05  −0.1  0.0  −0.1  −0.15  0.0  0.0  0.0  0.1  0.0
0.1  0.05  0.05  0.05  −0.05  0.1  0.05  0.0  −0.05  −0.1  0.0  0.05
−0.1  0.0  −0.1  0.0  −0.2  0.0  −0.05  −0.05  0.0  0.05  0.0  −0.05
```

```
0.35  0.35

-0.2
0.35

Time: 20
0.0  0.05  0.0  0.0  -0.05  0.05  0.15  -0.05  -0.05  0.0  -0.1  0.0
0.05  0.0  0.05  -0.1  0.0  -0.1  -0.15  0.0  0.0  0.0  0.1  0.0
0.1  0.05  0.05  0.05  -0.05  0.1  0.05  -0.05  -0.05  -0.1  0.0
0.05  -0.1  0.0  -0.1  0.0  -0.2  0.0  -0.05  0.0  0.0  0.05  0.0
-0.05

0.3  0.35

-0.25
0.5

Time: 21
0.0  0.05  0.0  0.0  -0.05  0.05  0.15  -0.05  -0.05  0.0  -0.1  0.0
0.05  0.0  0.05  -0.1  0.0  -0.1  -0.15  0.1  0.0  0.0  0.1  0.0
0.1  0.05  0.05  0.05  -0.05  0.1  0.05  -0.05  -0.05  -0.1  0.0
0.05  -0.1  0.0  -0.1  0.0  -0.2  0.0  -0.05  0.0  0.0  0.05  0.0
-0.05

0.3  0.35

-0.25
0.5

Time: 22
0.0  0.05  0.0  0.0  -0.05  0.05  0.15  -0.05  -0.05  0.0  -0.15
0.0  0.05  0.0  0.05  -0.1  0.0  -0.05  -0.15  0.1  0.0  0.0  0.1
0.0
0.1  0.05  0.05  0.05  -0.05  0.1  0.05  -0.05  -0.05  -0.1  0.0
0.05  -0.1  0.0  -0.1  0.0  -0.2  0.0  -0.05  0.0  0.0  0.05  0.0
-0.05

0.25  0.25

-0.3
0.5

Time: 23
0.0  0.05  0.0  0.0  -0.05  0.05  0.15  -0.05  -0.05  0.0  -0.15
0.0  0.05  0.0  0.05  -0.1  0.0  -0.05  -0.15  0.1  0.0  0.0  0.1
0.0
0.1  0.05  0.05  0.05  -0.05  0.1  0.0  -0.05  -0.05  -0.1  0.0
0.05  -0.1  0.0  -0.1  0.0  -0.2  0.0  -0.05  0.0  0.0  0.05  0.0
-0.05

0.15  0.2

-0.25

0.55

Time: 24
0.0  0.05  0.0  0.0  -0.05  0.05  0.15  -0.05  -0.05  0.0  -0.15
0.0  0.05  0.0  0.05  -0.1  0.0  -0.05  -0.15  0.1  0.0  0.0  0.1
0.0
0.1  0.05  0.05  0.05  -0.05  0.1  0.0  -0.05  -0.05  -0.1  0.0
0.05  -0.1  0.0  -0.1  0.0  -0.2  0.0  -0.05  0.0  0.05  0.05  0.0
-0.05

0.15  0.2

-0.25
0.55

Time: 25
-0.05  0.05  0.0  0.0  -0.05  0.05  0.15  -0.05  -0.05  0.0  -0.15
0.0  0.05  0.0  0.05  -0.1  0.0  -0.05  -0.15  0.1  0.0  0.0  0.1
0.0
0.1  0.05  0.05  0.05  -0.05  0.1  0.0  -0.05  -0.05  -0.1  0.0
0.05  -0.1  0.0  -0.1  0.0  -0.2  0.0  -0.05  0.0  0.05  0.0  0.0
-0.05

0.15  0.25

-0.3
0.5

Time: 26
-0.05  0.05  0.0  0.0  -0.05  0.05  0.15  -0.05  -0.05  0.0  -0.15
-0.05  0.05  0.0  0.05  -0.15  0.0  0.0  -0.05  -0.15  0.1  0.0  0.1
0.0
0.1  0.05  0.0  0.05  -0.05  0.1  0.0  -0.05  -0.05  -0.1  0.0  0.05
-0.1  0.0  -0.1  0.0  -0.2  0.0  -0.05  0.0  0.1  0.0  0.0  -0.05

0.25  0.2

-0.4
0.5

Time: 27
-0.05  0.05  0.0  0.0  -0.05  0.05  0.15  -0.05  -0.05  0.0  -0.15
-0.05  0.05  0.0  0.05  -0.15  0.0  0.0  -0.15  0.1  0.0  0.0  0.1
0.0
0.1  0.0  0.0  0.05  -0.05  0.1  0.0  -0.05  -0.05  -0.1  0.0  0.05
-0.1  0.05  -0.05  0.0  -0.2  0.0  -0.05  0.0  0.05  0.0  0.0  -0.05

0.2  0.2

-0.45
0.5

Time: 28
-0.05  0.05  0.0  0.0  -0.05  0.05  0.15  -0.05  0.0  0.0  -0.15

-0.05  0.05  0.0  0.05  -0.15  0.0  0.0  -0.15  0.15  0.0  0.0  0.1
0.0
0.1  0.0  0.0  0.05  -0.05  0.1  0.0  -0.05  -0.05  -0.1  0.0  0.05
-0.1  0.05  -0.05  0.0  -0.2  0.0  -0.05  0.0  0.05  0.0  0.0  -0.05

0.15  0.2

-0.45
0.5

Time: 29
-0.05  0.05  0.0  0.0  -0.05  0.05  0.15  -0.05  0.0  0.0  -0.15
-0.05  0.05  0.0  0.05  -0.15  0.0  0.0  -0.15  0.15  0.0  0.0  0.1
0.0
0.1  0.0  0.0  0.05  -0.05  0.1  0.0  -0.05  -0.05  -0.1  0.0  0.05
-0.1  0.05  -0.05  0.0  -0.2  0.0  -0.05  0.0  0.05  0.0  0.0  -0.05

0.2  0.2

-0.45
0.5

Time: 30
-0.05  0.05  0.0  0.0  -0.05  0.05  0.15  -0.05  0.0  0.0  -0.15
-0.05  0.05  0.0  0.05  -0.15  0.0  0.0  -0.15  0.15  0.0  0.0  0.1
0.0
0.1  0.05  0.0  0.05  -0.05  0.1  0.0  -0.05  -0.05  -0.1  0.0  0.05
-0.1  0.05  -0.05  0.0  -0.2  0.0  -0.05  0.0  0.05  0.0  0.0  -0.05

0.2  0.2

-0.45
0.5

Time: 31
0.0  0.05  0.0  0.0  -0.05  0.05  0.15  -0.05  0.0  0.0  -0.15
-0.05  0.05  0.0  0.05  -0.15  0.0  0.0  -0.15  0.15  0.0  0.0  0.1
0.0
0.1  0.05  0.0  0.05  -0.05  0.1  0.0  -0.05  -0.05  -0.1  0.0  0.05
-0.1  0.05  -0.05  0.0  -0.2  0.0  -0.05  0.0  0.05  0.0  0.0  -0.05

0.15  0.2

-0.45
0.5

Time: 32
0.0  0.05  0.0  0.0  -0.05  0.05  0.15  -0.05  0.0  0.0  -0.15
-0.05  0.05  0.0  0.05  -0.15  0.0  0.0  -0.15  0.15  0.0  0.0  0.1
0.0
0.1  0.05  0.0  0.05  0.0  0.1  0.0  -0.05  -0.05  -0.1  0.0  0.05
-0.1  0.05  -0.05  0.0  -0.2  0.0  -0.05  0.0  0.05  0.0  0.0  -0.05

0.2  0.25

-0.45
0.65

Time: 33
0.0  0.05  0.0  0.0  -0.05  0.05  0.15  -0.05  0.0  0.0  -0.15
-0.05  0.05  0.0  0.05  -0.15  0.0  0.0  -0.15  0.15  0.0  0.0  0.1
0.0
0.1  0.05  0.0  0.05  0.0  0.1  0.0  -0.05  -0.05  -0.1  0.0  0.05
-0.1  0.05  -0.05  0.0  -0.2  0.0  -0.05  0.0  0.05  0.0  0.0  -0.05

0.2  0.25

-0.35
0.65

Time: 34
0.0  0.05  0.0  0.0  -0.05  0.05  0.15  -0.05  0.0  0.0  -0.15
-0.05  0.05  0.0  0.05  -0.15  0.0  0.0  -0.15  0.15  0.0  0.0  0.1
0.0
0.1  0.05  0.0  0.05  0.0  0.1  0.0  -0.05  -0.05  -0.1  0.0  0.05
-0.1  0.05  -0.05  0.0  -0.2  0.0  -0.05  0.0  0.05  0.0  0.0  -0.05

0.25  0.25

-0.35
0.65

Time: 35
0.0  0.05  0.0  0.0  -0.05  0.05  0.15  -0.05  0.05  0.0  -0.15
-0.05  0.0  0.0  0.0  -0.15  0.0  0.0  -0.15  0.15  0.0  0.05  0.1
0.0
0.1  0.05  0.0  0.1  0.0  0.05  0.0  -0.05  -0.05  0.0  0.0  0.05
-0.05  0.05  -0.05  0.0  -0.2  0.0  -0.05  0.0  0.0  0.0  -0.1
-0.05

0.35  0.25

-0.3
0.7

Time: 36
0.0  0.05  0.0  0.0  -0.05  0.05  0.15  -0.05  0.05  0.0  -0.15
-0.05  0.0  0.0  0.0  -0.15  0.0  0.0  -0.15  0.15  0.0  0.05  0.1
0.0
0.1  0.05  0.0  0.1  -0.05  0.05  0.0  -0.05  -0.05  0.0  0.0  0.05
-0.05  0.05  -0.05  0.0  -0.2  0.0  -0.05  -0.05  0.0  0.0  -0.1
-0.05

0.35  0.25

-0.3
0.7

Time: 37
0.0  0.05  0.0  0.0  -0.05  0.05  0.15  0.0  0.05  -0.05  -0.15
-0.05  0.0  0.0  -0.05  -0.15  0.0  0.0  -0.15  0.15  0.0  0.05  0.1
0.0
0.1  0.05  0.0  0.1  -0.05  0.05  0.0  -0.05  -0.05  0.0  0.0  0.05
-0.05  0.05  -0.05  0.0  -0.2  0.0  -0.05  -0.05  0.0  0.0  -0.1
-0.05

0.4  0.25

-0.3
0.6

Time: 38
0.0  0.05  0.0  0.0  -0.05  0.05  0.15  0.0  0.05  -0.05  -0.15
-0.05  0.0  0.0  -0.05  -0.15  0.0  0.0  -0.15  0.15  0.0  0.05  0.1
0.0
0.1  0.05  0.0  0.1  -0.05  0.05  0.0  -0.05  -0.05  0.0  0.0  0.05
-0.05  0.05  -0.05  0.0  -0.2  0.0  -0.05  -0.05  0.0  0.0  -0.1
-0.05

0.4  0.25

-0.25
0.6

Time: 39
0.0  0.05  -0.05  0.0  -0.05  0.05  0.15  0.0  0.05  -0.05  -0.15
-0.05  0.0  0.0  -0.05  -0.15  0.0  0.0  -0.15  0.15  0.0  0.05  0.1
0.0
0.1  0.05  0.0  0.1  -0.05  0.05  0.0  -0.05  0.0  0.0  0.0  0.05
-0.05  0.05  0.0  0.0  -0.2  0.0  -0.05  -0.05  0.0  0.0  -0.1
-0.05

0.45  0.25

-0.25
0.6

Time: 40
0.0  0.05  -0.05  0.0  -0.05  0.05  0.15  0.0  0.05  -0.05  -0.15
-0.05  0.0  0.0  -0.05  -0.15  0.0  0.0  -0.15  0.15  0.0  0.05  0.1
0.0
0.1  0.05  0.0  0.1  -0.05  0.05  0.0  -0.05  0.0  0.0  0.0  0.05
-0.05  0.05  0.0  0.0  -0.2  0.0  -0.05  -0.05  0.0  0.0  -0.05
-0.05

0.45  0.25

-0.2
0.6

Time: 41
0.05  0.1  -0.05  0.05  -0.05  0.05  0.15  -0.05  0.05  -0.05  -0.15

-0.05  -0.05  0.0  -0.05  -0.15  0.05  0.0  -0.15  0.25  0.0  0.05
0.1  0.0
0.1  0.1  0.0  0.1  -0.05  0.1  0.0  -0.05  0.0  0.0  0.0  0.05
-0.05  0.05  0.0  0.0  -0.2  0.05  -0.05  -0.05  0.0  0.0  -0.05
-0.05

0.4  0.3

-0.2
0.65

Time: 42
0.05  0.1  -0.05  0.1  -0.05  0.05  0.1  -0.1  0.05  -0.05  -0.15
-0.05  -0.05  0.0  -0.05  -0.15  0.0  0.0  -0.15  0.25  0.0  0.05
0.15  0.0
0.15  0.1  0.0  0.1  -0.05  0.1  0.0  -0.05  0.0  0.0  0.0  0.1
-0.05  0.05  0.0  0.0  -0.2  0.05  -0.05  -0.05  0.0  0.0  -0.05
-0.05

0.4  0.35

-0.25
0.65

Time: 43
0.05  0.1  -0.05  0.1  -0.05  0.05  0.1  -0.1  0.05  -0.05  -0.15
-0.05  -0.05  0.0  -0.05  -0.15  0.0  0.0  -0.15  0.25  0.0  0.05
0.15  0.0
0.15  0.1  0.0  0.1  -0.05  0.1  0.0  -0.05  0.0  0.0  0.0  0.1
-0.05  0.05  0.0  0.0  -0.2  0.05  -0.05  -0.05  0.0  0.0  -0.05
-0.05

0.45  0.35

-0.25
0.65

Time: 44
0.05  0.1  -0.05  0.1  -0.05  0.05  0.1  -0.1  0.05  -0.05  -0.15
-0.05  -0.05  0.05  -0.1  -0.15  0.0  0.0  -0.15  0.25  0.0  0.05
0.1  0.0
0.15  0.1  0.0  0.05  -0.05  0.1  0.0  -0.05  0.0  -0.05  0.0  0.1
-0.05  0.0  0.0  0.0  -0.2  0.05  -0.05  -0.05  0.0  0.0  -0.05
-0.05

0.4  0.35

-0.25
0.65

Time: 45
0.05  0.1  -0.05  0.05  -0.05  0.05  0.1  -0.1  0.05  -0.05  -0.15
-0.05  -0.1  0.05  -0.1  -0.2  0.0  0.05  -0.15  0.25  0.0  0.05
0.1  0.0

0.15  0.1  0.0  0.05  -0.05  0.1  0.0  -0.05  0.0  0.0  0.0  0.1
-0.05  0.0  0.0  0.0  -0.2  0.05  -0.05  -0.05  0.0  0.0  -0.05
-0.05

0.5  0.15

-0.4
0.75

Time: 46
0.05  0.1  -0.05  0.05  -0.05  0.05  0.1  -0.1  0.05  -0.05  -0.15
-0.05  -0.1  0.05  -0.1  -0.2  0.0  0.05  -0.15  0.25  0.0  0.05
0.1  0.0
0.15  0.1  0.0  0.1  -0.05  0.1  0.0  -0.05  0.0  -0.05  0.0  0.1
-0.05  0.0  0.0  0.0  -0.2  0.05  -0.05  -0.05  0.0  0.0  -0.05
-0.05

0.5  0.15

-0.45
0.65

Time: 47
0.05  0.1  -0.05  0.05  -0.05  0.05  0.1  -0.1  0.05  -0.05  -0.15
-0.05  -0.1  0.05  -0.1  -0.2  0.0  0.05  -0.15  0.25  0.0  0.05
0.1  0.0
0.15  0.1  0.0  0.1  -0.05  0.1  0.0  -0.05  0.0  -0.1  0.0  0.1
-0.05  0.0  0.0  0.0  -0.2  0.05  -0.05  -0.05  0.0  0.0  -0.05
-0.05

0.5  0.2

-0.45
0.65

Time: 48
0.05  0.1  -0.05  0.0  -0.05  0.05  0.1  -0.1  0.05  -0.05  -0.15
-0.05  -0.1  0.05  -0.1  -0.2  0.0  0.1  -0.15  0.25  0.0  0.05  0.1
0.0
0.15  0.1  0.0  0.1  -0.05  0.1  0.0  -0.05  0.0  -0.1  -0.05  0.1
-0.05  0.0  0.0  0.0  -0.2  0.05  0.0  -0.05  0.0  0.0  -0.05  -0.05

0.5  0.2

-0.45
0.6

Time: 49
0.05  0.1  -0.05  0.0  -0.05  0.05  0.1  -0.1  0.05  -0.05  -0.15
-0.05  -0.1  0.05  -0.1  -0.2  0.0  0.1  -0.15  0.25  0.0  0.05  0.1
0.0
0.15  0.1  0.0  0.1  -0.05  0.1  0.0  -0.05  0.0  -0.1  -0.05  0.1
-0.05  0.0  0.0  0.0  -0.2  0.05  0.0  -0.05  0.0  0.0  -0.05  -0.05

0.45  0.25

-0.45
0.65

Time: 50
0.05  0.1  -0.05  0.0  -0.05  0.05  0.1  -0.1  0.05  -0.05  -0.15
-0.05  -0.05  0.05  -0.1  -0.2  0.0  0.1  -0.15  0.25  0.0  0.05
0.05  0.0
0.15  0.1  0.0  0.1  -0.05  0.1  0.0  -0.05  0.0  -0.1  -0.05  0.1
-0.05  0.0  0.0  0.0  -0.2  0.05  0.0  -0.05  0.0  0.0  -0.05  -0.05

0.45  0.3

-0.35
0.65

Time: 51
0.05  0.1  -0.05  0.0  -0.05  0.05  0.1  -0.1  0.05  -0.05  -0.15
-0.05  -0.05  0.05  -0.1  -0.2  0.0  0.1  -0.15  0.25  0.0  0.05
0.05  0.0
0.15  0.1  0.0  0.1  -0.05  0.1  0.0  -0.05  0.0  -0.1  -0.05  0.1
-0.05  0.0  0.0  0.0  -0.2  0.05  0.0  -0.05  0.0  0.0  -0.05  -0.05

0.4  0.3

-0.35
0.7

Time: 52
0.05  0.1  -0.05  0.0  -0.05  0.05  0.1  -0.1  0.05  -0.05  -0.15
-0.05  -0.05  0.05  -0.1  -0.2  0.0  0.1  -0.15  0.25  0.0  0.05
0.05  0.0
0.15  0.1  0.0  0.05  -0.05  0.1  0.0  -0.05  0.0  -0.1  -0.05  0.1
-0.05  0.0  0.0  0.0  -0.25  0.05  0.0  -0.05  0.0  0.0  -0.05
-0.05

0.4  0.25

-0.3
0.7

Time: 53
0.05  0.15  -0.05  0.0  -0.05  0.05  0.1  -0.1  0.05  -0.05  -0.15
-0.05  -0.05  0.05  -0.1  -0.2  0.0  0.1  -0.15  0.25  0.0  0.05
0.05  0.05
0.15  0.1  0.05  0.1  -0.05  0.1  0.0  -0.05  0.0  -0.1  -0.05  0.1
-0.05  0.0  0.0  0.0  -0.25  0.05  0.0  -0.05  0.0  0.0  -0.05
-0.05

0.4  0.25

-0.4
0.75

Time: 54
0.05  0.15  -0.1  0.0  -0.05  0.05  0.1  -0.1  0.05  -0.05  -0.15
0.0  -0.05  0.05  -0.1  -0.2  0.0  0.1  -0.15  0.2  0.0  0.05  0.05
0.1
0.15  0.1  0.05  0.1  -0.05  0.05  0.0  -0.05  0.0  -0.1  -0.05  0.1
-0.05  0.0  0.0  0.0  -0.25  0.05  0.0  -0.05  0.0  0.0  -0.05
-0.05

0.35  0.2

-0.4
0.85

Time: 55
0.05  0.15  -0.1  0.0  -0.05  0.05  0.1  -0.1  0.05  -0.05  -0.15
0.0  -0.05  0.05  -0.1  -0.2  0.0  0.1  -0.15  0.2  0.0  0.05  0.05
0.1
0.15  0.05  0.05  0.1  -0.1  0.05  0.0  -0.05  0.0  -0.1  -0.05  0.1
-0.05  0.0  0.0  0.0  -0.25  0.05  0.0  -0.05  0.0  0.0  -0.05
-0.05

0.3  0.2

-0.4
0.9

Time: 56
0.05  0.15  -0.1  0.0  -0.05  0.05  0.1  -0.1  0.05  -0.05  -0.15
0.0  -0.05  0.05  -0.1  -0.2  0.0  0.1  -0.15  0.2  -0.05  0.05
0.05  0.1
0.15  0.05  0.05  0.1  -0.1  0.05  0.0  -0.05  0.0  -0.1  -0.05  0.1
-0.05  0.05  0.0  0.0  -0.25  0.05  0.0  -0.05  0.0  0.0  -0.05
-0.05

0.35  0.15

-0.45
0.9

Time: 57
0.05  0.1  -0.1  0.0  -0.05  0.05  0.1  -0.1  0.05  -0.05  -0.15
0.0  -0.05  0.05  -0.1  -0.2  0.0  0.1  -0.15  0.2  -0.05  0.05
0.05  0.1
0.15  0.05  0.05  0.1  -0.1  0.05  0.0  -0.05  0.0  -0.1  -0.05  0.1
-0.05  0.05  0.0  0.0  -0.25  0.05  0.0  -0.05  0.05  0.0  -0.05
-0.05

0.35  0.1

-0.45
0.9

Time: 58
```

0.05 0.1 −0.1 0.0 −0.05 0.05 0.1 −0.1 0.05 −0.05 −0.15
0.0 −0.05 0.05 −0.1 −0.2 0.0 0.1 −0.15 0.2 −0.05 0.05
0.05 0.1
0.15 0.05 0.05 0.1 −0.1 0.05 0.0 −0.05 −0.05 −0.1 −0.05
0.1 −0.05 0.05 0.0 0.0 −0.25 0.05 0.0 −0.05 0.0 0.0 −0.05
−0.05

0.3 0.1

−0.45
0.95

Time: 59
0.05 0.05 −0.1 0.0 −0.05 0.05 0.1 −0.1 0.05 −0.05 −0.15
0.0 −0.05 0.05 −0.1 −0.2 0.0 0.1 −0.15 0.2 −0.05 0.05
0.05 0.1
0.15 0.05 0.05 0.1 −0.1 0.05 0.0 −0.05 −0.05 −0.1 −0.05
0.1 −0.05 0.05 0.0 0.0 −0.25 0.05 0.0 −0.05 0.0 0.0 −0.05
−0.05

0.3 0.1

−0.45
0.95

Time: 60
0.05 0.05 −0.1 0.0 −0.05 0.1 0.1 −0.1 0.05 −0.05 −0.15
0.0 −0.05 0.05 −0.1 −0.2 0.0 0.1 −0.15 0.2 −0.05 0.05 0.0
0.1
0.15 0.1 0.05 0.1 −0.1 0.05 0.0 −0.05 −0.05 −0.1 −0.05
0.1 −0.05 0.05 0.0 0.0 −0.25 0.05 0.0 −0.05 0.0 0.0 −0.05
−0.05

0.35 0.15

−0.45
0.95

Time: 61
0.05 0.0 −0.1 0.0 −0.05 0.1 0.05 −0.1 0.05 −0.05 −0.15
0.0 −0.05 0.1 −0.1 −0.2 0.0 0.1 −0.2 0.2 −0.1 0.05 0.0
0.1
0.15 0.1 0.05 0.1 −0.1 0.05 0.0 −0.05 −0.05 −0.1 −0.05
0.1 −0.05 0.05 0.05 0.0 −0.25 0.05 0.0 −0.05 0.0 0.0
−0.05 0.0

0.35 0.15

−0.45
0.9

Time: 62
0.05 0.0 −0.15 0.0 −0.05 0.1 0.05 −0.1 0.05 −0.05 −0.15
0.0 −0.05 0.1 −0.1 −0.2 0.0 0.1 −0.2 0.2 −0.1 0.05 0.0

0.1
0.15 0.1 0.15 0.1 −0.1 0.05 0.0 −0.05 −0.05 −0.1 0.0 0.1
−0.05 0.05 0.05 0.0 −0.2 0.05 0.0 −0.05 −0.05 0.0 −0.05
0.0

0.3 0.05

−0.4
0.85

Time: 63
0.0 0.0 −0.15 0.0 −0.05 0.1 0.05 −0.1 0.1 −0.05 −0.15
0.05 −0.05 0.1 −0.1 −0.2 0.05 0.05 −0.2 0.2 −0.1 0.05 0.0
0.05
0.15 0.1 0.1 0.1 −0.1 0.05 0.0 −0.05 −0.05 −0.15 0.0 0.1
−0.05 0.05 0.05 0.05 −0.25 0.05 0.0 −0.05 −0.05 0.0 −0.05
0.0

0.4 0.25

−0.6
0.55

Time: 64
0.0 0.0 −0.15 0.0 −0.05 0.1 0.05 −0.1 0.1 −0.05 −0.15
0.05 −0.05 0.15 −0.1 −0.2 0.05 0.05 −0.2 0.2 −0.1 0.05
0.0 0.05
0.15 0.1 0.1 0.1 −0.1 0.05 0.0 −0.05 −0.05 −0.15 0.0 0.1
−0.05 0.05 0.05 0.05 −0.25 0.05 0.0 −0.05 −0.05 0.0 −0.05
0.0

0.4 0.25

−0.55
0.55

Time: 65
0.0 0.0 −0.15 0.0 −0.05 0.1 0.05 −0.1 0.1 −0.05 −0.15
0.05 −0.05 0.15 −0.1 −0.2 0.05 0.05 −0.2 0.2 −0.1 0.05
0.0 0.1
0.15 0.1 0.1 0.1 −0.1 0.05 0.0 −0.05 −0.05 −0.15 0.0 0.1
−0.1 0.05 0.05 0.05 −0.25 0.05 0.0 −0.05 −0.05 0.0 −0.05
0.0

0.35 0.3

−0.55
0.55

Time: 66
0.0 0.0 −0.15 0.05 −0.05 0.1 0.05 −0.15 0.1 −0.05 −0.1
0.05 −0.05 0.15 −0.1 −0.2 0.05 0.05 −0.2 0.2 −0.1 0.05
0.0 0.1
0.15 0.1 0.1 0.1 −0.1 0.05 −0.05 −0.05 0.0 −0.15 0.0 0.1

−0.1 0.05 0.05 0.05 −0.25 0.05 0.0 −0.05 −0.05 0.0 −0.05
0.0

0.2 0.1

−0.5
0.45

Time: 67
0.0 0.0 −0.15 0.05 −0.1 0.1 0.05 −0.15 0.1 −0.05 −0.1
0.05 −0.05 0.15 −0.1 −0.2 0.05 0.0 −0.2 0.2 −0.1 0.05 0.0
0.1
0.15 0.1 0.1 0.1 −0.1 0.05 −0.05 −0.05 0.0 −0.15 0.0 0.1
−0.1 0.05 0.05 0.05 −0.25 0.05 0.0 −0.05 −0.05 0.0 −0.05
0.0

0.25 0.05

−0.55
0.45

Time: 68
0.0 0.0 −0.15 0.1 −0.1 0.1 0.05 −0.15 0.1 −0.05 −0.1 0.05
−0.05 0.15 −0.1 −0.2 0.05 0.0 −0.2 0.2 −0.1 0.05 0.0 0.1
0.15 0.1 0.1 0.1 −0.1 0.05 −0.05 −0.05 0.0 −0.15 0.0 0.1
−0.1 0.05 0.05 0.05 −0.25 0.05 0.0 −0.05 −0.05 0.0 −0.05
0.0

0.25 0.05

−0.5
0.45

Time: 69
0.0 0.0 −0.15 0.1 −0.1 0.1 0.05 −0.15 0.1 −0.05 −0.1 0.05
−0.05 0.15 −0.1 −0.2 0.05 0.0 −0.2 0.2 −0.1 0.05 0.0 0.1
0.2 0.1 0.1 0.1 −0.1 0.05 −0.05 −0.05 0.0 −0.15 0.0 0.1
−0.1 0.05 0.05 0.05 −0.25 0.05 0.0 −0.05 −0.05 0.0 −0.05
0.0

0.2 0.1

−0.5
0.45

Time: 70
0.05 −0.05 −0.2 0.1 −0.1 0.1 0.05 −0.15 0.05 −0.05 −0.1
0.05 −0.05 0.15 −0.1 −0.2 0.05 0.0 −0.2 0.2 −0.1 0.05 0.0
0.1
0.2 0.1 0.05 0.1 −0.1 0.05 −0.05 −0.05 0.0 −0.15 0.0 0.1
−0.1 0.0 0.05 0.05 −0.25 0.05 0.0 −0.05 −0.05 0.0 −0.05
0.05

0.15 0.05

−0.55
0.45

Time: 71
0.05 −0.05 −0.2 0.1 −0.1 0.1 0.05 −0.15 0.05 −0.05 −0.1
0.1 −0.05 0.15 −0.1 −0.2 0.05 0.0 −0.2 0.2 −0.1 0.05 0.0
0.1
0.2 0.1 0.05 0.1 −0.1 0.05 −0.05 −0.05 0.0 −0.15 0.0 0.1
−0.1 0.0 0.05 0.05 −0.25 0.05 −0.05 −0.05 −0.05 0.0 −0.05
0.05

0.15 0.05

−0.5
0.45

Time: 72
0.05 −0.05 −0.25 0.1 −0.1 0.1 0.05 −0.15 0.05 −0.05 −0.1
0.1 −0.1 0.15 −0.1 −0.2 0.05 0.0 −0.2 0.25 −0.1 0.05 0.0
0.1
0.25 0.1 0.0 0.1 −0.1 0.05 −0.05 −0.05 0.0 −0.1 0.0 0.05
−0.1 0.0 0.05 0.05 −0.25 0.1 −0.05 0.0 −0.05 −0.05 −0.05
0.05

0.1 −0.1

−0.45
0.45

Time: 73
0.05 −0.05 −0.25 0.1 −0.1 0.1 0.05 −0.15 0.05 −0.05 −0.1
0.1 −0.1 0.15 −0.1 −0.2 0.05 0.0 −0.2 0.25 −0.05 0.05 0.0
0.1
0.25 0.1 0.0 0.1 −0.1 0.05 −0.05 −0.05 0.0 −0.1 0.0 0.05
−0.1 0.0 0.05 0.05 −0.25 0.1 −0.05 0.0 −0.05 −0.05 −0.05
0.0

0.15 −0.05

−0.5
0.4

Time: 74
0.05 −0.05 −0.25 0.1 −0.1 0.1 0.05 −0.15 0.05 −0.05 −0.1
0.1 −0.1 0.15 −0.1 −0.2 0.05 0.0 −0.2 0.25 −0.05 0.05 0.0
0.1
0.25 0.1 0.0 0.1 −0.1 0.05 −0.05 −0.05 0.0 −0.1 0.0 0.1
−0.1 0.0 0.05 0.05 −0.25 0.1 −0.05 0.0 −0.05 −0.05 −0.05
0.0

0.15 −0.05

−0.5

0.3

Time: 75
0.05 −0.05 −0.25 0.1 −0.1 0.15 0.05 −0.15 0.05 −0.05 −0.1
0.1 −0.1 0.15 −0.1 −0.2 0.05 0.0 −0.2 0.25 −0.05 0.05 0.0
0.1
0.25 0.1 0.0 0.1 −0.1 0.1 −0.05 −0.05 0.0 −0.1 0.0 0.1
−0.1 0.0 0.05 0.05 −0.25 0.1 −0.05 0.0 −0.05 −0.05 −0.05
0.0

0.2 −0.1

−0.55
0.35

Time: 76
0.05 −0.05 −0.25 0.1 −0.1 0.15 0.05 −0.15 0.05 −0.05 −0.1
0.1 −0.1 0.15 −0.1 −0.2 0.05 0.0 −0.2 0.25 −0.05 0.05 0.0
0.1
0.25 0.1 0.0 0.1 −0.1 0.1 −0.05 −0.05 0.0 0.0 −0.1 0.0 0.1
−0.1 0.0 0.05 0.05 −0.25 0.1 −0.05 0.0 −0.05 −0.05 −0.05
0.0

0.25 0.1

−0.55
0.3

Time: 77
0.05 −0.05 −0.25 0.1 −0.1 0.15 0.05 −0.15 0.05 −0.05 −0.1
0.1 −0.1 0.15 −0.1 −0.2 0.05 0.0 −0.15 0.25 −0.05 0.05
0.0 0.1
0.25 0.1 0.0 0.1 −0.1 0.1 −0.05 0.0 0.0 −0.1 0.0 0.1
−0.1 0.0 0.05 0.1 −0.25 0.1 −0.05 0.0 0.0 −0.05 −0.05 0.0

0.3 0.05

−0.7
0.35

Time: 78
0.05 −0.05 −0.25 0.1 −0.1 0.15 0.05 −0.15 0.05 −0.05 −0.1
0.1 −0.1 0.15 −0.1 −0.2 0.05 0.0 −0.15 0.25 −0.05 0.05
0.0 0.1
0.25 0.1 0.0 0.1 −0.1 0.1 −0.05 0.0 0.0 −0.1 0.0 0.1
−0.1 0.0 0.05 0.1 −0.25 0.1 −0.05 0.0 0.0 −0.05 −0.05 0.0

0.2 0.0

−0.65
0.4

Time: 79
0.05 −0.05 −0.25 0.1 −0.1 0.15 0.05 −0.15 0.05 −0.05 −0.1

0.1 −0.1 0.15 −0.1 −0.2 0.05 0.0 −0.2 0.25 −0.05 0.05 0.0
0.1
0.25 0.1 0.0 0.1 −0.1 0.1 −0.05 0.0 0.0 −0.1 0.0 0.1
−0.1 0.0 0.05 0.1 −0.25 0.1 −0.05 0.0 0.0 −0.1 0.05 0.0

0.35 0.05

−0.55
0.4

Time: 80
0.05 −0.05 −0.2 0.1 −0.1 0.15 0.05 −0.15 0.05 −0.05 −0.1
0.1 −0.1 0.15 −0.1 −0.2 0.05 0.0 −0.2 0.25 −0.05 0.05 0.0
0.1
0.25 0.1 0.0 0.1 −0.1 0.1 −0.05 0.0 0.0 −0.1 0.0 0.1
−0.1 0.0 0.05 0.1 −0.25 0.1 −0.05 0.0 0.1 −0.1 −0.05 0.0

0.3 0.0

−0.55
0.45

Time: 81
0.05 −0.05 −0.2 0.15 −0.1 0.15 0.05 −0.15 0.05 −0.05 −0.05
0.05 −0.1 0.15 −0.15 −0.2 0.05 0.0 −0.2 0.25 −0.05 0.05
0.0 0.1
0.2 0.1 0.05 0.1 −0.1 0.1 −0.05 0.0 0.0 −0.1 0.05 0.1
−0.1 0.0 0.05 0.1 −0.25 0.15 −0.05 0.0 0.1 −0.1 −0.05
−0.05

0.25 −0.15

−0.55
0.6

Time: 82
0.05 −0.05 −0.2 0.15 −0.1 0.1 0.05 −0.15 0.05 −0.05 −0.05
0.05 −0.1 0.2 −0.15 −0.2 0.05 0.0 −0.2 0.25 −0.05 0.05
0.0 0.1
0.2 0.1 0.05 0.1 −0.1 0.1 −0.05 0.0 0.0 −0.1 0.05 0.1
−0.1 0.0 0.05 0.1 −0.25 0.15 −0.05 0.0 0.1 −0.1 −0.05
−0.05

0.2 −0.3

−0.5
0.6

Time: 83
0.05 −0.05 −0.2 0.15 −0.1 0.1 0.05 −0.1 0.05 −0.05 −0.05
0.05 −0.1 0.2 −0.15 −0.2 0.05 0.0 −0.15 0.25 −0.05 0.05
0.0 0.1
0.2 0.1 0.05 0.1 −0.1 0.1 −0.05 0.0 0.0 −0.1 0.05 0.1
−0.1 0.0 0.05 0.1 −0.25 0.15 −0.05 0.0 0.1 −0.1 −0.05 0.0

0.2 −0.35

−0.4
0.6

Time: 84
0.05 −0.05 −0.2 0.15 −0.1 0.1 0.05 −0.1 0.05 −0.05 −0.05
0.05 −0.1 0.2 −0.15 −0.2 0.05 0.0 −0.15 0.25 −0.05 0.05
0.0 0.1
0.2 0.1 0.05 0.1 −0.1 0.1 −0.05 0.0 0.0 −0.1 0.05 0.1
−0.1 0.0 0.05 0.1 −0.25 0.15 −0.05 0.0 0.1 −0.1 −0.05 0.0

0.2 −0.3

−0.4
0.6

Time: 85
0.05 −0.05 −0.2 0.15 −0.15 0.1 0.05 −0.1 0.05 −0.05 −0.05
0.05 −0.1 0.2 −0.15 −0.2 0.05 0.0 −0.15 0.25 −0.05 0.05
0.0 0.1
0.2 0.1 0.05 0.1 −0.1 0.1 −0.05 0.0 0.0 −0.1 0.05 0.1
−0.1 0.05 0.05 0.1 −0.25 0.15 −0.05 0.05 0.1 −0.1 −0.05
0.0

0.2 −0.3

−0.35
0.65

Time: 86
0.05 −0.05 −0.2 0.15 −0.15 0.1 0.05 −0.1 0.05 −0.05 −0.05
0.05 −0.1 0.2 −0.15 −0.2 0.05 0.0 −0.15 0.25 −0.05 0.05
0.0 0.1
0.2 0.1 0.05 0.1 −0.1 0.1 −0.05 0.0 0.0 −0.1 0.05 0.1
−0.1 0.05 0.05 0.15 −0.25 0.15 −0.05 0.05 0.1 −0.1 −0.05
0.0

0.2 −0.35

−0.35
0.6

Time: 87
0.05 −0.05 −0.2 0.15 −0.15 0.1 0.05 −0.1 0.05 −0.05 −0.05
0.05 −0.1 0.2 −0.15 −0.2 0.05 0.0 −0.15 0.25 −0.1 0.05
0.0 0.1
0.2 0.1 0.05 0.1 −0.15 0.1 −0.05 0.0 0.05 −0.1 0.05 0.15
−0.1 0.05 0.05 0.15 −0.25 0.15 −0.05 0.05 0.1 −0.1 −0.05
0.0

0.1 −0.3

−0.35
0.6

Time: 88
0.05 −0.05 −0.2 0.15 −0.15 0.1 0.05 −0.1 0.05 −0.05 −0.05
0.05 −0.1 0.2 −0.15 −0.2 0.05 0.0 −0.15 0.25 −0.1 0.05
0.0 0.1
0.2 0.1 0.05 0.1 −0.15 0.1 −0.05 0.0 0.05 −0.1 0.05 0.15
−0.1 0.05 0.05 0.15 −0.25 0.15 −0.05 0.05 0.1 −0.1 −0.1
0.0

0.05 −0.35

−0.35
0.65

Time: 89
0.05 −0.05 −0.2 0.15 −0.15 0.1 0.05 −0.1 0.05 −0.05 −0.05
0.1 −0.1 0.2 −0.15 −0.25 0.05 0.0 −0.15 0.25 −0.1 0.05
0.0 0.1
0.2 0.1 0.05 0.1 −0.15 0.1 −0.1 0.0 0.05 −0.1 0.05 0.15
−0.1 0.05 0.05 0.15 −0.25 0.15 −0.05 0.05 0.1 −0.1 −0.1
0.0

0.15 −0.5

−0.35
0.65

Time: 90
0.15 −0.05 −0.2 0.15 −0.15 0.1 0.05 −0.1 0.05 −0.05 −0.05
0.1 −0.1 0.2 −0.15 −0.25 0.05 0.0 −0.15 0.25 −0.1 0.05
0.0 0.1
0.2 0.1 0.05 0.1 −0.1 0.05 −0.1 0.0 0.05 −0.1 0.05 0.15
−0.1 0.05 0.05 0.1 −0.3 0.15 −0.05 0.05 0.1 −0.1 −0.05
0.0

0.2 −0.6

−0.25
0.7

Time: 91
0.15 0.0 −0.2 0.15 −0.15 0.1 0.05 −0.1 0.05 −0.05 −0.05
0.1 −0.1 0.25 −0.15 −0.25 0.0 0.0 −0.15 0.25 −0.1 0.05
0.0 0.1
0.2 0.1 0.05 0.1 −0.1 0.05 −0.1 0.0 0.05 −0.1 0.05 0.15
−0.05 0.05 0.05 0.1 −0.3 0.15 −0.05 0.05 0.1 −0.1 −0.05
0.0

0.15 −0.65

−0.25
0.75

−0.1 0.05 0.05 0.05 −0.25 0.05 0.0 −0.05 −0.05 0.0 −0.05
0.0

0.3 0.05

−0.55
0.45

Time: 92
0.15 −0.05 −0.2 0.15 −0.15 0.1 0.05 −0.1 0.05 −0.05 −0.05
0.1 −0.1 0.25 −0.15 −0.3 0.0 0.0 −0.15 0.25 −0.15 0.0 0.0
0.1
0.2 0.1 0.05 0.1 −0.1 0.05 −0.1 0.0 0.05 −0.1 0.05 0.15
−0.05 0.05 0.05 0.1 −0.3 0.15 −0.05 0.05 0.1 −0.1 −0.05
0.0

0.0 −0.65

−0.25
0.85

Time: 93
0.15 −0.05 −0.2 0.15 −0.15 0.1 0.05 −0.1 0.1 −0.05 −0.05
0.1 −0.1 0.25 −0.1 −0.3 0.0 0.0 −0.15 0.25 −0.15 0.0 0.0
0.1
0.2 0.1 0.05 0.1 −0.1 0.05 −0.1 0.0 0.0 −0.1 0.05 0.15
−0.05 0.05 0.05 0.1 −0.3 0.15 −0.05 0.0 0.1 −0.15 −0.05
0.0

−0.05 −0.65

−0.15
0.7

Time: 94
0.15 −0.05 −0.2 0.15 −0.15 0.1 0.05 −0.1 0.1 −0.05 −0.05
0.1 −0.1 0.25 −0.1 −0.3 0.0 0.0 −0.15 0.3 −0.15 0.0 0.0
0.1
0.2 0.1 0.05 0.1 −0.1 0.05 −0.1 0.0 0.0 −0.15 0.05 0.15
−0.05 0.05 0.05 0.2 −0.3 0.15 −0.05 0.0 0.1 −0.15 −0.05
0.0

−0.05 −0.7

−0.1
0.8

Time: 95
0.15 −0.05 −0.2 0.15 −0.15 0.1 0.05 −0.1 0.1 −0.05 −0.05
0.1 −0.1 0.2 −0.1 −0.3 0.0 0.0 −0.15 0.35 −0.15 0.0 0.0
0.1
0.2 0.1 0.05 0.1 −0.1 0.05 −0.1 0.0 0.0 −0.15 0.05 0.15
−0.05 0.05 0.05 0.2 −0.3 0.15 −0.05 0.0 0.1 −0.15 −0.05
0.0

−0.15 −0.55

−0.05
0.75

Time: 96

```
0.15  −0.05  −0.2  0.15  −0.2  0.1  0.0  −0.1  0.1  −0.05  −0.05
0.1  −0.1  0.15  −0.1  −0.3  0.0  0.0  −0.15  0.35  −0.15  0.0  0.0
0.1
0.2  0.1  0.05  0.1  −0.1  0.05  −0.1  0.05  0.0  −0.15  0.05  0.15
−0.05  0.05  0.05  0.2  −0.3  0.15  −0.05  0.0  0.1  −0.15  −0.05
0.0

−0.15  −0.55

−0.05
0.75

Time: 97
0.15  −0.05  −0.2  0.15  −0.2  0.1  0.0  −0.1  0.1  −0.05  −0.05
0.1  −0.1  0.15  −0.1  −0.25  0.0  −0.05  −0.15  0.35  −0.15  0.0
0.0  0.1
0.2  0.1  0.0  0.1  −0.1  0.05  −0.1  0.05  0.0  −0.15  0.05  0.15
−0.05  0.05  0.05  0.2  −0.3  0.15  −0.05  −0.05  0.1  −0.15  −0.05
0.0

−0.2  −0.65

−0.1
0.8

Time: 98
0.15  −0.05  −0.2  0.15  −0.2  0.1  0.0  −0.1  0.1  −0.05  −0.05
0.1  −0.1  0.15  −0.1  −0.25  0.0  −0.05  −0.15  0.35  −0.15  0.0
0.0  0.1
0.2  0.1  −0.05  0.05  −0.1  0.05  −0.1  0.05  0.0  −0.15  0.1  0.15
−0.05  0.05  0.05  0.2  −0.3  0.15  −0.05  −0.05  0.1  −0.15  −0.05
0.0

−0.2  −0.65

−0.05
0.9

Time: 99
0.15  −0.05  −0.2  0.15  −0.2  0.1  0.0  −0.1  0.1  −0.05  −0.05
0.1  −0.1  0.15  −0.1  −0.25  0.0  −0.05  −0.15  0.35  −0.15  0.0
0.0  0.15
0.15  0.15  −0.05  0.05  −0.1  0.05  −0.15  0.0  0.0  −0.15  0.1
0.15  −0.05  0.05  0.05  0.2  −0.3  0.15  −0.05  −0.05  0.1  −0.15
−0.05  0.0

−0.25  −0.65

0.0
0.95
```