DESIGNING AND DEVELOPING AN INTELLIGENT CONGKAK

A project report submitted to the
Faculty of Information Communication Technology
in partial fulfilment of the requirement for the degree
Master of Science (Information Communication Technology)
Universiti Utara Malaysia

By
Muhammad Safwan Bin Mohd Shahidan (s805747)

PERMISSION TO USE

In presenting this project report in partial fulfillment of the requirements for a postgraduate degree from Universiti Utara Malaysia, I agree that the University Library may make it freely available for inspection. I further agree that permission for copying of this project report in any manner, in whole or in part, for scholarly purpose may be granted by my supervisor(s) or, in their absence by the Dean of the Graduate School. It is understood that any copying or publication or use of this project report or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to Universiti Utara Malaysia for any scholarly use which may be made of any material from my project report.

Requests for permission to copy or to make other use of materials in this project report, in whole or in part, should be addressed to

Dean of Graduate School
Universiti Utara Malaysia
06010 UUM Sintok
Kedah Darul Aman.

ABSTRAK

Congkak ialah permainan tradisional Malaysia dan ia terdedah kepada risiko untuk dilupakan jika kewujudannya tidak diambil serius, tetapi setakat kini tinjuan teks-teks rujukan tidak menjumpai sebarang publikasi yang menyatakan penggunan algoritma neural-network (NN) ke atas permainan Congkak. Oleh itu projek ini ingin untuk menangani isu ini dengan membina satu sistem Congkak dengan NN dan juga cuba untuk menjawab persoalan-persoalan formal berikut: “Apakah fungsi penilaian Congkak yang sesuai untuk melatih NN bagi permainan Congkak?” (contohnya: adakah penilaian Congkak melalui mengiraan buah terkumpul lebih baik dari penilaian melalui pengiraan 'rumah-terbakar'?) dan “Bolehkah prestasi bagi algoritma Min-Max & Alpha-Beta cut-off (MM) ditingkatkan jika NN digunakan sebagai sejenis teknik 'forward-pruning' untuk MM?”. Permasalahan ini diselesaikan dengan membina satu sistem Congkak berdasarkan kerja-kerja terdahulu yang berkaitan dengan sistem Mancala dan sistem NN, dan kemudiannya merekod prestasi algoritma yang terlibat untuk membuat kesimpulan. Hasilnya: projek ini berjaya mencipta satu sistem Congkak dengan 3 jenis ajen kecerdasan buatan (AI agent), dan mendapati bahawa gabungan NN dan MM adalah lebih perlahan daripada MM semata-mata.

ABSTRACT

Congkak is the nation's traditional game which could soon be forgotten if no serious attention is given to it, but literature survey has not yet found any research publication that mentioned the use of neural network algorithm (NN) on Congkak. Therefore the project want to try to rectify this issue by trying to develop an Intelligent Congkak System that also implemented NN and try answer research question such as this: “What is the best Congkak evaluation function for training NN for game playing?” and “Can Min-Max algorithm (MM) be speeded up by using NN as a forward-pruning method?”. This issues can solved by programming the Congkak system based on previous work on Mancala and NN system, and then recording the performance of the related algorithm. As a result: the project had created a Congkak system that had featured 3 Artificial Intelligence (AI) agent, and discovered that the combination of NN and MM is slower than MM alone.

ACKNOWLEDGEMENT

Firstly I would like to thank all the people in the world that have willingly shared their knowledge freely on the internet. Without them the internet would have been empty and the project would fail. Thank you for sharing.

I would also like to thank Madam Latifah binti Abdullah for sharing with me the guideline & protocol for project writing and also allowed me to extend my project report submission date, and I would also like to thank both my project supervisor Miss Noraziah binti Che Pa and Miss Aniza binti Mohamed Din for giving me the encouragement & helpful comment and for checking and suggesting to me the correction for my project, and Dr. Yuhanis for evaluating my project, and also a friend Fadi Shaar Abdulghani who had helped me by sharing the guideline of the project proposal. Without them the project would have failed. Thank you for all the help.

I would also like to thank Ministry of Higher Education (MOHE) for funding my study (under Mini Budget 2009), and also both my parent for giving me the financial support, and also my cousin Rizal Harun for helping me with everything. Life would be difficult if without them. Thank you.

TABLE OF CONTENT

LIST OF TABLE

LIST OF FIGURE

Chapter 1

1. INTRODUCTION

This project is focussed on implementing Artificial Intelligence (AI) technique in Congkak game playing. An AI agent was created as a player that could be configured to play with a human or with itself. The agent used Neural Network algorithm (NN), Min-Max algorithm with Alpha-Beta function (MM), and Random-moves-generator to play the game.

AI is an exciting field of research. The goal of AI field is to develop a system that can solve real world problems: such as Chess game, predicting stock market and facial recognition. Most recent and exciting development is an AI agent named Watson developed by IBM; it can answer question posed in natural language and has won a game in an American quiz show called "Jeopardy" (IBM (2011)). Another AI field is visual recognition; which also has become ubiquitous nowadays in form of facial recognition software installed on our laptop, and other exciting development is in computer gaming; where an AI agent named Milo can recognize player's emotion and interact with the player (Gibson. E (2009)).

The project will use AI on a small scale. Several AI technique was used on the game Congkak; Congkak has simpler rule and simpler mechanics than

the real world problem so it is an ideal place to test any AI algorithm. The AI will acted as an artificial player that will try to defeat its opponent by moving pebbles or marbles into specific hole.

1.1. Problem Statement

Based on current literature survey there doesn't appear to be a Congkak system mentioned that implemented NN. However there was a Congkak system mentioned by Alifia et. al. (2006) that uses greedy search algorithm (similar to MM) to find for optimal move, and there are also a Dakon system mentioned by Donker, J. et. al. (2000) which has almost similar rule to Congkak. However, all known intelligent system appears to be focussed on non-Congkak variant of Mancala game (Bylander, T. (2008); Wee-Chong,O. et. al. (2003); H. Jaap van den Herik et. al. (2001); Donkers, J. et. al. (2000); Cofer, A. (2003); Irving,G. et.al. (2000); Davis, J.E. et. al. (2002); Pickhard, A. (2007); Ahlschwede, J. (2000); Kronenburg,T. (2008); Alifia et. al. (2006); Mohammed Daoud et. al. (nd); Romein, J. W. et. al. (2003)) . Congkak is a local Malaysian traditional game but the literature survey only found few research paper that mentioned AI research on Congkak system (and none was from Malaysia).

Awari, Kalah and Dakon is a type of Mancala game and it is the most popular Mancala game among computer scientist (H. Jaap van den Herik et. al. (2001)). Awari, Kalah and Dakon is played by a larger community (thus more popular) and its solution is not trivial; which justify the use of AI method on

these game. Dakon is the most similar game to Congkak but Dakon doesn't represent a Congkak game because there was no simultaneous start rule like in Congkak.

Davis, J. E. et. al. (2002) and Wee-Chong,O. et. al. (2003) has both demonstrated the use of NN on Mancala game specifically in Awari and Kalah respectively. Both researchers has produced a very good master level NN agent. This shows that NN can be used as an agent to play games similar to Congkak.

MM is the most common AI algorithm for games and it is known to have been applied on Mancala (Cofer, A.(2003); Bylander, T. (2008)), but MM alone was long recognized to be a slow algorithm (greedy algorithm). If the state-space is too big it will takes hours to completely explore it (depending on CPU speed and cut-off depth), but an addition of forward-pruning algorithm (such as Alpha-Beta algorithm) has significantly improve its performance (Lim,Y. J. (2007)). The project's aim is to try to use NN as an additional forward-pruning method and see if it has a positive effect on MM's speed performance.

1.2. Research Question

The research question are:

- Can MM be speeded up if using NN as a forward-pruning method?
- What is the best evaluation function to train NN for Congkak game

playing?

In the effort to answer those questions, a complete Congkak system must be constructed. The development of a Congkak system will provide the platform for testing the algorithms that could answer the research questions.

1.3. Goal and Objective of the Project

The goal of the project is to develop a system that can answer the research questions and can resolves the issues mentioned in problem statements. The deliverable for the project will be a system that could be useful for future research and will add to the body of knowledge regarding the "know-how" for developing future Congkak system and its AI agent. Therefore he project's objectives are:

- to develop an AI agent based on NN and MM that which can be tested to play a Congkak game.
- to develop a Congkak system that can simulate a real Congkak game and which can collect data about the performance of the AI agents.
- to develop a Congkak system that can interact with users and allow users to play Congkak virtually.
- to make a conclusion that can answer the research question.

1.4. The Significance of the Project

This project can be useful to other future project because it contain a

working example of a Congkak intelligent system implementation. This project can demonstrate-

- a working example of how a NN was used as an agent, and;
- a working example of how Congkak rule was simulated,
- a working example of how the interface was done, and;
- a working example of how data collection was made.

With this problem solved, other more complex system can be built on top of the project.

1.5. Scope, Assumption and Limitation of the Project

1.5.1. Scope

The scope of the programming is limited to within the subject touched by an introductory Java course, however some concept may exceed that scope: such as multi-threading, dynamic array, recursive programming, and NN. But it was implemented by the help of external package such as Neuroph, and from other people's work on Mancala intelligence system, and from NetBean IDE's auto-correct function, and some are from online search and from Javadoc. The programming scope of the project is within the introductory courses and the AI courses, but also uses some external resources (which could add to the size of scope).

1.5.2. Assumption Made Prior to Project Execution

The project assumes Congkak intelligence system is a unique project, and no resource (research paper or open-source code) of whatsoever was publicly available for such system of similar title or of same goal, and is therefore worth pursuing based on this uniqueness. It is assumed that a creation of Congkak system is desired because it will preserve national identity (Noraziah Che Pa, master proposal discussion, March 13, 2011).

1.5.3. Limitation

Due to time limitation: the project was not able to test multiple configuration of the applied algorithm (the code is not guaranteed as the most efficient alternatives) and is not aiming to find a better alternative to the current applied algorithm (the project could test other NN package but not done so. Eg: Encog is reportedly faster than Neuroph). Some algorithm, ideas and NN configuration were derived from other people's work; such as the concept of NN from GNU-Backgammon (Gerald Tesauro's system) and MM from Mancala intelligent system (Adam Cofer's system) (Cofer, A. (2003); Nyugen, D. et. al. (1989), Goldberg, C. (2005); JuMpErFLY (April 24th 2004); Deitel, P. J. et. al. (2006); Eck, D. J (2006); "5up3rJ" (aka. SuperJ) (Jan 4th, 2006); Tesauro, G. et. al. (1989); Tesauro, G. (1992); Tesauro, G. (1995); Tesauro, G. (2002)), but some algorithm must be designed within the project because no resource is available; eg: Congkak algorithm. Time was allocated mostly on training/test NN and other sub-system (for few hours and is

repeated if correction need to be done), then devising a working algorithm, and correcting logic error , which then leave less time for risky exploration of new alternative algorithm.

The Congkak system has expanded its initial requirement from just a system for testing NN into a system that also feature a functional Graphical-User-Interface (GUI) for user interaction. The development on GUI had reduced the testing done on the NN subsystem. Therefore the data on NN agent was not complete and the NN agent did not perform to high expectation due to lack of tuning, and more time is needed if to improve the performance for NN agent.

The implementation of the GUI is not the most visually appealing, but it is sufficiently functional. More advanced process (such as: smooth animation) is possible but require more literature review and development time. The scope of the project is limited only to AI field (eg: is not involved with Human Interaction Design) .

1.6. Definition of Terms

In this section the term “Mancala” and “MM” was mentioned several times. Mancala is a traditional game originating from Africa that has similar rule and board shape to Congkak but is much simpler than Congkak, and MM is an algorithm that uses state searching to find the best possible move. Many board games (such as Mancala) uses MM as an artificial agent and act as a benchmark to other artificial agent because of its completeness in finding

solution.

“Java” is a programming language with similar syntax to C++ but with simpler object model and fewer low-level programming function. Java has an automatic memory management and doesn't need and doesn't allow pointers and other direct memory manipulation to to be used, thus it is simpler than C++.

1.7. Organization of Report

This report is divided into 6 chapter: Chapter 1 is the introduction, Chapter 2 is Literature Review, Chapter 3 is project methodology, Chapter 4 is the result, Chapter 5 is the discussion, and finally Chapter 6 is project conclusion and recommendation. Chapter 2 discusses the ideas and knowledge obtained from literature review. Chapter 3 discusses how the system is developed and discusses its implementation. Chapter 4 presents the data (collected from the system). Chapter 5 discusses any system bug and weaknesses revealed by the data. Chapter 6 discusses conclusion on the project and recommendation for future work.

Chapter 2

2. LITERATURE REVIEW

2.1. Introduction

This project is conceptually based on Gerald Tesauro's paper (Tesauro, G. (1989); Tesauro, G. (1992); Tesauro, G. (1995); Tesauro, G. (2002)) on NN Backgammon player, and also programmatically based on Mancala intelligence system written by Cofer, A. (2003). Gerald Tesauro developed the backgammon's NN system under IBM company research and the software was released publicly under GNU licence as “GNU-Backgammon” and was available online as C++ source-code. While the Mancala system was developed by Adam Cofer for his Master Thesis and it utilized MM (Min-Max with Alpha-Beta cut-off function) and also released under GNU licence and available online as Java source-code in the Appendix of his Thesis paper. This Congkak project is based on Adam Cofer's work since his project is more closely related to Congkak.

2.2. Brief Review on Artificial Intelligence Literature

Gerald Tesauro implement a NN is called TD-Lambda. 'TD' is an

abbreviation for “Temporal-difference-learning”, and temporal-difference-learning is a theory of how reinforcement-learning is conducted for NN to allow a system to make a better prediction in time after time (Wikipedia: “Temporal difference learning” (2011), Sutton, R. S. et. al. (2005)) . In TD-lambda: the network was configured to train with a time-series data, and the goal was to minimize the prediction error (such that predictions follows closely to the actual outcome). By default the 'actual outcome' is the winning move from the winning player, and the goal is to make a better predictor for the winning move: which consequently lead to a better player. (Sutton, R. S. et. al. (2005))

In Gerald Tesauro's work (1995) the network reached the level of world champion player after 300,000 games training with self-play. The purpose of training is to allow the network to improve its approximation of the game's rule such that it became a better player. Simple rule is approximated by a linear function, while complex rule can be approximated by non-linear function; NN can discover both and approximate both function through training (StatSoft Inc. (2011)).

For example: logic function such as “number1 --> number2” (number 1 imply number 2) can be approximated by “number2 = constant * number1” (a mathematical function), and logic function “(number1 ^ -number1) --> number2” (number 1 or a negative of number 1: imply number 2) can be approximated by “number2 = number1 * number1” (number 2 is equal to the square of number 1). The former is a linear function and the latter is non-linear

function (specifically: the later is a polynomial function of order 2). NN can approximate both function, but if over-fitting occur: NN will then approximate a simple function (such as a polynomial function of order 1) with an over complicated non-linear function (such as a polynomial function of order 3) (StatSoft Inc. (2011); Lawrence, S. et. al.(1997)).

Big NN (NN with large amount of neurons or perceptron) is able to generalize problem better (Lawrence, S. et. al.(1997)) but it is also susceptible to over-fitting (StatSoft Inc. (2011)).

For Mancala: Cofer, Adam (2003) implement Min-Max with Alpha-Beta forward pruning function (MM). The MM does its work by searching the game-state for a state that contain the largest amount of pebbles in its storehouse, and Alpha-Beta works by excluding game-state that do not lead to such state (thus lessening the size of the state search) (Bylander, T. (2007)). MM in Adam Cofer's work is fully compatible with Congkak and it only require small/minimal change for it to be functional with Congkak.

2.3. Overall Literature Review

The following is a review of all literature and tools relevant to the project. The literature review has revealed many solution to the many development questions of this project, such as: how the project should be conducted and what tool it need to make the system work. Many important concept about neural network system can be learn from Gerald Tesauro's paper (Tesauro, G. (1989); Tesauro, G. (1992); Tesauro, G. (1995); Tesauro, G.

(2002)), and several month of work has been cut out from this project when Adam Cofer's system (Cofer, A. (2003) is used as a template, and the use of Neuroph (Sevarac, Z., et al. (2008)) & the use of Java and Netbean IDE (Oracle, (2011)) had transformed a very complicated/daunting programming problem into a tractable (solvable) problem. The only biggest challenge left was to make sure that the idea is implemented successfully.

2.3.1. Neural Network (NN)

The input for the NN must be as a time-series data. This is based on the article written by Carter-Greaves, L.E. (2009) and Steinhauer,V. (2009) from Neuroph project: a time-series data is a set of data point that can be plotted over time axis. For input: each data point will have similar (temporal) distance between each other and is feed into the NN's input neurons in a serial manner, while the latest data point is also feed into the output neuron.

The NN will be trained in the above said configuration, but when a new data point arrive: the old data point is shifted backward in time and the new data point is feed into the output neuron and similar pattern will continue (Steinhauer, V. (2009)). The goal is to make NN learn the behaviour/pattern of the data point(s) in those time-series. The NN will learn faster if the data points had showed a discernible pattern (Carter-Greaves, L. E. (2009)).

For Congkak system: both the board state and the move state is those data points, and this data points is separated from each other by the player's turn, and the player's turn was interleaved with the opponent's turn to form a

long series of input (which is visually similar to the work by Nyugen, D. et. al (1989)). Only the winning player's turn is feed into the output neurons, but the trailing turn (winner and loser) is all assigned to the input neuron (view Figure 3.3 (page 31)). The training aim to detect a patterns in the trailing input that can lead to a winning state in the output neuron.

In Gerald Tesauro's system (1995): the data point near the endgame is flagged as more significant for NN training than the one further away from the endgame. This is based on the assumption that not all data point was a contributor to the winning state, some may just be a noise. Gerald Tesauro refer to this problem as “temporal credit distribution problem” and was solved by assuming the initial move (for Backgammon) was less significant to the winning outcome than the final move.

In this project the data point is flagged as 'significant' or 'insignificant' by reducing or increasing the NN's “max_training_error”. This mean that the training will be performed with either more iteration to achieve a desired minimum-error or is interrupted early to allow deviations (“StatSoft Inc. (2011)). For Congkak: when a player reclaimed their burnt_pit (gained a lost hole) or caused the opponent to loose hole (induce more burnt_pit), or when a move increased the storehouse count by more than 1 and also prolonged the turn: then those moves was considered 'significant'.

2.3.2. Congkak Game

Congkak game is similar to a Mancala game which originated from

Africa (Voogt, A. de (2001)). Mancala game did not featured a multi-lap move, and no multi-stage game and no burnt_house concept. Mancala player start playing the game by picking up pebbles from the holes on the player's side and then move counter clockwise while deposited a pebble on each hole each time the player passes a hole or storehouse (except the opponent's storehouse). If the player emptied the pebbles from his hand on a empty hole then he will lose the current turn, and if this hole was on his own side then he could capture pebble from the opposite hole and put them (and the last pebble that he had deposited) into his storehouse. However if this empty hole is in his opponent's side then he must left the pebble there. Mancala player can deposit pebbles in any hole except in his opponent storehouse which he must skip. (Cofer, A.(2003))

In Congkak: each time a player emptied the pebbles from his hand on any non-empty hole, then the player must continue moving using the content from that last hole (the player must pick-up the content and continue move) or if the hole was empty: his turn will end just like in Mancala (mentioned in previous paragraph). For new game: both player must start simultaneously and perform the Congkak's move rule until they ended their turn, if they had ended their turn: then they must wait for their opponent to finish first, the player who ended his turn last will go first in the next phase (Yaakub Rashid (1981); *"How To Play Congkak"* (2010)), in next phase: the player will play in turn like normal (simultaneous start only apply to the first game, not on the consequent round). This rules made Congkak different from Mancala. In

Congkak: a single move can be prolonged considerably and the first game will start like a race track.

The endgame is reached whenever a player had all the hole on his side empty. When this happen, all pebbles from the remaining hole will be sent to his opponent's storehouse and both players will count the total pebbles collected in their storehouse. Player with the highest number of pebbles will wins, and player who lose this game will start first in the next round. (*"How To Play Congkak"* (2010))

To start a new round: the player must fill-in their empty holes with pebbles from their storehouse, the players must do this from left to the right hole (holes from their own side), and each hole must be filled with exactly seven pebbles and the remainder can be stored in the storehouse, a non-filled hole is considered a "burnt_hole" and will be ignored during play (this is called burnt_house rule). The burnt_house rule allow the loosing player to reclaim pebbles from their opponent in next round (Yaakub Rashid (1981);*"How To Play Congkak"* (2010)). Player with 1 or 2 burnt_hole seems to have a certain advantage; because random play suggest that it is easier to win the round when you have about 1 or 2 burnt_hole.

The project is built upon the code written by Cofer, A. (2003): it integrated Congkak rule into a pre-existing Mancala system, and has included NN, Random-move and Neural-Min-Max hybrid (NMM) as 3 new artificial intelligence agent in addition to the original MM. Adam Cofer's system is a complete system in which Congkak rules and new AI agent can be applied and

tested fairly quick. Adam Cofer's Mancala system already had a robust framework such as a main class, a working MM class, and working text-based interface class: which allowed new codes to be build on top of existing one and tested quickly.

2.3.3. Min-Max Algorithm with Alpha-Beta function (MM)

MM algorithm is the most effective algorithm for creating an artificial agent for board game. It works by creating a secondary copies of the present game during play and exhaustively test moves on this copies to get the outcome, and when the desired outcome was found: it retrieve the corresponding move (that is responsible for this outcome) and send it to the present game. MM can work on any board game as long as the game's instance is copyable and it can make a perfect copy, and as long as its game-state is easily evaluated by a mere calculation or algorithms.

The MM's evaluation-function for Congkak system is “the amount of pebble contained in the agent's storehouse minus the pebbles contained in the opponent's storehouse” (the difference of stored pebbles). This is similar to MM's evaluation-function for Mancala.

MM is actually similar to greedy-search algorithm, thus it can be imagined in term of search-tree. Cofer, A. (2003) MM algorithm performed a depth-first-search (DFS) with Alpha-Beta cut-off function (as mentioned in the code description).These algorithm starts by opening a child nodes and then evaluate it, and then it open another node and then evaluate it along the way

until it reaches the endgame node (or until it reach a cut-off depth), when it reach the end: it passes the endgame's evaluation value to its parent node, and then the parent will select the highest evaluation value (which was received from many of its child node) to be passed on up to the grandparent node and so on until it reaches the original board state (original node/grand-grandparent node), then the original node will select the highest evaluation value (received from many of its child node) as the next move.

The Alpha-Beta cut-off function works by immediately returning a best move value whenever a specific condition was meet. This skips the entire MM search-space and saves processing time. MM can still return a same value if without Alpha-Beta cut-off: but will require more time to completely search the entire search-space (depending on the cut-off depth). (Samuel, A. L. (1967); Lim,Y. J. (2007).)

2.4. Extra Definition and Term

The term 'plies' or 'ply' means “turn”. Therefore “self-ply” mean “self-turn”. This means that an agent will play with itself. This term is used by Gerald Tesauro's paper to indicate his NN training strategy.

Congkak uses alternative terms like 'village' for storehouse and 'house' for holes or pits. Both terms has similar meaning. Storehouse is preferred because it is more descriptive.

A “training evaluation function” is a function that will determine the value of “max_training_error”. It is mentioned in the research question in

Chapter 1. There are a total of 4 different training evaluation function mentioned in the project.

2.5. Summary

Only relevant literature was reviewed in this chapter. Concepts like over-fitting, time-series configuration, MM, and Congkak rule is essential to understand the following chapters.

Chapter 3

3. PROJECT METHODOLOGY

3.1. Introduction

The project methodology contain 4 phases and 9 activities. Such methodology is loosely based on the book written by Kothari (2004). Table 3.1 (next page) summarize this 4 phases and 9 activities, and the following section will described in more detail on each of the activities in this methodology.

Table 3.1: Project's Methodology. There were 3 phases and 9 activities conducted in period of 3 month.

Phase	Activity	Method	Deliverable
-Project identification & initiation.	-Identify the problems.	-Discussion with supervisor. -Quick literature survey.	-Project's topic.
	-Perform an extensive literature review.	-Review the literature.	-Method & technique for system design.
	-Create a valid research question.	-Analyse the literature.	-Research question.
-System design & development.	-Design system that answers research questions.	-Identify system's requirement. -Perform XP Programming. -Debug the system.	-Congkak prototype.
-Data collection & analysis.	-Identify the data to be collected from the system.	-Identify Data.Collection-sub-system's requirement.	-Data type.
	-Perform the data collection.	-Program Data.Collection-sub-system. -Evaluate/Run prototype	-Raw data.
	-Process the data & analyse the data.	-Sort data. -Analyse data with SPSS.	-Graphs & aggregate values.
	-Interpret the data & answer research question.	-Review the literature. -Answer research question.	-Discussion & Conclusion.
-Project documentation.	-Prepare a report.	-Identify APA guideline. -Get correction from supervisor. -Review Report-writing-guideline.	-Report.

3.2. Project Methodologies

The project methodology consist of 4 phases: (1) Project Identification & Initiation, (2) System Design & Development, (3) Data Collection & Analysis, and finally (4) Project Documentation. Each phase contain several activities and each phase summarize the type of activity performed. For example: phase (1) is involved with activity that determine the goal of the project, and phase (2) is involved with activities that deal with the technical aspect of the project, while phase (3) is involved with activities that deal with data collection, and finally phase (4) is involved with activities that is conducted for report writing.

3.2.1. Project Identification & Initiation

“Project Identification & Initiation” consist of 3 activities: (a) Identify the Problems, (b) Perform and Extensive Literature Review, and finally (c) Create a Valid Research Question. Such activities was aimed to identify the project's goal and to gather any resources that can help with system programming: such as software-tools and algorithms. The output for this phase is the “Research Question”, the “Project's goal”, and the “Method & Technique” that is essential for performing the next phase: “System Design & Development”.

a. Identify the Problems

The project was started with a discussion with the project supervisor: Ms. Aniza and Ms. Noraziah, regarding the issues that can be solved by the project. The project supervisors have more qualification in making suggestion for a new project title because they have more experience in the related research field (AI field). As a result: “Designing and Developing an Intelligent Congkak” is chosen as the project title because such project will have some value to the nation and 'Jabatan Warisan Negara' also had requested a Congkak system to be developed.

Then a quick literature survey is conducted to determine the present level of research that has been done (by other researcher) for a Congkak system. This helps the project to identify useful tool from previous research, and to identify unresolved problem that could be the focus of the project. As a result: the survey discovered a number of AI research already been conducted

for Mancala game (which is a very similar game to Congkak), one for Congkak (Alifia et. al. (2006)), but none had used Congkak and NN, and therefore the implementation of Congkak & NN has became the goal of the project.

b. Perform Literature Review

The second process involved doing an extensive reading on all AI literature and Congkak literature (gathered during the previous activity). These literature contain solutions & techniques that could help to develop the actual Congkak system. For example the literature review has found many resources that had helped the development process; such as a Mancala source-code, a 'blueprint' for developing a NN agent (the concepts/design), and several algorithms and codes that has allowed other function such as data-collection and GUI to be available for the Congkak system.

c. Create a Valid Research Question

The research questions is the questions that naturally arise from a literature reading but wasn't answered by the literature itself. It is akin to a knowledge gap that exist because of an incomplete/not self-contained literature; which demanded another literature or an experiment for an answer. Thus, based on the idea that Congkak has a billions of state space (Donker (2000)) and MM must slowly searches through all those state-space (Newborn (2003)) and NN is an algorithm that can learn/remember any sort of

problem and solution (StatSoft Inc. (2011)), then could the speed of MM be increased by combining it with NN? this question was not answered explicitly by any of the reviewed literature. The literature also didn't reveal what evaluation function will work best for training Congkak's NN. So the 2 question became the project's research question.

3.2.2. System Design & Development

The “System Design & Development” phase contain one continuous long activity: (a) Design a System that Answer the Research Question. This activity consist of 3 basic programming process: (1) translate system-requirement into low-level requirement (this requirement is basically the system's design/specification), (2) write the code, and finally (3) debug the code, which was performed iteratively (repeatedly in small portion) until a complete system is build. The output of this phases is a stable Congkak prototype system (not a throwaway version or a beta version).

a. Design a System that Answers Research Question

The project's goal was to create an intelligent Congkak system; which can be fulfilled by programming the system in Java language. Java language is more intuitive to use than C++ and has many resources written as an open-source software. The user interface can be programmed using Java-Swing, the NN agent can be programmed using Neuroph, and the MM agent and Congkak system can be programmed by modifying Adam Cofer's (2003)

Mancala intelligent system written by (all written in Java).

The design process for the system is started by first visualizing about the desired output (eg: the system design and programming flow. However XP programming did not need such activity to be documented: therefore it was not included here) and then to try to achieve that output using a clever usage of these following 3 programming components: (1) a control-statement (eg: “if-else”), (2) a loop-statement (eg: “while”), and (3) a call-in to several useful packages (eg: “Java Swing” or/and “Neuroph”). This is because basic programming is entirely consist of such components: such as a looping-statement, a control-statement, a mathematical-statement (eg: multiplication and addition), and a call-in to other function/method/classes/package (Deitel (2006)). The Congkak system appears complex because of its code length and flow control but it was entirely based on those simple rules, however other Java specific/special function such as 'variable-localization' (eg: the “private”, “public” and “static” syntax) and the function 'override' function (eg: “extend”, “override” syntax) is also implemented for increasing the code's efficiency (eg: to allow complex behaviour to manifest without using a great code length) and was usually inserted automatically by NetBean IDE (eg: during Java GUI implementation) and was also present in the original Mancala source-code (eg: the “static”).

The design and development phases did not follow a rigid “waterfall model” but follow a “XP programming” style (Dennis (2010)). The coding is started as soon as the system-requirement/design is received (eg: in other

project the stakeholder usually work alongside programmer to reiterate/ communicate the system requirement to the programmer and the programmer will usually try to programs the design) and there was no lengthy system documentation needed; the only documentation was the source-code. This allow a small systems to be developed faster than anything that is possible with other software development cycle (SDLC) (Dennis (2010)). The following sub-section describe the design of a Congkak NN sub-system, but other sub-system is left out because of time constraint:

i. Neural Network (NN) Sub-system: Intro

This section focusses on NN sub-system. Other sub-system also exist, but is skipped: such as Graphical-User-Interface (GUI), text-based user interface, logging system, MM agent, Random-Move agent, and NMM hybrid agent. The NN system cover 5 Java class: CongkakNNMove.java, CongkakServer.java, CongkakDataKnitter.java, CongkakBoard.java, and CongkakTrainer.java, and uses Neuroph as an external library.

Neuroph is the Java package that contain NN method. The project uses Neuroph for creating and training NN, and it is treated as a blackbox. The project didn't deal with the complexity of NN formula & coding, and therefore is not in control of the NN's performance.

ii. Neural Network (NN) Sub-system: Classes

The CongkakServer.java class is where the main method is located.

Another main method is in CongkakGUI.java (which display Graphical User Interface), but CongkakServer.java is where the NN is trained because it is much simpler to reprogram the CongkakServer.java than to reprogram the GUI. CongkakServer.java is a modified version of the original MancalaServer.java written by Adam Cofer.

CongkakDataKnitter.java is where the data transformation and storing was done. It store the game history for later training, and it also transform present board state into a compatible neural network input. Basically it normalized the pebble count into the range of 0 to 1, and then stored all these values into a very long array for later training purposes.

CongkakBoard,java class is where the Congkak game was simulated. It is a modification of MancalaBoard.java class written by Adam Cofer. The CongkakBoard.java is different from MancalaBoard.java because it contain algorithm for simultaneous start, multi-lap move, and burnt_house rule.

CongkakNNMove class is where the neural network make its move. It contain an algorithm which selects move from the NNs output. NN doesn't always make a valid move; so an algorithm in this class will select the valid move from the one offered by the network.

CongkakTrainer class is where NN is trained. It uses Application Programming Interface (API) from Neuroph to create a training element and to set learning rules. It is also responsible for creating new NN, and the training-evaluation-function for NN training is configured here.

iii. Neural Network (NN) Sub-system: Design Intro

The rest of this section discusses the concept used to develop the Congkak system (but focussed primarily on NN sub-system). These concept is important because it explains why the system behave the way it is as displayed in the results (Chapter 4). For example: the speed of the Neural-Network training and the effectiveness of the training depended largely on the choice of the network's size and the way the Congkak board is represented to the network (there are many ways to represent the Congkak board, and the network size can be set arbitrarily).

iv. Neural Network (NN) Sub-system: Design

The first design and development phase started with a broad literature review on all subject related to Mancala intelligent system, which then ended with a review on Gerald Tesauro's paper and Adam Cofer's Mancala system. The literature review suggest that building a NN agent is not a complicated problem and is possible. The next step was to understand how Neuroph work and to test an initial NN system on Adam Cofer's Mancala system, the whole process took several days.

For Mancala system: the NN agent works. The agent can beat MM (of depth cut-off value of 4) and Random-Move all the time. One of the important idea tested at this time was the board representation; the original idea was to represent the entire Mancala board as a binary representation but it require 1000 neural input and the system throws stackOverflow error (indicating that

Java Virtual Machine's (JVM) working memory is full), the second idea was to normalize the pebble count into range 1-to-0 and this reduced the number of input neuron to 14 (for network 1), and 39 (for network 2), and 64 (for network 3). The project uses 3 NN in parallel for the reason to be discussed next.

The Mancala board was represented using an array, this is located in MancalaBoard.java. The array store the amount of pebbles for each hole followed by a series of zeros-or-one that represent the turn's move. For example: the initial board state is represented as 4,4,4,4,4,4, 0, 4,4,4,4,4,4, 0, (end pebble amount) 0,0,0,0,0,1, 0,0,0,0,0,0 (end move state indicator. With move at hole 5), this is an array of size 14, plus 12 for move state .

The Congkak board is similar to Mancala board but with more information, this is located in CongkakBoard.java. It included the burnt_pit and the amount of pebbles on the player's hand. For example: the initial board state is represented as 7,0,0,7,0,0,7,0,0,7,0,0,7,0,0,7,0,0,7,0,0, 0,0,0, 7,0,0,7,0,0,7,0,0,7,0,0,7,0,0,7,0,0,7,0,0, 0,0,0, (end pebble amount) 0,0,0,0,0,0,1, 0,0,0,0,0,0,0, (end move state. With move at hole 6), this is represented as an array of size 49, plus 14 for move state. Figure 3.1 (next page) shows the board representation, Figure 3.2 (next page) shows the move state representation.

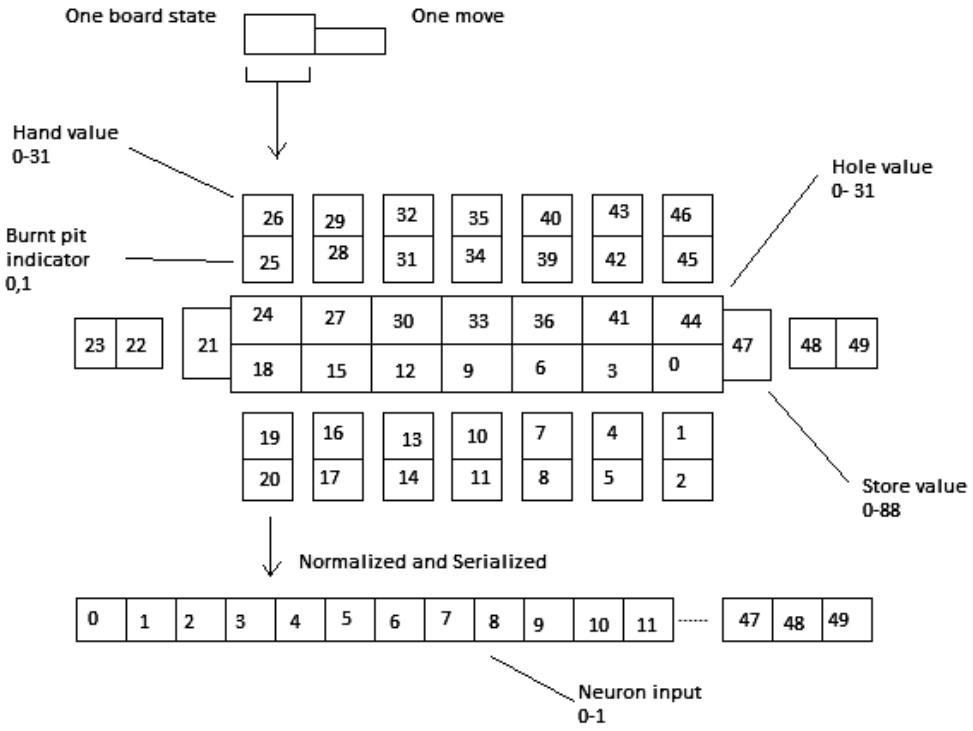

Figure 3.1: Congkak board representation for Neural Network (NN). An array of 49 input.

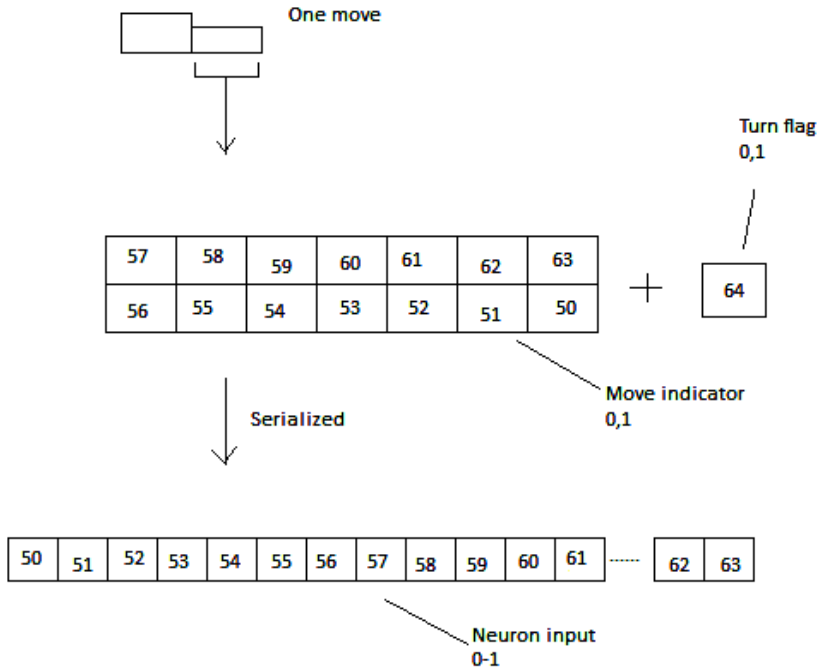

Figure 3.2: One move state representation for NN input. An array of 14 input.

For training: The pebble portion of the board state (the first part) is feed into the input neurons and the move-state portion (the second part) is assigned to the output neurons. Then, NN was trained to match the output of its neurons with the value assigned to it. This mean that the NN will try to association the board state with its corresponding move state .

For larger NN: the input is a 2 complete board state followed by 1 latest board state truncated at it's pebble portion, and the truncated move state portion was assigned to the output neuron. This mean that the network will be training to associate the history of the game board with its resultant move. A network such as this require at least 166 input neuron for Congkak and 64 for Mancala. Figure 3.3 (next page) shows a NN with 2 complete board state and 1 truncated board state as input.

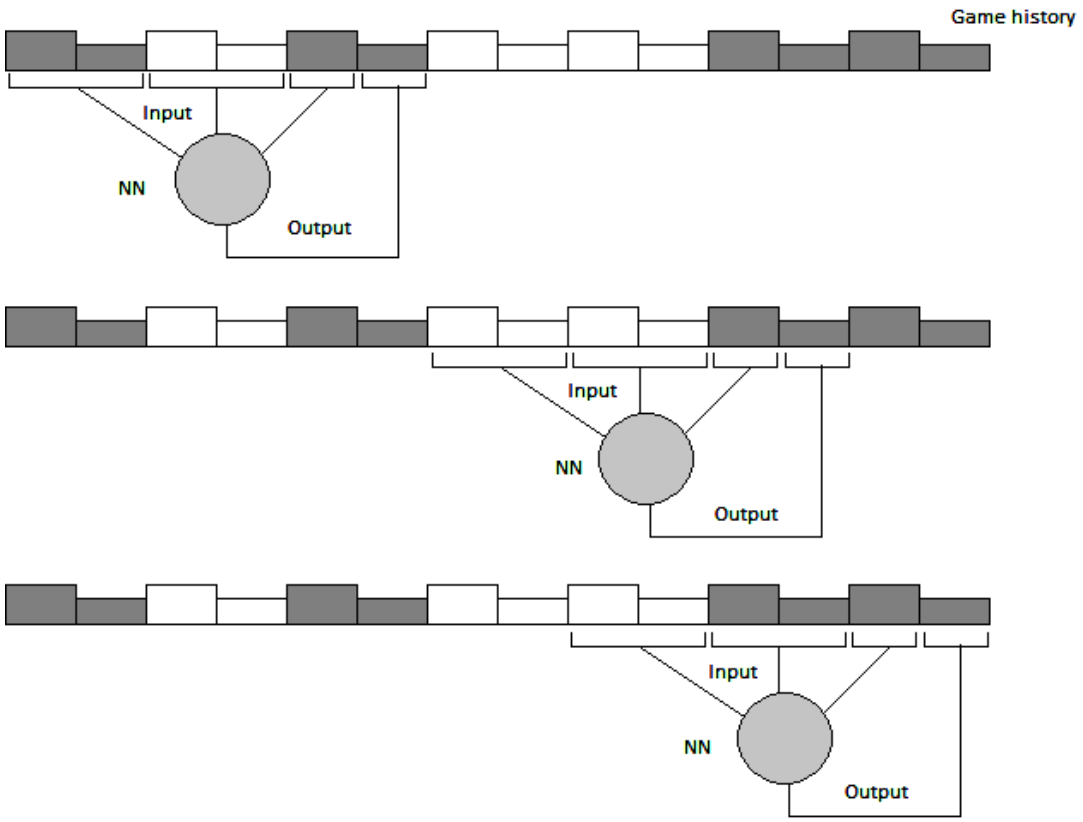

Figure 3.3: A typical NN training configuration. 2 complete board state + 1 truncated board state is feed into the NN, and only the winning move was selected as output (in this case only the dark shaded move-state was trained as output).

To overcome stackOverFlow error: Java Virtual Machine (JVM) was configured to use 1 Megabyte of stack memory. StackOverFlow error occurs in this project because Neuroph (version 2.5b) is using to much of JVM's working memory (due to large size of the NN). The stack size can be changed by adding "-Xss1024k" to Java.exe during its execution.

The idea of using 3 different network at once is to make the system utilize output from multiple neural network in parallel. Which is to create 3 network with different size: which learn with different size of input and deliver its own set of output all at once. In that way: one network will train with one board size game history, another with 2 board size game history, and the third with 3 board size game history. This aim to allow larger network to learn

larger context of the game history (such as a multiple turn move) and the smaller network to learn only the simple move.

The move is selected from the networks by selecting the neurons that output the largest value. Large value imply a winning move; but the definition of 'winning' depends on the training-evaluation-function (which will be discussed next). So, if neuron number 2 outputted a 0.99 then this mean the move should be on hole 2. Figure 3.4 (below) shows how move was selected from NN output.

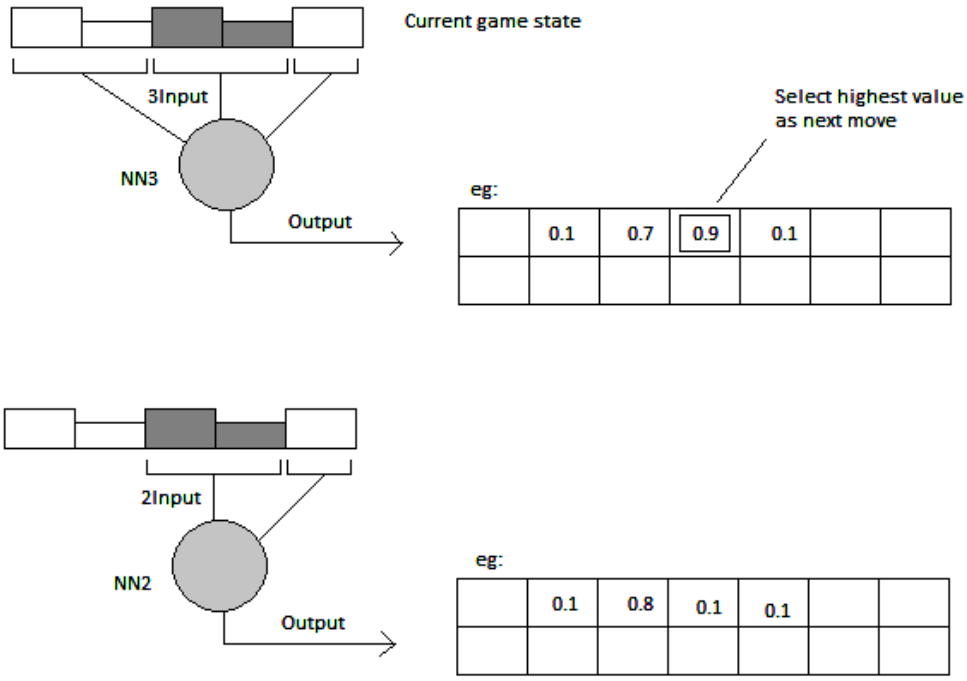

Figure 3.4: Method for selecting move from 2 parallel NN output. Only the highest value is selected, and if the first selection is invalid: the second highest value is selected instead.

A precursor to large value selection scheme is the “difference/contrast” selection scheme. Such scheme select move based on whether an output is distinctly higher & different/contrasting from the other outputs (this is inspired

by Gerald Tesauro's idea on NN's certainty: where larger 'contrast' imply more certainty). In this scheme: a neuron with output of 0.65 will not be selected as move if there exist another 2 more neuron with similar value, so a random value will be used instead. The motive was to prevent the NN from making a move that it wasn't sure.

Only the winner's move is trained as NN output, loser's move will just become a trailing input for the network. This is to ensure that the network will only be trained to produce a winning move. The winner's move is selected during the training session using a “whose turn” information embedded in the game history, only the winner's turn is selected for training.

All board state used for training is treated as if it was player 1 (or player 0; depending on context), if the board state belong to player 2 then its contents would be flipped 180 degree (mimicking player 1). This is to make sure that the network didn't learn the game as a 'third person', but rather: the game was reflected back to the network as though it was all of his own move, and only winning moves were selected for training. The flipping is possible because the Congkak board is symmetrical on both side.

A set of winning moves is determined as “significant moves” or “insignificant moves” based on these 2 evaluation method (called “game evaluation” method): the first one is by calculating the absolute difference in storehouse count (between 2 player), and the second one is by calculating the increase in storehouse count (between the current round and the previous round). The first scheme was designed to allow learning of the “winning

moves” (moves that lead to current winning status regardless of the previous game), and the second scheme was designed to allow learning of “defensive moves” (moves that allow player to reclaim burnt_house; depends upon previous game). Only one scheme is used for each game, and its evaluation will effects the global-minimum-training-error.

Each single move can also be evaluated as “significant moves” or “insignificant moves” using these 2 evaluation method (called “move evaluation” method): the first one is by increasing the local-minimum-error based on the distance from endgame (initial move is more significant), the second one is to define the condition when to increase or decrease the local-minimum-error (eg: double turn is more significant). The first scheme is originated from Tesauro's idea of “distant move is insignificant to the endgame” (for Backgammon), and the second scheme is based on “custom game strategy” inspired by observing the game. This method will apply to each training elements/each move, its evaluation will effect only the local-minimum-training-error (current instance of training; there are multiple move to train). The floor value of local-minimum-training-error is set to equal to the value of global-minimum-training-error (mentioned in previous paragraph). Figure 3.5 (next page) illustrate 2 of the “move evaluation” method.

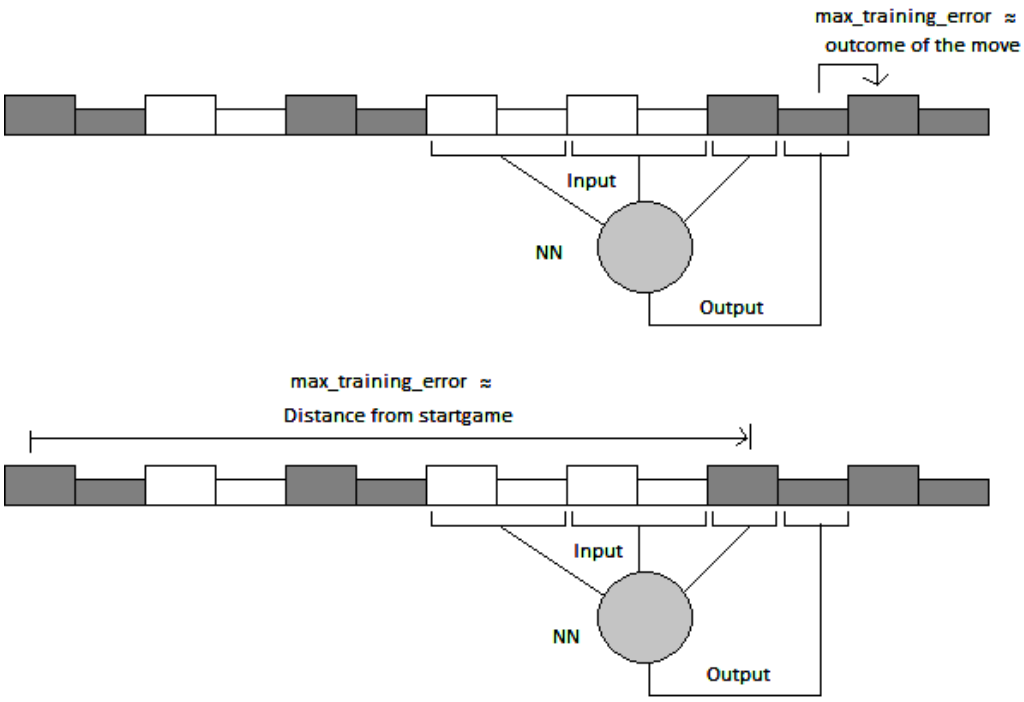

Figure 3.5: Illustration of the 2 "move evaluation" method. A max-training-error based on move type (good or bad move), or based on distance-from-endgame.

The "significance" of a move is inversely proportional to the minimum-training-error. A large minimum-training-error means that the training was "insignificant" and the training will be interrupted early to allow deviation in the output. This minimum-training-error can be changed using any kind of transfer-function (eg: exponential, linear, or IF-else statement), while the "significance" level can be evaluated using any kind of selection scheme (such as mentioned in previous paragraph).

An example of evaluation function is available in the source-code (http://congkaksystem.sourceforge.net/) in CongkakTrainer.java. There was an example of exponential function, linear function, and IF-else statement's transfer-function. The value of minimum-training-error used in for the project is between 0.01 to 0.49.

The learning rule used for this project is Backpropagation without momentum. Momentum is preferred but Neuroph (version 2.5b) caused nullPointerException whenever momentum is used (can't find solution to this issue). Momentum would allow faster learning, but if a bug exist in Neuroph then it is impossible to fix it since Neuroph is treated as a blackbox.

The learning rate was initially set at 0.1 (Tesauro, G. (1995)) but then was changed to 0.01, and the weight range is between -5 to 5. The initial learning-rate and weight value is based on Gerald Tesauro's paper. A lower learning rate can make the NN less susceptible to noise but will require longer training time (refer to Chapter 4).

For the game history: dynamic array is used. Dynamic array allow the game history to expand indefinitely without initializing an infinite array. This is demonstrated in DynamicArrayOfDouble.java; whenever the history exceeded a certain size: a new bigger array will automatically be allocated to game history, and the game history will never run out of space.

The training cycle and the data collection cycle can be configured in CongkakDataLogger.java. This sub-system can be disabled or enabled by setting 'logging' to a value of “true” or “false” in CongkakServer.java, however this sub-system is unavailable in CongkakGUI.java. By default: CongkakDataLogger.java is set to firstly measure the win-loose count for a self-play between the AI agents, and then to do a NN training with a self-play of NN, RandomMove and NNMM agent, and then to repeat everything for a certain set of cycle count.

v. Neural Network (NN) Sub-system: Extra Term and Definitions

A “nullPointerException” happens when a code or a function is calling a non-existing object. This happens when that object has never existed, or it is referring to non-existing external library or Java had automatically deleted those object. Java's automatic memory management will delete any object that is not used.

“Burnt_pit” and “burnt hole” is two different term that refer to a same thing. It means that a hole or a pit was 'burnt'. A burnt_pit (a pit that was 'burnt') is ignored during game play, it will be left empty until a player reclaim it on next round.

A burnt_pit is 'reclaimed' whenever a player collected enough pebbles to fill the pit during the start of a new round. The reason why the player had a burnt pit in the first place is because the player doesn't have enough pebble to fill this pit during the start of new round. A player need to collect at least 7 additional pebbles to reclaim one burnt pit.

“Blackbox” is the term used to describe a system that behave in a known way but its mechanism was hidden. Neuroph is a blackbox because its mechanism is entirely ignored. It is trusted that Neuroph version 2.5b is a reliable NN system.

3.2.3. Data Collection & Analysis

a. Identify the Data to be Collected

During the design and development phase all possible data from the system is collected. This is used for debugging the system and not yet for the purpose of answering the research question. Eg: the move history, the number of nodes that MM explore, NN output, and the content of CongkakBoard is among the data collected; they can reveal any unexpected behaviour that could imply a logic error in the code, which then can be fixed/corrected.

After the system is debugged, the data-collection sub-system is created to collect & save data for the research question. The data-collection sub-system export the collected data into a text file outside of the Congkak system for use by other software such as SPSS. It is designed to collect about the speed for all AI agents, win-loose count for all AI agents, and the training-time for NN agent.

b. Perform the Data Collection

Firstly, all the old ".nnet" file was deleted (this is the NN's 'brain'), then the data-collection sub-system is configured to record for about 1 training cycle (eg: set stopCycle=1 at CongkakDataLogger.java.), then the system is run and the total amount of time for completion was recorded. The time for completion for 1 training-cycle is used to determine if more training-cycle can be completed within project's time budget or not. The project aim to do about

100,000 training-count (about 10,000 training-cycle) for all 4 different type of training-evaluation-function.

During actual data collection: NN training is performed in a batch of 1 to 100 training-cycle (depending on the training's speed). After each batch is completed: the resultant ".nnet" file and the text data was copied and stored in different folder for later processing and as a backup, and then the record-ID (eg: trainingCount=10, and nNid=600 in CongkakDataLogger.java) was set to the last recorded value and the system is re-started. The training for the first NN agent was systematic (see attached softcopy), but the third NN agent was performed to do 10,000 training-count at once; unfortunately the training must be ended abruptly due to time constraint (the training took too long) and half of the training data was lost except the first 3000 training-count (Java did not output the entire text if the output file was not 'closed' properly. An abrupt termination will prevent a proper file 'close').

c. Processing the Data and Analyze the Data

All the text data was then opened with SPSS and then tediously combined into 1 file using "merge data" function in the SPSS, and then the training-count was "visually-binned" into 10 segment (to summarize the rest of the data), and finally the data was analysed using "descriptive-statistic" and processed into a "line-chart". The following is the 3 output: (1) the "mean value" of the processing speed for all AI agent, (2) the graph of win-loose count (for all AI agent against NN agent) vs. total training-count, and (3) the

graph of NN's maximum-training-time vs. total training-count. This result is displayed in Chapter 4.

Screen-capture of the GUI was also taken to show the functionality of the system. The GUI can be used to demonstrate the Congkak game playing to an audiences, and can also be used to evaluate the accuracy of the Congkak system in simulating the Congkak rule. The screen-capture was edited using MS Paint.

d. Interpret the Data and Answer Research Question

Literature review must be the basis for the data interpretation. The literature is used to either confirm the expected output or to try explain the non-ideal result. The interpretation is written down in Chapter 5, but as a summary: based on current knowledge and current data: the NN agent is performing poorly (it even loses to RandomMove) probably because the network has an issue with noise and over-fitting and because of lack of training (which is mentioned in the literature but wasn't considered seriously during the design phase), also: the MM agent is significantly faster than NN probably because of its Alpha-Beta function and also because of Neuroph itself being low performance NN algorithm (taheretaheri (2010)).

3.2.4. Project Documentation: Prepare the Report

The report was written with an aim to allow the reader to grasp the concept used in the source-code. The source-code is an important deliverable

for the project but it is not easy to be understood because it was written entirely on low-level language (for reference: source-code can be retrieved from http://congkaksystem.sourceforge.net/). Therefore Section 3.2.2: “Design & Development” had described the concepts that is used to write the low-level language (Section 3.2.2, in sub-section a), in sub-section i.-v.).

The report also aim to show the performance/capability of the Congkak system. This is the reason Graphical User Interface (GUI) was shown in Chapter 4. A screen shot was taken to show that the system is functional and could interact with users, and has achieved the project's objectives.

The report also answered the research question presented in Chapter 1. Answering research question is essential for the fulfilment of a Master project. The report describes the Congkak system as a system that has produce an output that will answer the theoretical questions presented in earlier chapter (the system can answer a variety of research question as long as a relevant algorithm is used).

Then the report must follow a stringent format and style of writing. This is to ensure that all report were written in the same way like other researcher did, and therefore allows an easy understanding of the content by readers and authors (eg: standardized citation, standardize font, standardized page number, standardized methodology). For this report: The project need to spent nearly 2 month to ensure that it follows the required format (doing correction takes considerable time).

3.3. The Limitation of the Project's Methodology

The “Data Analysis & Collection” phase was performed only at the end of the project: this leave very little time to fix the Congkak system if an issue was revealed by the data analysis. Issues such as noise, and increasing & decreasing trend of winning cannot be observed during system design because data analysis is not a priority task during the design phase (data collection is performed for immediate testing and debugging only). The data collection sub-system is also a portion of the whole design process and it only became operational once the Congkak system is completed: but then it leave only a little time to collect, process and analyse the data after the system is completed.

3.4. Summary

This chapter has described the research methodology used and also the important concept used for developing the NN's sub-system (Section 3.2.2). It has described the classes used for NN sub-system, and how the board state was represented to the NN, and what kind of evaluation function was used, and what kind of error encountered and its solution, and finally it described how the training was conducted. It also briefly describe how the project was conducted using the research methodology.

Chapter 4

4. RESULT

4.1. Introduction

This chapter focusses on showing the GUI and showing the data on the performance of all the artificial agent. It is an assumption that data-collection sub-system is working accurately, and the content of the data indicate how well the artificial agent is performing.

4.2. Brief Statement of the Result

The GUI is basic but functional. It allow user to click a button to make a move rather than typing a number. It illustrate hand move and it uses colour to emphasis important idea, it also feature a menus to select an artificial agent, and featured a 'helper' that can help player to win.

All the artificial agents can offer an artificial adversary to a human player, but the NN is not yet a good player; it looses to RandomMove and has never won against MM. However other agent worked extremely well: Random-Move makes random move as specified, and MM can defeat other player with 100% certainty.

4.3. Result and Processed Data

Data was collected using the game's engine itself and is processed into graphs using SPSS. The collected data includes: the speed of agent's algorithm, the speed of training, the number of training, and the number of winning.

The GUI screen-shots were captured using Microsoft's (MS) Windows “print screen function” and edited using MS Paint. The screen-shot shows how the GUI work. It showed that the GUI is capable of showing the Congkak board accurately.

4.3.1. The GUI

The screen shot (Figure 4.1 (next page)) shows the initial board state. The green boxes indicate the valid button for player 1 to press, and the red boxes (Figure 4.2 (next page)) indicate the valid button for player 2 to press. The player make a move by clicking on these coloured boxes.

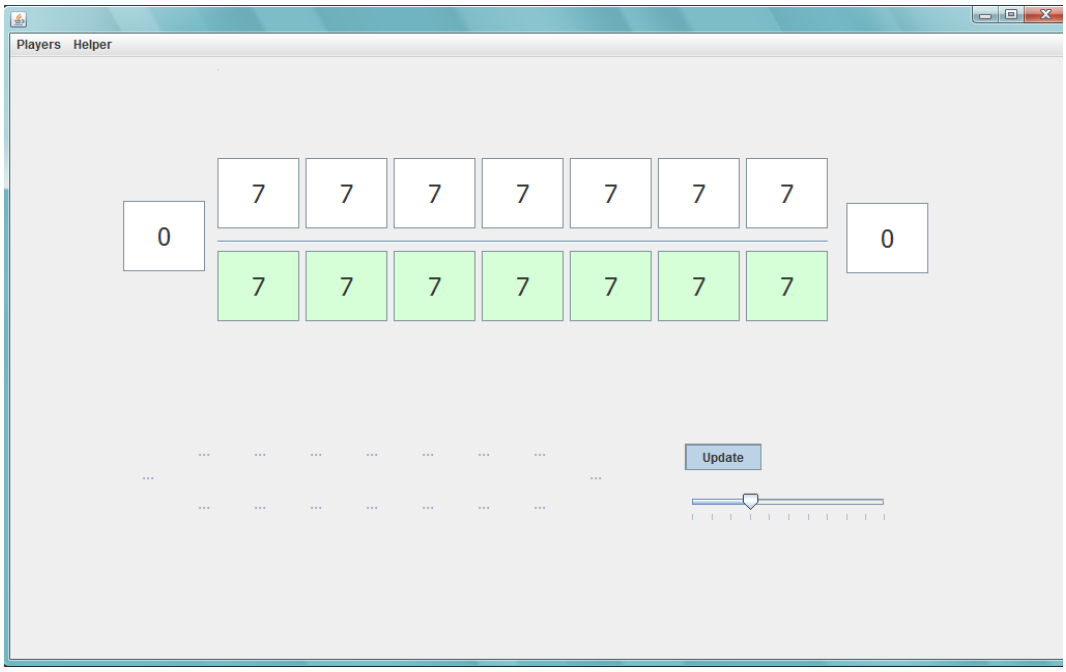

Figure 4.1: GUI waiting for input. Valid move for player 1 is coloured green.

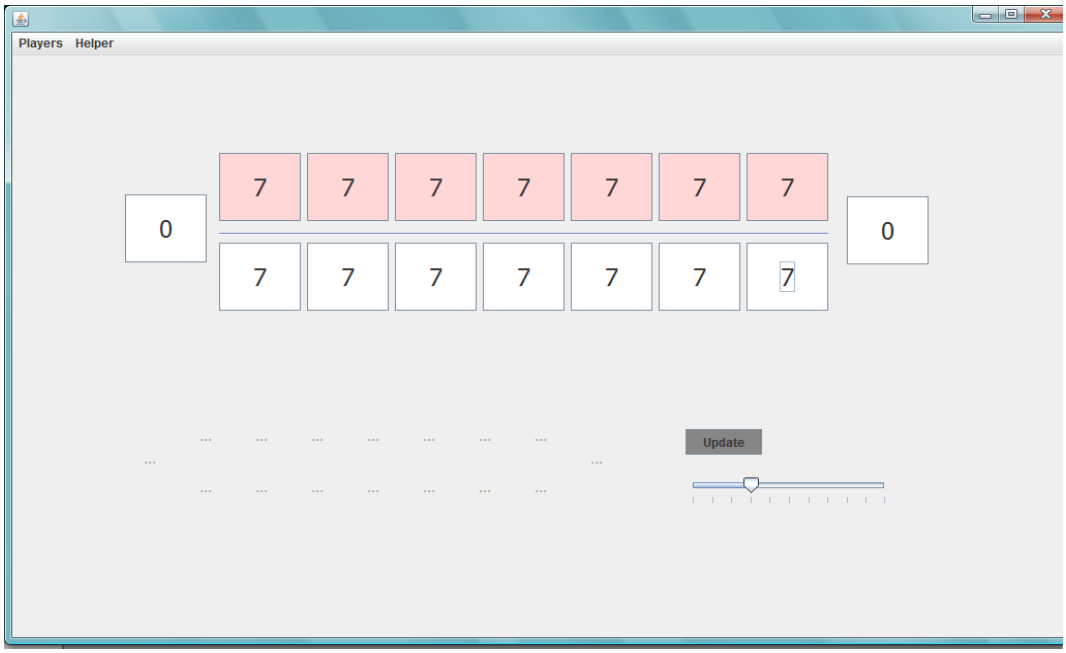

Figure 4.2: GUI waiting for input. Valid move for player 2 is coloured red.

After both player made their move: changes made to the pebble count was stored as a sequence in the Congkak system, and then it is displayed one-by-one using a time delay to simulate motions. Using these motion the player can see how the pebbles move and thus can deduce the rule of the game. The player can control the speed of this motion using the slider and the "update"

button.

The following figure (Figure 4.3 (below)) illustrate how simultaneous move and hand move is simulated. A coloured box will blink-in and blink-out on top of each hole/pit in order to simulate the motion of a hand. The speed of the blink is also determined by the slider under the “update” button on bottom right.

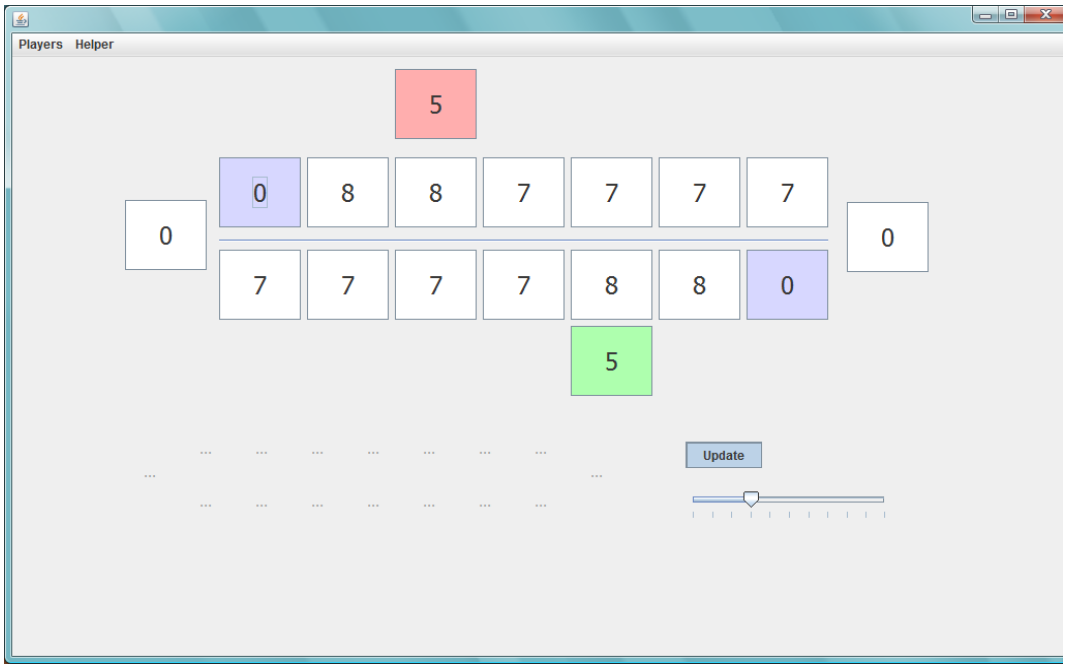

Figure 4.3: GUI Simultaneous move. Coloured box simulate player's hand.

Figure 4.4 (next page) shows the GUI pausing the game during “move_pause”. But in real game: a player must asynchronously select a new move whenever he/she has emptied his hand on his own storehouse (there is no pause). However, current Congkak engine (CongkakBoard.java) doesn't have any method to implement asynchronous move, so alternatively: the game is paused until a new move is inserted.

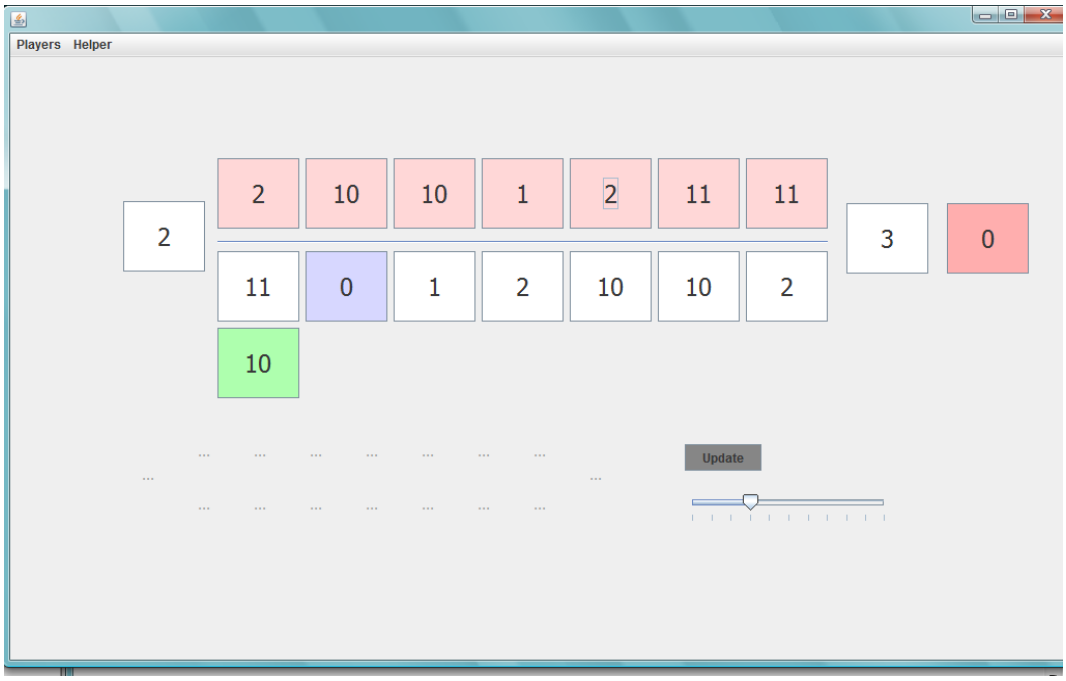

Figure 4.4: Move pause during simultaneous move. Player 2 has landed on his own storehouse: GUI is requesting another move.

Figure 4.5 (in next page) shows the sorting simulation. It shows how the available pebble is distributed to the remaining hole according to burnt_house rule. The purple boxes will blink-in and blink-out at each player's storehouse and blink-out and blink-in at the empty hole to simulate the addition and subtraction of pebbles.

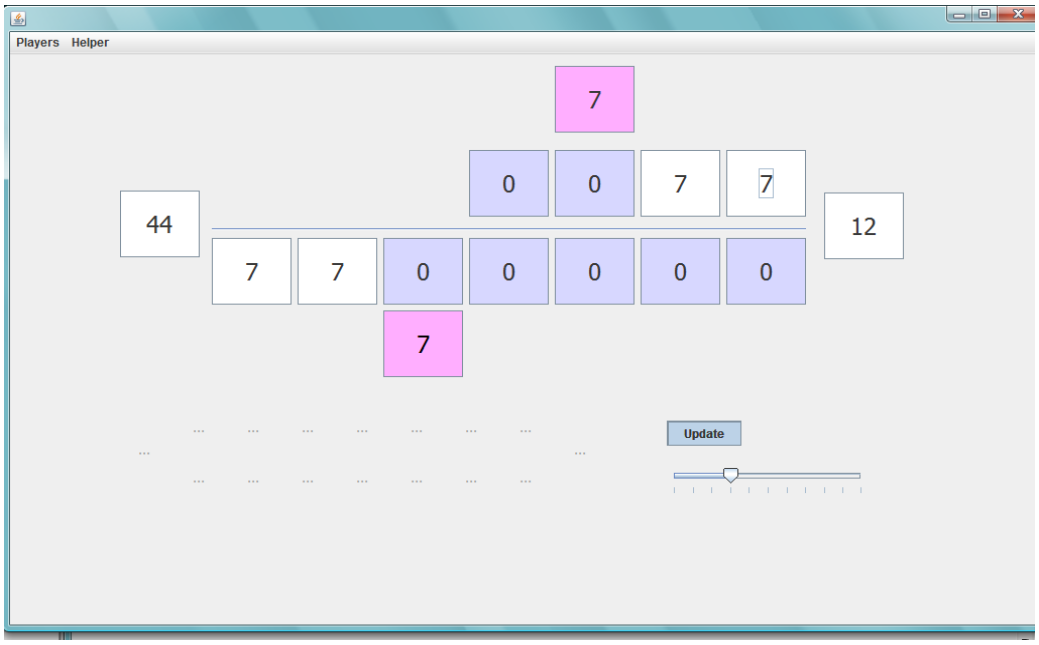

Figure 4.5: Sorting simulation. New round is initialized.

Figure 4.6 (below) shows the endgame dialogue. It display who is the winner and showed the storehouse difference, and prompt if player wanted to initialize a next round. If player press "Next Round" then the board will prepare itself for a new round, if not: then nothing will happen.

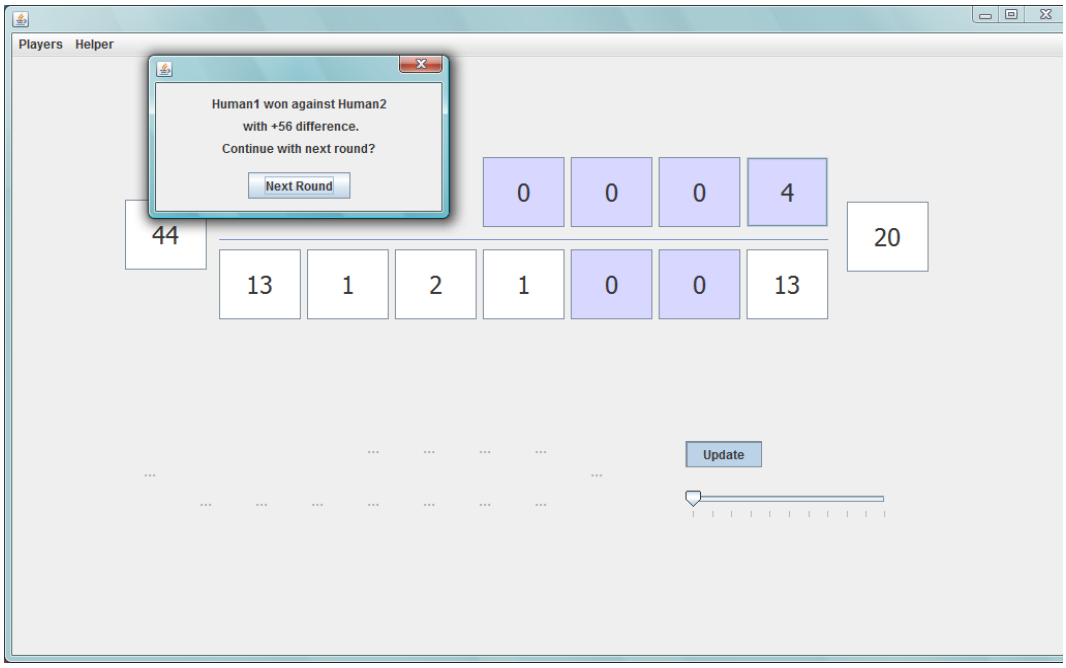

Figure 4.6: The endgame dialogue. Display winner and prompt for next round.

Figure 4.7 shows the helper function. Whenever a mouse cursor hover over a button: a GUI Calculator will display the outcome of the move, and a player can also see what move an artificial agent would make if it was playing. This would help human player defeat a difficult artificial agent.

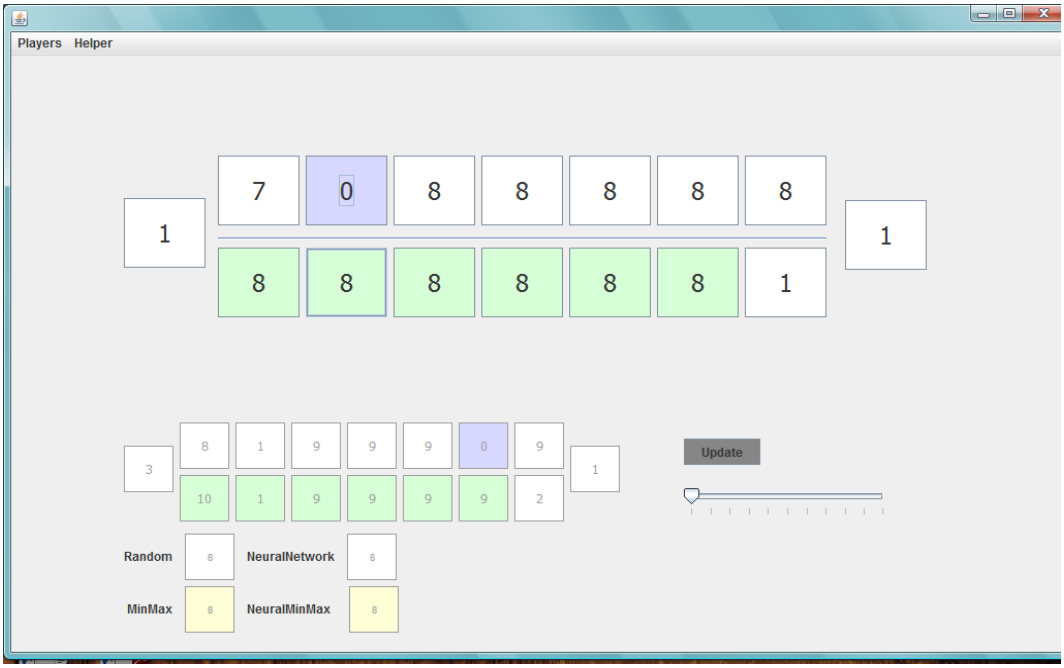

Figure 4.7: Helper function. Help player to make better decision.

4.3.2. Artificial Intelligent Agents

Table 4.1 (next page) shows the win-loose count for all AI agent. MM is the winner for all games while NN is the worse (worse than Random move). NN agent was balanced when fighting against itself but will looses more when fighting against a Random move agent, and looses completely when fighting against MM.

Table 4.1: Win-loose count for all agent. Min-Max won all game.

Artificial Agent	Winner Player 1	Player 2	Tie	Total Num. of Games	Win Ratio
Random vs MinMax	0	19		19	0|1
NeuralMinMax vs MinMax	0	20		20	0|1
NeuralMinMax vs Random	52	1		53	1|0.02
NeuralNetwork vs NeuralMinMax	0	100		100	0|1
NeuralNetwork vs Random	39	59	2	100	0.66|1
NeuralNetwork vs MinMax	0	30		30	0|1
NeuralMinMax vs NeuralMinMax	60	40		100	3|2
NeuralNetwork vs NeuralNetwork	50	50		100	1|1
Random vs Random	41	42	8	90	0.98|1
MinMax vs MinMax	0	54		54	0|1

Each testing is capped at maximum 10 rounds, and the test is repeated for 10 times for each combination of agents; resulting in total of 100 round for each combination. Notice that MM had played less than 100 round; this is because MM immediately win in each game; and on average MM only need 2 round to defeat the other agent: so it only played a maximum of 10x2=20 game. 'Constants' such as: the number of test performed, the combination of AI agent, and the training-Count is configured in data-collection sub-system (CongkakDataLogger.java).

The NN agent mentioned in Table 4.1 above is trained using “Defensive Move” game-evaluation-function and “Custom strategy” move-evaluation-function with 4000 training-count. There are also other game evaluation function (mentioned in Chapter 3), but not shown in Table 4.1 due to time constraint. In total there are 2 game evaluation function; “Winning Move” and “Defensive Move”, and 2 move evaluation function; “Custom-strategy” and “Distance-from-endgame”: resulting in a total of 4 combination.

Figure 4.8 (next page) shows the effect of training using “Defensive

Move" & "Custom Strategy" evaluation function. It shows that Random-Move had more wins than NN, and MM is unbeatable as usual. Discussion related to this and the following figure is delayed until Chapter 5.

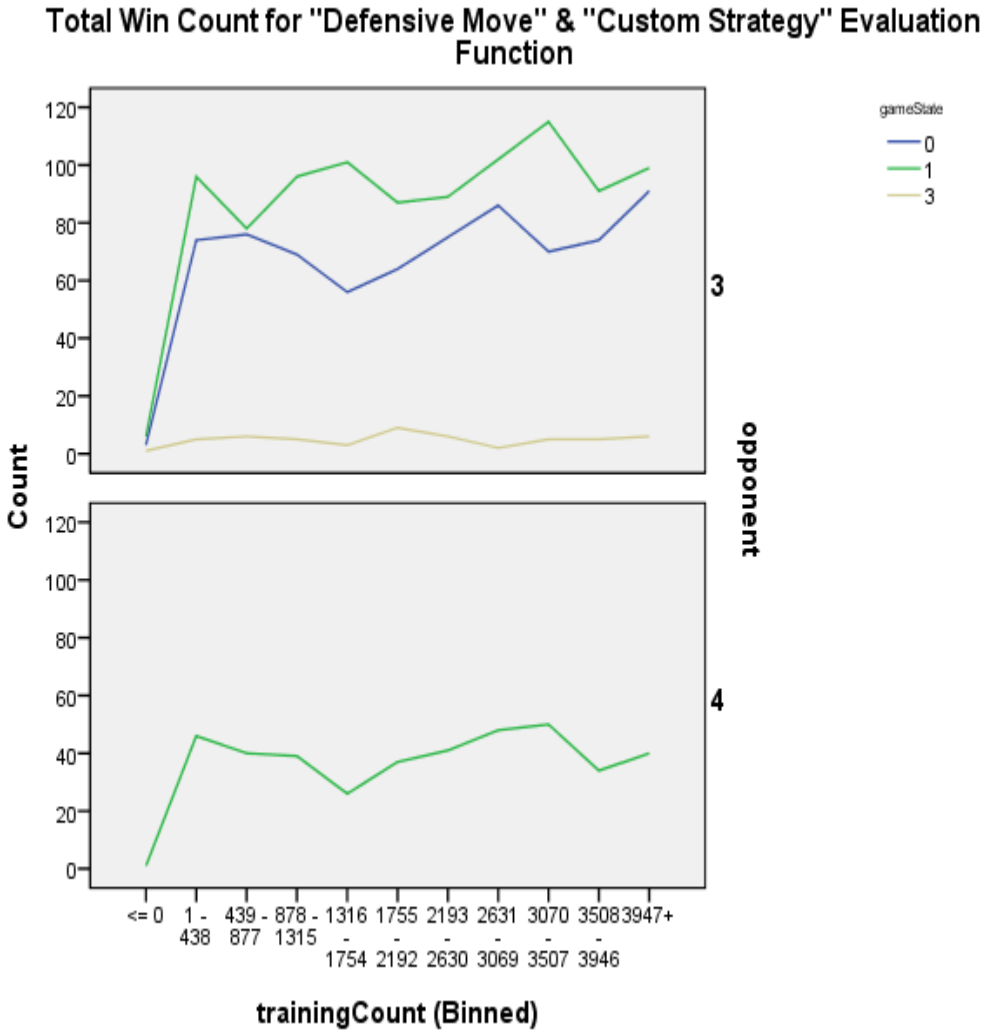

Figure 4.8: The Win over TrainingCount graph for "Defensive move" and "Custom strategy" (4000 trainingCount). Line 0 represent NN agent's wins, Line1 means opponent's wins, Line 3 means Tie, Opponent 0 is RandomMove, Opponent 4 is MM.

Figure 4.9 (next page) shows the effect of training using "Defensive Move" & "Distance from Endgame" evaluation function, however at training-count 3000 the move-evaluation-function was changed into "Custom Strategy" for testing. It showed a trend where Random-Move started to win massively at

initial, but then its win count start to fell after training-count 3000. The reason is because "Custom Strategy" move-evaluation-function had given more significance to a multi-lap move thus allowing the NN to collect more pebble.

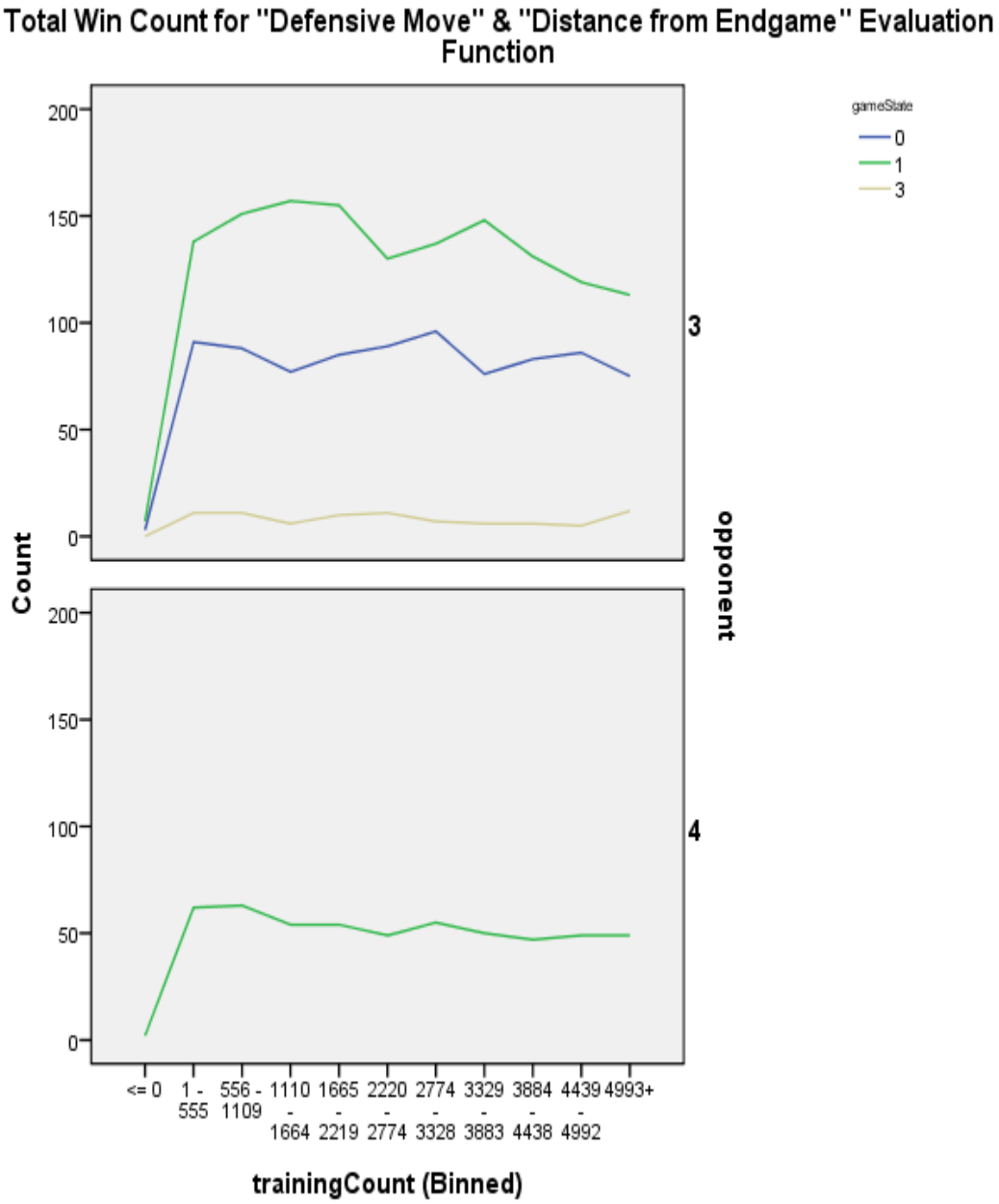

Figure 4.9: The Win over TrainingCount graph for "Defensive move" and "Distance from Endgame" (5000 trainingCount). Line 0 represent NN agent's wins, Line 1 means opponent's wins, Line 3 means Tie, Opponent 0 is RandomMove, Opponent 4 is MM. "Distance from Engame" was changed into "Custom Strategy" at training-count 3000.

Figure 4.10 (next page) shows the effect of training using “Winning Move” & “Distance from Endgame” evaluation function. This is the only session that goes over 10,000 training count, and it showed that NN is starting to win; it has more win over Random-Move after training-count reach 6000 and the amount of game with MM also increases; probably indicating that NN is becoming more resistant to MM. The training took about 3 days to complete, but eventually was stopped because NN's win-loose percentage appears to be degrading and other evaluation function also need to be tested.

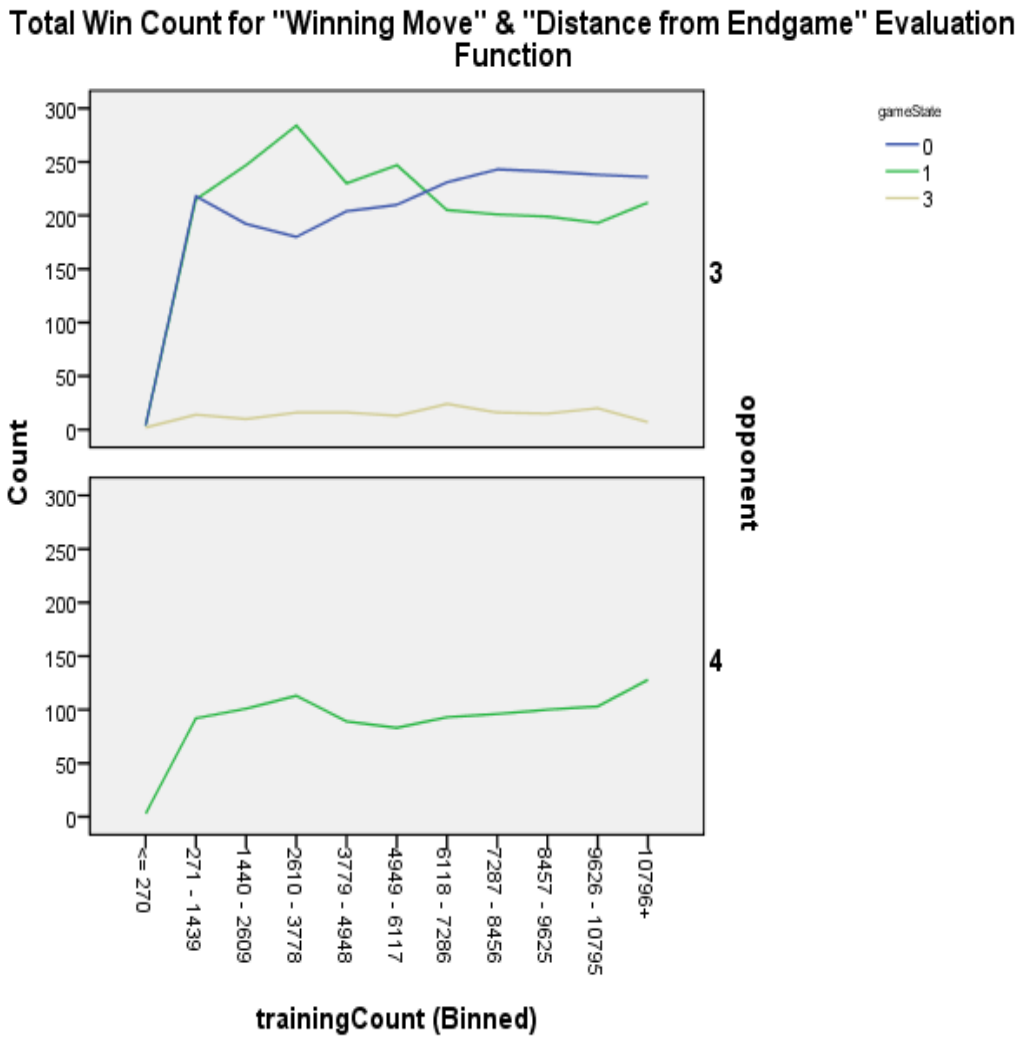

Figure 4.10: The Win over TrainingCount graph for "Winning Move" and "Distance from Endgame" (11000 trainingCount). Line 0 represent NN's wins, Line 1 means opponent's wins, Line 3 means Tie, Opponent 0 is RandomMove, Opponent 4 is MM.

Figure 4.11 (next page) shows the maximum-training-time vs, training-count for "Defensive Move" and "Custom Strategy" evaluation function. The training time will naturally decrease with the number of training because each training bring the network closer to its minimum-error, the closer the network to its minimum-error the lesser the training-iteration needed to reach it; hence less training time. The decreasing training time also mean that the network is now is closer to being able to predict the board state accurately; the small training time means that the difference between network prediction and the

actual output is sufficiently small such that it require almost no new training. (discussion related to this and the following figure is on Chapter 5)

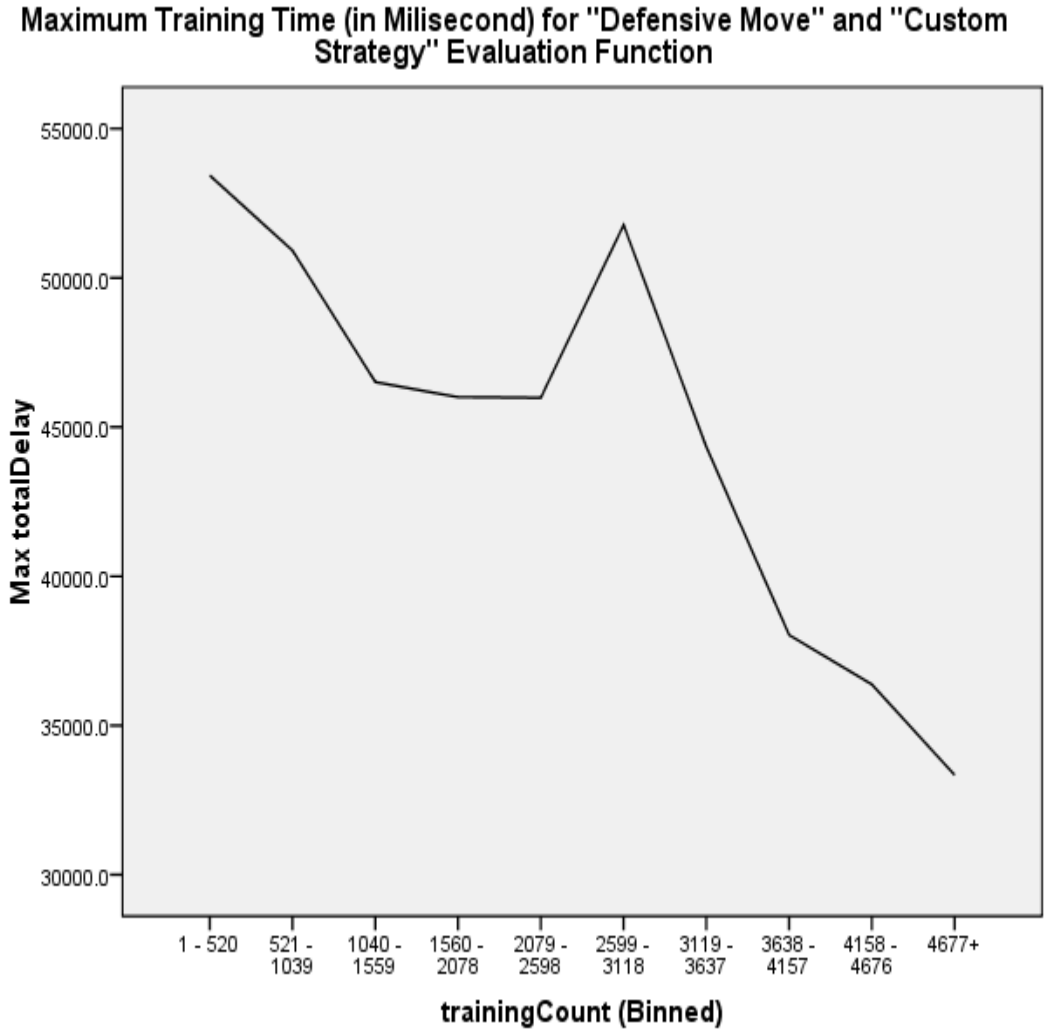

Figure 4.11: Maximum training time for "Defensive Move" and "Custom Strategy" evaluation function. Training time decrease over number of training.

Figure 4.12 (next page) shows the maximum-training-time over training-count for "Winning Move" and "Distance from Endgame" evaluation function (this was the combination that showed good result after training Count 6000 (Figure 4.10 (page 54))). It shows that: the training-time abruptly drop to a minimum after 2000 training-count, but then rise and then drop again after 6000 training-count, and then it level-off at 5 second. The level-off

means that the network is trying to learn noises but is constantly failing; but it kept trying thus this prevent the training-time from going down.

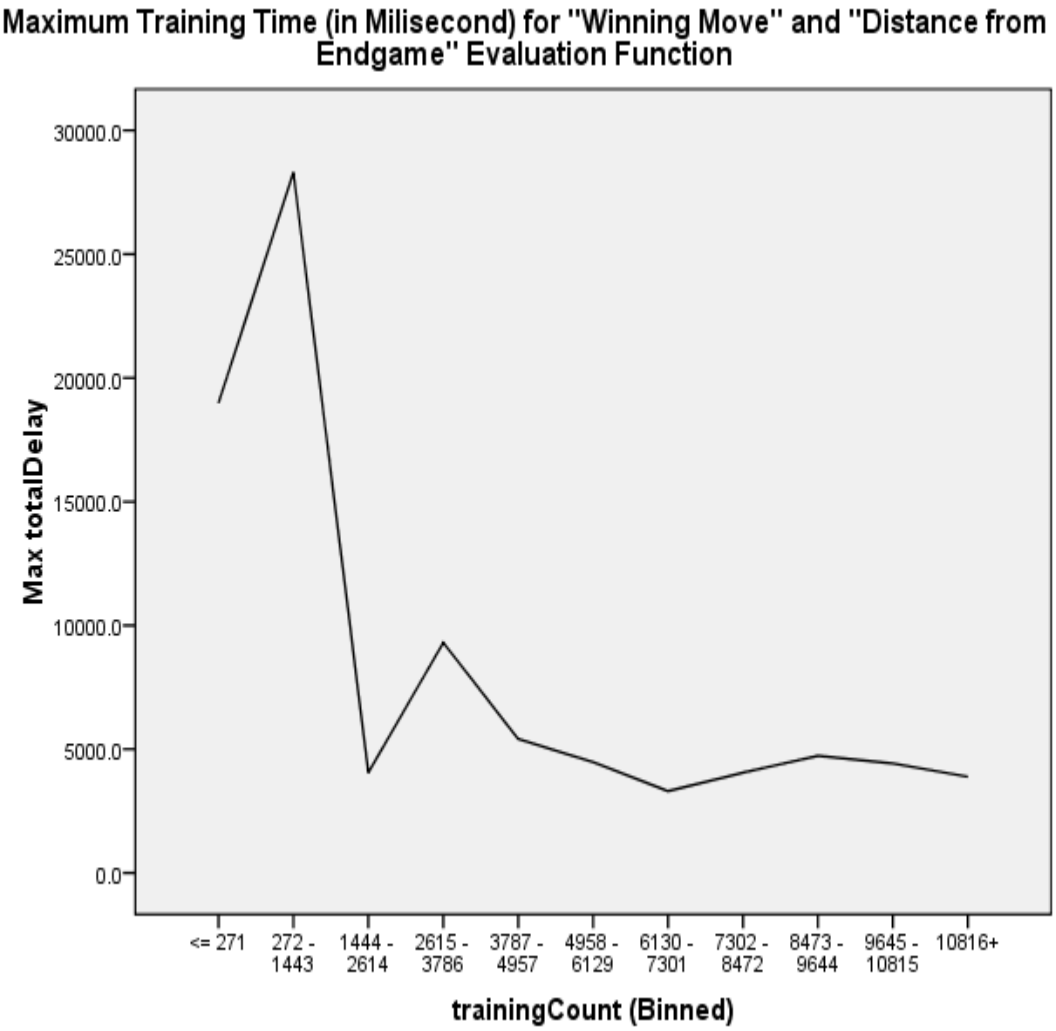

Figure 4.12: Maximum training time for "Winning Move" and "Distance from Endgame" evaluation function. Training time decrease over number of training.

Table 4.2 (next page) display the calculated speed of NN algorithm. It require on average 386.0 millisecond with a deviation about +- 69.2 millisecond to complete. This algorithm consist of several array manipulation algorithm (for about 360 array points), a several call-in to Neuroph, and a simple move selection algorithm.

Table 4.2: The speed of NN agent algorithm.

Descriptive Statistics (Miliseconds)

	N	Minimum	Maximum	Mean	Std. Deviation
Neural Network	4095	249.0	610.0	386.040	69.1850
Valid N (listwise)	4095				

Table 4.3 (below) display the speed of NeuralMinMax (NMM) algorithm. The algorithm require on average of 388 millisecond with a deviation of +- 71.2 millisecond to complete (this is almost similar to NN). But NMM is a combination of NN agent plus Min-Max agent; which mean that MM is incredibly fast when compared to NN.

Table 4.3: The speed of NMM agent algorithm.

Descriptive Statistics (Miliseconds)

	N	Minimum	Maximum	Mean	Std. Deviation
NeuralMin-Max	4095	249.0	640.0	387.864	71.1967
Valid N (listwise)	4095				

Table 4.4 (below) display the speed of Random-Move algorithm. The algorithm appear to operate almost instantaneously with 0.015 milliseconds delay on average. RandomMove is consist of a call-in to Java's Random class and some simple algorithm that check and select for valid move.

Table 4.4: The speed of RandomMove agent algorithm.

Descriptive Statistics (Milisecond)

	N	Minimum	Maximum	Mean	Std. Deviation
RandomMove	4095	.0	16.0	.015	.4368
Valid N (listwise)	4095				

Table 4.5 (next page) display the speed of MM algorithm. The MM require on average only 5.3 millisecond, with a deviation of +- 9.1 millisecond to operate. MM algorithm consist of repetitive call-in to CongkakBoard.java

and CongkakNode.java, and it also contain an algorithm that evaluate the nodes and also Alpha-Beta cut-off algorithm. The MM in this project was limited to search 5-plies only.

Table 4.5: The speed of MM agent algorithm.

Descriptive Statistics (Miliseconds)

	N	Minimum	Maximum	Mean	Std. Deviation
Min-Max	4095	.0	78.0	5.257	9.1013
Valid N (listwise)	4095				

4.4. Summary

This Chapter showed that NN require more training. The maximum training ever achieved was only 10,000, whereas real research on NN usually involve training up to 100,000 count (Tesauro, G. (1995); Lawrence, S. (1997)).

The problem is due to lack of computational resources. The CPU used for this project (single core 1.666Ghz) is too slow to even perform a 10,000 training set.

In summary this chapter did not reveal any immediate flaw with the system except the failure of NN training to reach its target value of 100,000 training count. In overall the Congkak system appear to be working as expected.

Chapter 5

5. DISCUSSION OF THE RESULT

5.1. Issues

5.1.1. The GUI's Simultaneous Move

The game is on default started with a simultaneous move but the GUI (Figure 4.1 (page 45), Figure 4.2 (page 45)) didn't appear to accurately reflect a simultaneous move; the system demanded an inputs from player 1 before demanding an input from player 2 (suggesting a turn-move). This issues is not a problem. The game engine will not perform any calculation until both move was inserted; this is to emulate the simultaneous move.

Taking turn to insert move can be beneficial to both players. Player 1 can hide his own move from player 2 (and vice-versa) by hiding where he pressed the button. Hiding the first move is the main reason why simultaneous move in Congkak even exist; to prevent second player from exploiting the knowledge of the first move. The GUI work in such a way that the first move is not displayed, where the second player must decide its move independently of player 1's move.

5.1.2. Possible Aesthetic Improvement

To make the GUI appear like a real game: any artful illustration can be added to replace the boxes used in the GUI. Woodcarving figure and illustration of a hand could be used to make the GUI appear more Congkak rather than like a calculator. Java Swing has this feature which allowed those boxes to be replaced with a picture or an icon.

Animation programming could also allow the hand to move more fluidly. Currently the boxes only blink-in and blink-out at fixed position. If animation was use: hundred of boxes need to be drawn on top of each hole and it will blink-in and blink-out into existent at a rate of about 20 frame-per-second (emulating smooth motion).

5.1.3. The Move Pause

The move pause is a simplification of an actual game. It is similar to a person who can instantaneous pick a move and then start moving (without delay). This is not a problem, but perhaps an option could be made to allow a delayed move to be made; making the system operate more like a real Congkak, however this is not done in this current Congkak system because it would add extra complexity to the artificial agent's system; because the agent would be required to perceive a delayed move instead of just the player's instant turn.

Also, having a different rate of motion (another characteristic of real

Congkak) will require complicated new algorithm in CongkakBoard.java. It will require a difficult but not impossible system where 2 asynchronous thread run in parallel with each other and interact with the board like 2 independent player do (JuMpErFLY (April 24th 2004)). The GUI itself also used external thread to do asynchronous periodic screen update which apply similar concept. The interval between update was controlled by the slider on the bottom right and the thread will periodically push the "Update" button (in the GUI) to allow the GUI to update its screen, but various flow control must be devised to prevent the an asynchronous thread from effecting the system unexpectedly.

5.1.4. Congkak and Numbers

The GUI uses a lot of numbers. Perhaps playing Congkak will make people be more intuitive about numbers (Voogt ,A de (2001)). This can be used in education.

5.1.5. More Functionality for GUI

A complete GUI should have the options to train the NN, and a button to restart the game, and option to set the cut-off depth (of MM), and the ability to input the random-seed of Random-Move. Currently the GUI only show a representation of Congkak Board and provided a way to input move by clicking, and none of the feature mentioned above is available. At present the NN can be trained using CongkakServer.java, and the cut-off depth can be set by editing CongkakMMMove.java, and the random-seed can be set at

CongkakRandomMove.java.

5.1.6. Real Congkak and Simulated Congkak

Real Congkak is a very slow pace game (Yaakub Rashid (1981)). In real life it would take hours and hours of playing before one can totally defeat the opponent. The player would need to perform many long multi-lap pebble dropping, and need to repeat several round before one one side decided to surrender, or is defeated like how MM defeated its opponent. But fortunately Simulated Congkak can perform all those function instantaneously: which turn Congkak into a fast pace game.

5.1.7. Issue: First Move is not Hidden to Artificial Agent.

Notice that MM agent win more against itself when placed on player 2's position (Table 4.1 (page 50)). This is an example of how the knowledge of first move give an advantage to the second player. This is an unintentional failure during development, it should have been corrected; the problem occur because both MM uses the same CongkakBoard.java to run their simulation, obviously the first player's input would effect the way the simulation would run. An adjustment can be done to fix this but this problem was only realized now. (similarly, CongkakDataKnitter.java would also have same problem too)

5.1.8. Issues with the Training Graph

The x-axis of the training graph in Figure 4.8 (page 51), Figure 4.9

(page 52), and Figure 4.10 (page 54) is the training-count binned into 10 segment of equal size, the Y-axis of the graph is the number of win for each agent counted for each bin: the green line represent the Opponent's win and the blue line represent the Neural Network's win. The win count for each graph is different because the bin size is different; the graph with large bin will count more win. The binning has an unequal size because the training count is not made constant; only one training session reach 10,000 training-count, however the graph issue is not a serious problem.

5.1.9. Noise During Neural-Network Training

Noise (described in Table 4.12 (page 47)) can be reduced by creating a better evaluation-function. A better evaluation function will 'flag' any move that is important for learning and 'thumbs down' any worthless one. For example: move that increase storehouse count by 1 is worthless because almost all move can increase storehouse count by 1, the problem is: which move did increase storehouse count by 2 or more? This move should be emphasised using better evaluation-function.

Also, the effect of noise can be reduced by using small learning-rate. Small learning-rate will prevent the network from changing too much due to noise (Tesauro, G. (1995)). Having a slow learning-rate means that the network will only change slightly during training and most change due to noise will be cancelled-out by another set of noise due to the symmetric nature of noise (noise is a move that did not contribute to winning), but when an

asymmetrical change was encountered (such as a move that only appear with specific kind of condition: such as one that caused double move) the network will have a net positive change.

However, using a large learning-rate will prevent the network from being trapped in local-maxima. A local-maxima occurs when a small change in the network's weight would not reduce the total-error any further (instead it increases the relative total-error): therefore the network reject any further changes and thus remain in this state forever (and no more training can improve the network's total-error). However, a sufficiently large weight change can allow the network to escape the local-maxima; this can be induced by using a sufficiently large learning-rate during training.

5.1.10. Possible Reasons for Neural-Network Poor Performance

Apparently NN is just too slow and too difficult to train. Perhaps the network size is too big; big network is susceptible to over-fitting (which causes prediction error) and big network require big array processing which waste CPU time. A test should be performed to see if smaller network could perform better.

By comparison: previous NN for Mancala is 2 times smaller than current Congkak's NN. Mancala's Network is trained significantly faster than current Congkak's network and there's no problem associated with JVM's stack size. The previous Mancala's biggest Network uses 198 neurons

(64,52,40,28,12) while current Congkak's biggest Network uses 464 neurons (172,132,93,53,14).

The NN is either over-fitting or need more training. More-training is a most likely the answer because currently the network has less that 10,000 training count.

5.2. Summary

The Congkak system contain no serious issues except the “first move” issue. The “first move” issue must be fixed first before any NN training is to be done; this is to make sure that NN is to be trained with a correct representation of the Congkak rule, other issues may or may-not require any future development.

Chapter 6

6. CONCLUSION AND RECOMMENDATION

6.1. Conclusions

The project has achieved almost all of its goals and objectives. The project has produced a Congkak system that could simulate (or accurately approximate) a real Congkak game, and also featured 3 type of AI agent that can be configured to pit against each other or to pit against human player, and also allowed a human player to pit against another human for the game of Congkak. In summary:

- A human player can play the Congkak game through the Congkak system using a mouse click.
- The NN agent has the ability to learn through self-play, while MM agent has the ability to defeat all other AI agent despite being limited to depth 5 search.
- The output produced from the Congkak System allowed the research questions to be answered.

The project also answered 1 of 2 research questions posed in Chapter 1. The following is the conclusion derived from the data:

- NMM hybrid agent is significantly slower than MM agent alone. NMM's algorithms took on average 380-milliseconds longer time to process than MM's algorithms, (both MM and NMM is programmed with Alpha-Beta Cut-off function and is limited to 5 depth search). The idea that “NN can improve MM's performance” is dis-proven.
- The best NN training evaluation function is not yet determined. Result has not yet been conclusive because training could not reach the targeted 100,000 training count. For the moment: training based on store-count (“Winning Move”) and “Distance-from-endgame” appears to be the best so far.

Future project can explore the NN's performance issue and complete the NN training to yield a more definitive conclusion.

It is also possible that this project is the first demonstration of NN agent for Congkak game. Previous NN research was not focussed on Congkak (but was focussed on Dakon and Awari), but now the project has filled those gap. Hopefully the project can also attract future research on Congkak; which will make Congkak the center of attention (and help to preserve the game from being forgotten). The Congkak system can be found at the public domain at http://congkaksystem.sourceforge.net/ .

6.2. Recommendation/ Future work

The next step would be to fix the “first move” issue (as mentioned in Chapter 5). Fixing this issue will probably improve NN's training performance (since it is a better representation of the game's actual rule), and MM will play much fairer when it cannot see the first-player's first move. This will involve a minor restructuring of the CongkakDataKnitter and CongkakMMMove

The second step is to re-test the MM but without its Alpha-Beta function. The goal is to compare the performance of a pure Min-Max (MM without Alpha-Beta) against a hybrid pure Neural-Min-Max algorithm (NMM without Alpha-Beta). This way we can really see if NN did offer any performance advantage or not.

The third step is to test the system with different NN package. There is other much faster open-source NN package called Encog (taheretaheri (2010)), which could be used instead of Neuroph. According to the benchmark test performed by the Neuroph's team themselves: Encog (version 2.4) is 9 times faster than Neuroph version 2.5, which mean: a NN training can be completed 9 times faster if using Encog.

The final step is to continue with the NN training, and to try different combination of evaluation function, and to see which work best. The motive is to find the result that might be useful to other kind of NN problem. All other control system that is using NN is also based on time-series input (which is similar to the Congkak system).

Thus, by understanding all the peripheral system that make NN perform better (such as a better evaluation function, and better board representation): other NN system can also be build to perform better as well. For example: having understood why an evaluation function work on Congkak could help understand how to make an evaluation function work better on other system too.

Also, the project can proceed to do a more advanced literature review. The fact that system building is a difficult task to explain meant that it is rarely documented; but most literature still refer to such concept again and again and thus making them very difficult to understand, but having understood all the basic of system building from the project: those unreachable concept can now be understood as well. Therefore a complete understanding of the literature can be progressively achieved.

Future project can also focus on testing different kind of NN topology. Currently the project has not yet able to explore with different kind of NN topology and is using NN topology that is shaped like a triangle (eg: large number of neurons at input and progressively lesser neuron near the output). It is not know what the effect would be to the NN's performance if different topology is used (eg: ring shape).

REFERENCES

5up3rJ (aka SuperJ) (Jan 4, 2006). Java Discussion Thread: Time Delay [Msg 3]. Message posted to http://www.daniweb.com/software-development/java/threads/37443

Ahlschwede, J. (2000). *Using Genetic Programming to Play Mancala*. Retrieved on 13.3.2011 from http://www.corngolem.com/john/gp/project.doc

Alex de Voogt (2001). Mancala: A Game That Count. *Expedition Philadelphia*, Issue 2001. Penn Museum. Retrieved on 10.3.2011 from http://www.penn.museum/documents/publications/expedition/PDFs/43-1/Mancala.pdf

Alifia; Frilla Ariani; Tania Krisanty, (2006) Pencarian Solusi Optimal Pemilihan Lubang pada Permainan Congklak dengan Algoritma Greedy dan Program Dinamis. In *Majalah Jurusan Teknik Informatika*. Institut Teknologi Bandung, Bandung

Blais, A., Mertz, D. (2001) *An introduction to neural networks*. DeveloperWorks, IBM. Retrieved on 13.3.2011 from http://www.ibm.com/developerworks/library/l-neural/

Bylander, T. (2008). CS 3793 Lab 3: Mancala min-max assignment. CS 3793: Introduction to Artificial Intelligence Course. University of Texas at San Antonio. Retrieved on 13.3.2011 from http://www.cs.utsa.edu/~bylander/ cs3793/ lab3.pdf

Bylander, T. (2007). CS 5233: Best-Case Analysis of Alpha-Beta Pruning. *CS 5233: Artificial Intelligence Course.* University of Texas at San Antonio. Retrieved on 20.3.2011 from http://www.cs.utsa.edu/~bylander/ cs5233/a-b-analysis.pdf

Brunei Darussalam Information Department (2008). *Congkak: A Game of Wits*. Brunei Darussalam Newsletter. Retrieved on 3.4.2011 from http://www.information.gov.bn/bdnewsletter/index.php?option=com_content&task=view&id=95&Itemid=44

Carter-Greaves, L. E. (2009). *Time Series Prediction With Feed-Forward Neural Networks*. Retrieved on 13.3.2011 from http://neuroph.sourceforge.net/TimeSeriesPredictionTutorial.html

Cofer, A. (2003), *Mancala In Java: Experiment in Artificial Intelligence and Game Playing.* Departmental Honor Thesis. University of Tennesse at Chattanooga.

Congkak (2011). In *Wikipedia*. Retrieved on 13.3.2011 from http://en.wikipedia.org/wiki/Congkak

CPU Time (2011). In *Wikipedia*. Retrieved on 25.5.2011 from http://en.wikipedia.org/wiki/CPU_time

Davis, J. E., Kendall, G. (2002). An Investigation, using Co-Evolution: to Evolve an Awari Player. *In Proceedings of 2002 Congress on Evolutionary Computation (CEC2002) (pp. 1409-1413).* IEEE Press

Donkers, J., Alex de Voogt; Uiterwijk, J. (2000). Human versus Machine Problem-Solving: Winning Openings in Dakon. In M. Baud; R.A.H.D. Effert; M. Forrer; F. Hüsken; K. Jongeling; H. Maier; P. Silva; B. Walraven (Eds.), *Board Games Studies (pp. 79-91), Vol. 3*. CNWS Publication, Universiteit Leiden.

Deitel, P. J.; Deitel, H. M. (2006). *Java: How To Program* (7th Edition). Pearson Education Inc, New Jersey.

Dennis, A.; Wixom, B.H.; Roth, R. M. (2010). *System Analysis and Design* (4th Edition). John Wiley & Sons Inc, New Jersey.

Gibson. E (2009). E3: Molyneux and Milo Xbox360 Interview. Eurogamer.net. Retrieved on 19.3.2011 from http://www.eurogamer.net/articles/e3-project-natals-molyneux-and-milo-interview

Eck, D. J. (2006). Introduction to Programming Using Java, Fifth Edition: *Section 7.3: Dynamic Array and ArrayList*. Department of Mathematics and Computer Science, Hobart and William Smith Colleges. Published online under Creative Common Attribution licence. Retrieved on 3.4.2011 from http://math.hws.edu/javanotes/c7/s3.html

Ghory,I. (2004). Reinforcement Learning In Board Game. *Technical Report CSTR04-004.* Retrieved on 20.3.2011 from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.4.7712

H. Jaap van den Herik, Uiterwijk, J, Jack van Rijswijck, (2001) Games solved: Now and in the future. *Artificial Intelligence (pp. 277–311), Issue 134* (2002). Elsevier Science B.V.

IBM (2011). *IBM – What is Watson?*. In IBM Watson. Retrieved on 19.3.2011 from http://www-03.ibm.com/innovation/us/watson/what-is-watson/index.html

Irving, G., Donkers, J., Uiterwijk, J. (2000), Solving Kalah. *ICGA Journal*. California Institute of Technology, Pasadena, and Universiteit Maastricht, Maastricht.

Mohammed Daoud; Nawwaf Kharma; Ali Haidar; Julius Popoola. (nd). Ayo, the Awari Player: How Better Representation Trumps Deeper Search. *Evolutionary Computation, 2004. CEC2004. Congress on (pp. 1001-1006), Vol.1.* Concordia University, Montreal.

Newborn, Monty; Newborn, Monroe (2003). *Deep Blue: an Artificial Intelligence milestone (1st Eds.) (pp. 228-229)*. Springer-Verlag Inc, New York.

Pickhard, A. (2007) Final year project of evolving AI for Mancala. Retrieved on 13.3.2011 from http://amplizine.com/ wp-content/uploads/2008/10/mancala.doc

Pinto, P. (2002). Introducing the Min-Max Algorithm. Retrieved on 13.3.2011 from http://www.progtools.org/ games/tutorials/ ai_contest/ minmax_contest.pdf

Perpustakaan Negara Malaysia (2000). *Congkak Warisan Negara*. Retrieved on 13.3.2011 from http://malaysiana.pnm.my/03/0305congkak.htm

Romein, J. W.; Bal, H. E. (2003). Solving the Game of Awari using Parallel Retrograde Analysis. *IEEE Computer (pp. 2003), Vol 36.*

Smith, P. (2000) Mancala Games. *MindZine.* Mind Sport Worldwide, msoworld.com Retrieved on 13.3.2011 from http://www.msoworld.com/mindzine/news/classic/mancala.html

Shah Ali Reza Yaacob (2006). Knowledge-Based System Development For The Game Congkak: Bachelor Thesis. *UNIVERSITI TEKNOLOGI MARA Digital Repository.* Universiti Teknologi MARA. Retrieved on 13.3.2011 from http://eprints.ptar.uitm.edu.my/990/

Steinhauer, V. (2009). S*tock market prediction using neural networks: An example for time-series prediction* . Retrieved on 13.3.2011 from http://neuroph.sourceforge.net/tutorials/StockMarketPredictionTutorial.html

Kronenburg, T. (2008). *Towards a Quasi-Endgame-Based Bao Solver*; Master Thesis. Universiteit Maastricht, Maastricht. Retrieved on 13.3.2011 from http://www.manqala.org/docs/thesistomkronenburg.pdf

Kothari, C.R (2004). *Research Methodology: Methods and Techniques (2nd Edition) (pp. 10-21)*. New Age International Publisher ltd., New Delhi.

Oracle (2011) *Understanding Instance and Class Members*. The Java Tutorials. Oracle, oracle.com. Published online under Java Tutorial SE Tutorial Copyright and Licence. Retrieved on 3.4.2011 from http://download.oracle.com/javase/tutorial/java/javaOO/classvars.html

Oracle (2011). *Java SDK download (JDK 6 Update 24)* [Software]. Oracle, Sun Microsystems Inc., Retrieved on 13.3.2011 from http://www.oracle.com/technetwork/java/javase/downloads/index.html

Wee-Chong, O.; Yew-Jin, L.(2003). An Investigation on Piece Differential Information in Co-Evolution on Games Using Kalah. *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on (pp. 1632 – 1638), Vol.3*.

Sevarac, Z. ; Goloskokovic, I. ; Tait, J. ; Morgan, A. ; Carter-Greaves, Laura (2008). *Neuroph main page: resource for Neuroph*. Retrieved on 13.3.2011 from http://neuroph.sourceforge.net/index.html

JuMpErFLY (April 24th 2004). Java: JButton and actionPerformed [Msg 7]. Message posted to http://forums.bit-tech.net/showthread.php?t=56894

Nur Liyana Zainal Abidin. (2011). *"Dunia Matematik: Mancala"*, Komuniti Dunia Matematik, online magazine issue: April 2011, volume: 5th. Universiti Teknologi Malaysia, Johor Bahru. Retrieved on 25.5.2011 from http://mathed.utm.my/duniamatematik/index.php/permainan

Bonny, T. (2010). *Congkak*. National Library Board, Singapore. SingaporeInfopedia. Retrieved on 3.4.2011 from http://infopedia.nl.sg/articles/SIP_1733_2010-11-26.html

taheretaheri (2010). *Benchmarking and Comparing Encog, Neuroph and JOONE Neural Networks.* Retrieved on 6.6.2011 from http://www.codeproject.com/KB/recipes/benchmark-neuroph-encog.aspx

Tesauro, G., Sejnowski, T. J. (1989). A Parallel Network that Learn to Play Backgammon. *Artificial Intelligence* (pp. 357–390), Issue 39 (2002). Elsevier Science Publisher, B.V.

Tesauro, G. (1992). Practical Issues in Temporal Difference Learning. *Machine Learning (pp. 257-277),* Issue 8, (1992). Kluwer Academic Publishers, Boston.

Tesauro, G. (1995). Temporal Difference Learning and TD-Gammon. *Communications of the ACM (No. 3)*, Issue March 1995 Volume 38. Association for Computing Machinery.

Tesauro, G. (2002). Programming backgammon using self-teaching neural nets. *Artificial Intelligence* (pp. 181–199), Issue 134 (2002). Elsevier Science Publisher, B.V..

Temporal difference learning (2011). In *Wikipedia*. Wikipedia Inc. Retrieved on 25.5.2011 from http://en.wikipedia.org/wiki/Temporal_difference_learning

Goldberg, C. (2005). *StopWatch.java: Java Timer Class.* Goldb.org. Retrieved on 3.4.2011 from http://www.goldb.org/stopwatchjava.html

Yaakub Rashid (1981). *Revival of Congkak*. The Strait Times, National Library Board, Singapore. NewspaperSingapore. Retrieved on 3.4.2011 from http://newspapers.nl.sg/Digitised/Article/straitstimes19811006.2.130.6.1.aspx

How To Play Congkak (2010). In WikiHow: The How To Manual that You can Edit. Retrieved on 3.4.2011 from http://www.wikihow.com/Play-Congkak

Samuel, A. L. (1967). Some Studies in Machine Learning Using the Game of

Checkers. II – recent progress. *IBM Journal of research and development, 1967.*

Sutton, R. S., Barto, A. G. (2005). *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge. Retrieved on 22.3.2011 from http://neuro.bstu.by/ai/RL-3.pdf

StatSoft, Inc. (2011). Electronic Statistics Textbook. Tulsa, OK: StatSoft. WEB: http://www.statsoft.com/textbook/neural-networks/.

Lim,Y.J. (2007). *On Forward Pruning In Game Tree Search*. National University of Singapore, Singapore. Retrieved on 21.3.2011 from http://www.yewjin.com/storage/papers/PhDThesisLimYewJin.pdf

Lawrence, S., Giles, C. L., Tsoi, A.C. (1997). Lessons in Neural Network Training: Overfitting May be Harder than Expected. *Proceedings of the Fourteenth National Conference on Artificial Intelligence (pp. 540–545)*, Issue AAAI-97. AAAI Press, California.

Nyugen, D., Widrow, B. (1989). The Truck Backer Upper: An Example of Self Learning in Neural Network. In *International Joint Conference on Neural Network (IJCNN)*.

APPENDIX

(source-code is included in softcopy version of this report, and also was uploaded to http://congkaksystem.sourceforge.net/)