**An Exploration of Artificial Intelligence and Search Algorithms in Chinese Chess**

**Student Name:** Suki Tsz Ki Chan

**Course Title:** Bachelor of Science (Honours) in Computing with Games Development

**Supervisors:** Pat Doody, Robert Sheehy

**Year:** 2018

**Submission Date:** 04/05/2018

# **Abstract**

Artificial Intelligence is concerned with machines that learn and Machine Learning is concerned with the construction of computer programs that allows a machine to learn by examples and ultimately let machines teach themselves. Both Artificial Intelligence and Machine Learning has increasingly become a prominent research areas in the field of computer games development. Computer Chinese chess in particular has been identified as a significant area for Artificial Intelligence and Machine Learning research after Deep Blue’s win against world chess champion in 1997. The use of different search algorithms and problem-solving techniques has also become an important part of creating an effective artificial intelligence agent or learning machine for a computer game.

This thesis explored and researched the different types of algorithms, structures, techniques and methods used in creating a Computer Chinese Chess program. In this study, a computer Chinese chess game was created to evaluate whether artificial intelligence works in Chinese chess. It involved creating an Artificial Intelligence (narrow AI) that implemented a minimax search algorithm to determine the best possible moves for each game pieces.

Table of Contents

[**Abstract** 2](#_Toc511926550)

[**Chapter 1: Introduction** 6](#_Toc511926551)

[**1.1 Overview** 6](#_Toc511926552)

[**1.2 Objectives** 6](#_Toc511926553)

[**Chapter 2: Machine Learning** 7](#_Toc511926554)

[**2.1 Introduction** 7](#_Toc511926555)

[**2.2 Types of Machine Learning** 7](#_Toc511926556)

[**2.2.1 Supervised Learning** 7](#_Toc511926557)

[**2.2.2 Unsupervised Learning** 7](#_Toc511926558)

[**2.2.3 Reinforcement Learning** 8](#_Toc511926559)

[**2.3 Machine Learning methods** 8](#_Toc511926560)

[**2.3.1 Decision Tree Learning** 8](#_Toc511926561)

[**2.3.2 Artificial Neural Networks (ANN)** 8](#_Toc511926562)

[**2.3.3 Deep Learning (DL)** 9](#_Toc511926563)

[**2.3.4 Genetic Algorithms (GA)** 9](#_Toc511926564)

[**2.3.5 Temporal Difference (TD)** 9](#_Toc511926565)

[**2.4 Commercial Examples of Machine Learning** 10](#_Toc511926566)

[**Chapter 3: Artificial Intelligence** 11](#_Toc511926567)

[**3.1 Introduction** 11](#_Toc511926568)

[**3.2 Artificial Intelligence Approaches** 11](#_Toc511926569)

[**3.2.1 The Turing Test Approach** 11](#_Toc511926570)

[**3.2.2 The Cognitive Model Approach** 12](#_Toc511926571)

[**3.2.3 The Law of Thought Approach** 13](#_Toc511926572)

[**3.2.4 The Rational Agent Approach** 13](#_Toc511926573)

[**3.3 Artificial Intelligence in Games** 14](#_Toc511926574)

[**3.3.1 Decision Making** 14](#_Toc511926575)

[**3.3.2 Basic Perceptions** 14](#_Toc511926576)

[**3.3.3 Prediction** 14](#_Toc511926577)

[**3.4 Artificial Intelligence Accomplishments** 14](#_Toc511926578)

[**3.4.1 IBM 701 Electronic Data Processing Machine – The First Self-Learning Program** 14](#_Toc511926579)

[**3.4.2 TD-Gammon – The First Game-Learning Program** 14](#_Toc511926580)

[**3.4.3 Deep Blue - The First Chess Computer** 15](#_Toc511926581)

[**3.4.4 IBM Watson - Fast Computer Artificial Intelligence Software** 15](#_Toc511926582)

[**3.4.5 AlphaGo - The First Computer Go** 16](#_Toc511926583)

[**Chapter 4: Chinese Chess** 17](#_Toc511926584)

[**4.1 Introduction** 17](#_Toc511926585)

[**4.2 Chinese Chess Piece Names** 18](#_Toc511926586)

[**4.3 Chinese Chess Rules** 19](#_Toc511926587)

[**4.3.1 King** 19](#_Toc511926588)

[**4.3.2 Guard** 19](#_Toc511926589)

[**4.3.3 Bishop** 20](#_Toc511926590)

[**4.3.4 Knight** 20](#_Toc511926591)

[**4.3.5 Rook** 21](#_Toc511926592)

[**4.3.6 Cannon** 22](#_Toc511926593)

[**4.3.7 Pawn** 23](#_Toc511926594)

[**4.3.8 Other Rules** 23](#_Toc511926595)

[**4.2 Techniques used in modern Chess programs** 23](#_Toc511926596)

[**4.2.1 Game Tree** 24](#_Toc511926597)

[**4.2.1 Search Algorithm** 24](#_Toc511926598)

[**4.2.2 Minimax Algorithm** 24](#_Toc511926599)

[**4.2.3 Alpha-Beta Pruning (ABS)** 25](#_Toc511926600)

[**4.2.4 Evaluation Functions** 26](#_Toc511926601)

[**Chapter 5: Methodology and Design** 27](#_Toc511926602)

[**5.1 Review of research undertaken** 27](#_Toc511926603)

[**5.2 Research Question** 27](#_Toc511926604)

[**5.3 Methodology** 27](#_Toc511926605)

[**5.4 Vision Document** 28](#_Toc511926606)

[**5.4.1 Purpose** 28](#_Toc511926607)

[**5.4.2 Scope** 28](#_Toc511926608)

[**5.4.3 Stakeholder and User Description** 28](#_Toc511926609)

[**5.4.4 Requirements** 28](#_Toc511926610)

[**5.4.5 Risk Analysis** 29](#_Toc511926611)

[**5.4.6 Solutions (Tools and Platform)** 30](#_Toc511926612)

[**5.5 Functional Specification** 30](#_Toc511926613)

[**5.5.1 User Stories** 30](#_Toc511926614)

[**5.5.2 Project Plan** 31](#_Toc511926615)

[**5.6 Design** 32](#_Toc511926616)

[**5.6.1 System Use Case Diagram** 32](#_Toc511926617)

[**5.6.2 Activity Diagram** 42](#_Toc511926618)

[**5.6.3 Key Class Diagram** 43](#_Toc511926619)

[**5.6.4 VOPC** 43](#_Toc511926620)

[**5.6.5 Sequence Diagrams** 43](#_Toc511926621)

[**Chapter 6: Implementation** 45](#_Toc511926622)

[**6.1 Prototype and Sprints** 45](#_Toc511926623)

[**6.1.1 Sprint 1** 45](#_Toc511926624)

[**6.1.2 Sprint 2** 49](#_Toc511926625)

[**6.1.3 Sprint 3** 51](#_Toc511926626)

[**6.1.4 Sprint 4** 54](#_Toc511926627)

[**6.1.5 Sprint 5** 59](#_Toc511926628)

[**Chapter 7: Findings and Conclusion** 60](#_Toc511926629)

[References 61](#_Toc511926630)

# **Chapter 1: Introduction**

## **1.1 Overview**

Both Artificial Intelligence and Machine Learning concerned with creating effective learning machines that can perform complex tasks. . In particular, Computer Chinese chess has become an important field for Artificial Intelligence and Machine Learning research after Deep Blue’s win against world chess champion Garry Kasparov in 1997. Many researchers focuses on using effective search algorithms and problem-solving techniques to implement an Artificial Intelligence agent and learning machines that is able to learn and analyse all possible moves as quickly as possible.

The objectives of this thesis is to explore and research the different types of methods, techniques and algorithms used to create an artificial intelligence Chinese chess program.

## **1.2 Objectives**

This study involved looking at creating an Artificial Intelligence agent that implemented a minimax search algorithm that can determine the best possible moves for each game pieces.

# **Chapter 2: Machine Learning**

## **2.1 Introduction**

Machine learning (ML), a subfield of Artificial Intelligence (AI), is associated with the construction of computer programs that automatically improve with experience (Mitchell, 1997). It is about making computers modify or adapt their actions so that these actions get more accurate. The aim of machine learning is to establish learning algorithm, procedures that allow a machine to learn from examples presented to it and to ultimately let machines teach themselves (Bengio, 2016). The concept of a computer machine and intelligence began in the 1950 when Alan Turing questioned “Can machine think?” (Turing, 1950). Machine learning has led to the development of machines that can learn intelligent behaviour directly from data rather than being explicitly programmed to display such behaviour (Schölkopf, 2015).

Machine learning became a prominent research area since the 1980s, with a purpose of giving computers the ability to learn and build models so that they could perform activities like prediction within specific domains (Jones, 2017). Machine learning methods are sometimes called subsymbolic because no symbols or symbolic manipulation are involved (Marsland, 2015).

## **2.2 Types of Machine Learning**

### **2.2.1 Supervised Learning**

Supervised learning is the most widely used technique in machine learning. In supervised learning, systems are trained to infer pattern from observational data. A particularly simple type of pattern, a mapping between input and output can be learned through supervised learning. It involves given training data which consists of example inputs and the corresponding outputs and comes up with a model to explain those data (Schölkopf, 2015). Decision tree learning is a type of supervised learning algorithm.

### **2.2.2 Unsupervised Learning**

The goal of unsupervised learning is to identify and explore regularities and dependencies in data. Similar to supervised learning, unsupervised learning proceeds from a finite sample of training data, meaning that the learned concepts are stochastic variables depending on the particular training set (Hansen & Larsen, 1996).

### **2.2.3 Reinforcement Learning**

In reinforcement learning, the training information provided to the learning system by the environment is in the form of scalar reinforcement signals that measure how well the system operates. The learner is not told which actions to take but and must discover which actions to be the best by trying each action in turn (Maglogiannis, et al., 2007). IBM researcher Gerald Tesauro developed a backgammon player called TD-Gammon by using reinforcement learning (Tesauro, 1995).

## **2.3 Machine Learning methods**

### **2.3.1 Decision Tree Learning**

**Decision Tree Learning** is a logic-based type of supervised learning algorithm. It is a tree that classify instances by sorting them based on feature values. Each node in a tree represents a feature in an instance to be classified, and each branch represents a value that a node can assume. Instances are classified starting at the root node and sorted based on their feature values (Maglogiannis, et al., 2007).

### **2.3.2 Artificial Neural Networks (ANN)**

**Artificial Neural Networks (ANN)** is a perception-based type of supervised learning algorithm from biology, designed to simulate the way in which the human brain processes information. ANNs gather their knowledge by detecting the patterns and relationships in data (Agatonovic-Kustrin & Beresford, 2000). The basic unit of ANN is neuron. An artificial neuron corresponds to a nonlinear threshold apparatus with multiples inputs and a single output (Wang & Li, 2008).



Fig 2.1: Structure of an Artificial neural networks (ANN) (Agatonovic-Kustrin & Beresford, 2000)

### **2.3.3 Deep Learning (DL)**

**Deep Learning** is a form of machine learning that allows computers to learn from experience. It refers to the simulation of networks of neurons that gradually learn to recognised images, understand speech or even make decisions on their own (Bengio, 2016). Deep Learning is a family of algorithms that implement deep networks with unsupervised learning (Jones, 2017).

### **2.3.4 Genetic Algorithms (GA)**

A **Genetic Algorithm** is a global optimization algorithm that introduces the idea of biology genetics, enhancing the adaptability of each individual by the genetic operation mechanism such as selection, crossover, thus simulating the evolution process of natural selection (Wang & Li, 2008). It is a probabilistic search procedures designed to work on large spaces involving states that can be represented by strings. The methods used in Genetic Algorithm are inherently parallel, using distributed set of samples to generate a new set of samples (Goldberg & Holland, 1988). The aim of Genetic Algorithm is to obtain the approximate optimal solution from all feasible solutions under genetic operators (Li , et al., 2016).

### **2.3.5 Temporal Difference (TD)**

**Temporal Difference** learning method is a prediction method within reinforcement learning that introduce the idea that predictions can be learned from observations in an environment (Jones, 2017). The goal of learning in Temporal Difference method is to make the learner’s current prediction for the current input pattern more closely match the next prediction at the next time step (Tesauro, 1995).

## **2.4 Commercial Examples of Machine Learning**

* **Siri:** A voice recognition system by Apple Inc. Siri uses machine learning to help users answer questions and make recommendations.
* **Facebook:** A social media service that uses image recognition algorithm to recognised people in photo.
* **AVG:** An antivirus software that uses machine learning to detect malicious software on computer device.
* **Google Search:** A search engine that uses machine learning algorithm to improve search results and search suggestions.
* **Gmail:** A popular email service developed by Google. Gmail uses machine leaning to allow the inbox to respond to emails on behalf of the account user, i.e. Out of office reply.
* **PayPal:** An online payment platform that uses machine learning algorithm to detect fraud.
* **Spotify:** A popular digital music service that uses machine learning to keep track of user’s music taste and will provide the user with a list of related tracks.
* **UBER:** A global transportation technology company that uses machine learning algorithms to determine arrival times and pick up locations.

# **Chapter 3: Artificial Intelligence**

## **3.1 Introduction**

**Artificial Intelligence (AI)**, also known as **Machine Intelligence (MI)**, is the area of computer science dedicated to produce software capable of sophisticated, intelligent, computations similar to those that the human brain routinely performs (Agatonovic-Kustrin & Beresford, 2000). Artificial Intelligence study began in the 1950s with a focus on thinking machines and machines that could generally perform any intellectual task that a human could (Jones, 2017). Artificial Intelligences

There are four approaches to define artificial intelligence system.

|  |  |
| --- | --- |
| Systems that think like humans | Systems that think rationally |
| Systems that act like humans | Systems that act rationally |

Fig 3.1 Approaches that define Artificial Intelligent systems (Russell & Norvig, 2010)

## **3.2 Artificial Intelligence Approaches**

### **3.2.1 The Turing Test Approach**

Alan Turing proposed The Turing Test (Turing, 1950) which provided a satisfactory operational definition of intelligence. He stated to test a machine’s ability to exhibit intelligent behaviour, it would need to possess the following capabilities:

* **Natural Language** Processing to enable it to communicate in English.
* **Knowledge Representation** to store what it knows or hears.
* **Automated Reasoning** to use the stored information to answer questions and to draw new conclusions.
* **Machine Learning** to adapt to new circumstances and to detect and extrapolate pattern.
* **Computer Vision** to perceive objects.
* **Robotics** to manipulate and move objects.

(Russell & Norvig, 2010)

In the Turing test (also known as The Imitation Game), it involves a remote human interrogator, and a given fixed time frame, who have to distinguish between a computer and a human subject based on their replies to various question posed by the interrogator (Curley, 2011).

#### **3.2.1.1 Impact and Criticism of the Turing Test and Eugene Goostman (the first artificial intelligence that passed the Turing Test)**

There has been mixed reaction of the Turing Test approach. Some researchers claim that the Turing Test acts as the operational definition of intelligence and thinking, while some claim the test fails to provide a real impediment to progress in the field of artificial intelligence (French, 2000). There has been criticism whether the Turing Test is a valid test for intelligence, especially with the victory of Eugene Gootsman at the Turing Test event in 2014. Eugene Goostman, is a computer program that simulates a 13 year old boy, has been said to pass the Turing Test for the first time in 65 years. The program won the event because ten out of thirty human judges believed they were speaking to a real teenage boy during a five minute conversation (Edgar, 2014). However, some experts claims that the program did not really pass the Turing Test and questioned whether the Turing Test is an important indicator of machine intelligence (Vardi, 2014).

### **3.2.2 The Cognitive Model Approach**

This is the interdisciplinary field of study that combines computer model artificial intelligence and psychology techniques to construct precise and testable theories of the human mind (Russell & Norvig, 2010). Cognitive computing uses neural networks and deep learning and then applies knowledge from cognitive science to build systems that simulate human thought processes. It uses several disciplines such as machine learning, natural language processing, vision and human computer interaction (Jones, 2017).

#### **3.2.2.1 IBM Watson**

IBM Watson is an example of cognitive system. It involves deep content analysis and evidence-based reasoning to help speed up and improve decisions. Watson uses a set of transformational technologies which incorporates natural language, hypothesis generation and evidence-based learning (IBM, 2017). Watson has extended into other area through a set of web services such as visual recognition, speech to text/text to speech function and language understanding and translation (Jones, 2017), and to other fields such as healthcare (helps identify cancer treatments), education, finance and retail. The aim of Watson is to “help each one of us by augmenting our intelligence to help us do our daily tasks, make the right decisions, discover the right kinds of information and eventually get the right outcome that we all desiring” (Banavar, 2016).

#### **3.2.2.2 Guruduth Banavar: Cognitive Systems**

Dr Guruduth Banavar, Vice President of Cognitive Computing at IBM Research, is responsible for creating the next generation of cognitive systems in the family of IBM Watson with the aim to combine human intelligence with machine intelligence to help answer big questions such as “Why does cancer exist?” and “Can we reverse global warming?” (Banavar, 2016). Another objective of cognitive computing is to analyse and garner insights from that massive amounts of data in order to create computer systems (platforms) that will transform a number of industries.

The essence of cognitive means cognitive systems must learn at scale.

* Learning at scale in the data
* Learning at scale for your solution/business
* Reasoning with a goal to take an action

(Trice, 2015)

### **3.2.3 The Law of Thought Approach**

This approach uses logic to solve any solvable problem described in logical notation. It emphasis on correct inferences. (Russell & Norvig, 2010)

### **3.2.4 The Rational Agent Approach**

The idea of this approach is to create a rational agent able to operate autonomously, adapt to changes, create and pursue goals and acts so that it can achieve the best outcome. (Russell & Norvig, 2010)

## **3.3 Artificial Intelligence in Games**

Artificial Intelligence can play multiple different roles in gaming. The goals of Artificial Intelligence in games is to simulate intelligence behaviour, providing the player with a challenge, a challenge that the player can overcome.

### **3.3.1 Decision Making**

This is the core concept of Artificial Intelligence. To be able to execute these choice, the system needs to be able to affect the entities using the AI system. When the AI makes a decision, that decision is then broadcast to the entities involved. This approach works well in real-time strategy games, where AI is concerned with the big picture. (Kehoe, 2015)

### **3.3.2 Basic Perceptions**

To allow AI to make meaningful decision, it needs to be able to perceive its environment. In a simple system, this perception can be as simple as checking the position of the player entity. However, as the system become more demanding, entities need to identify key features of the game world and it is up to the designer and developer to come up with a way to identify these key features important to the system. (Kehoe, 2015)

### **3.3.3 Prediction**

This is the ability to effectively anticipate an opponent’s next move. A basic method of this is to allow the system to keep track of past decisions and evaluate their success. The data gathered can be evaluated, allowing the system to determine the success of previous actions and to decide whether a change in tactics is required. (Kehoe, 2015)

## **3.4 Artificial Intelligence Accomplishments**

### **3.4.1 IBM 701 Electronic Data Processing Machine – The First Self-Learning Program**

Built by Arthur Samuel in 1959, IBM 701 was also known as the Defense Calculator. It implements an optimization to search trees called alpha-beta pruning. The program recorded reward for specific move, allowing the application to learn with each game played. Samuel also programmed it to play itself, increasing the rate the program learn and play (Jones, 2017).

### **3.4.2 TD-Gammon – The First Game-Learning Program**

Developed by Gerald Tesauro in 1992, TD-Gammon program is an artificial neural network that uses reinforcement learning technique to train itself to be an evaluation function for the game of backgammon by playing against itself and learning from outcome. It also uses Temporal Difference learning methods to make the learner’s current prediction for the current input pattern more closely match the next prediction at the next time step. TD-Gammon learned the game of backgammon without knowledge of the game, building its expertise through self-game (Jones, 2017). TD-Gammon was designed as a way to explore the capability of multilayer neural networks trained by TD (lambda) to learn complex nonlinear functions (Tesauro, 1995). This achievement has been highly influential in the Artificial Intelligence and computer gaming communities, and has inspired numerous real-world application of similar reinforcement learning techniques (IBM Research, 2017).

### **3.4.3 Deep Blue - The First Chess Computer**

Deep Blue, developed by IBM, was the first chess-playing supercomputer that beat against world chess champion Garry Kasparov after a six game match in 1997 (Hsu, 1999). There were number of factors that contributed to this success:

* A single-chip chess search engine
* A massively parallel system with multiple levels of parallelism
* A strong emphasis on search extensions
* A complex evaluation function
* Effective use of a Grandmaster game database

(Campbell, et al., 2002)

Deep Blue had a significant impact in the area of computing across many different industries. It was programmed to solve the complex strategic game of chess, thus allowing researchers to explore and understand the limits of massively parallel processing. This research gave developers deeper understanding into ways they could design a computer to tackle complex problems in other fields, using deep knowledge to analyse a higher number of possible solutions (IBM, 2011).

### **3.4.4 IBM Watson - Fast Computer Artificial Intelligence Software**

IBM Watson, developed by IBM in 2010, is the first computer competed on the TV quiz show “Jeopardy!”. It won against the show’s two greatest champions. IBM Watson is the first cognitive system that was implemented using DeepQA, which allows the computer to search for answers in unstructured data more effectively than standard search technology (IBM 100, 2011). It also uses machine learning, statistical analysis and natural language processing to find and understand the clues in the questions, then compared possible answers, by ranking its confidence in their accuracy and responded in three seconds (IBM Research, 2014).

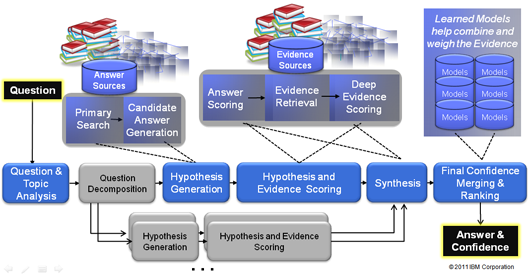


Fig 3.2 DeepQA philosophy in IBM Watson (IBM Research, 2011)

### **3.4.5 AlphaGo - The First Computer Go**

AlphaGo, developed by Google DeepMind, is the first Computer Go program to defeat a professional human Go player in 2016. This was considered to be a significant milestone in the quest of artificial intelligence (AI). AlphaGo uses combination of advanced search tree with deep neural networks. These neural networks record the description of the Go board as an input and process it through a number of different network layers containing millions of neuron-like connections. One neural network, the policy network, selects the next move to play while another neural network, the value network, predicts the winner of the game (DeepMind Technologies Limited, n.d.). These deep neural networks are trained by combination of supervised learning from human expert games, and reinforcement learning from games of self-play (Silver, et al., 2016).

# **Chapter 4: Chinese Chess**

## **4.1 Introduction**

Chinese Chess also known as Xiang Qi (Xiang means elephant and Qi means chess) or The Elephant Game, is a popular two player strategy board game in China. Similar to Chess, the aim of the game is to capture the opponent’s General piece or putting the opponent’s General piece in checkmate to win. The Chinese chess is played on a 9x10 board (Fig 4.1), the board contains a river in the middle that divides the board between two sides. Each side also contains a 3x3 intersection at its centre with diagonal lines which represent the imperial palace. Chinese Chess is played with 32 pieces, each player has 16 pieces (one King, two Guards, two Bishops, two Rooks, two Knights, two Cannons and five Pawns).

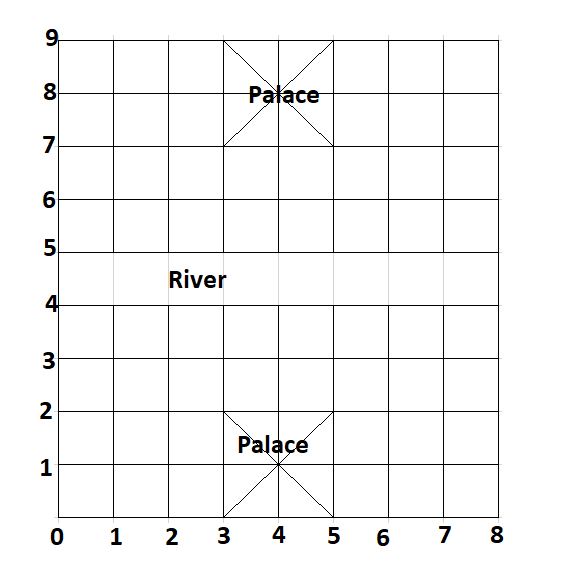


Fig 4.1 Chinese Chess Board

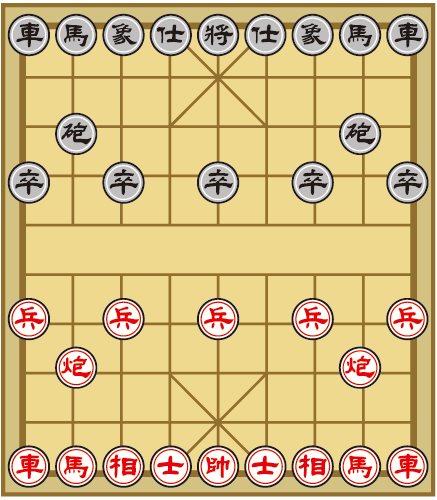


Fig 4.2 Chinese Chess Board with Pieces set up (Yellow Mountain Imports, 2010)

## **4.2 Chinese Chess Piece Names**

|  |  |
| --- | --- |
| Piece Name in Chinese | Piece Name in English |
| 帥 | King |
| 士 | Guard |
| 相 | Bishop |
| 車 | Rook |
| 馬 | Knight |
| 炮 | Cannon |
| 兵 | Pawn |

## **4.3 Chinese Chess Rules**

### **4.3.1 King**

* King moves one space vertically or horizontally only
* It cannot leave its palace.
* King is not allowed to make a move to a position which is being attacked by an enemy piece.
* If a king is being attacked (is in check), the corresponding player must cancel the check immediately. If it is not possible, the player loses the game.

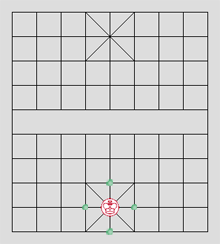


Fig 4.3 King possible moves (BrainKing, 2005)

### **4.3.2 Guard**

* A guard moves one space diagonally
* They cannot leave its palace.

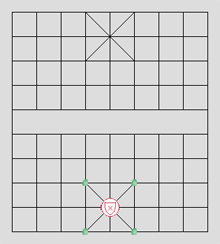


Fig 4.4 Guard possible moves (BrainKing, 2005)

### **4.3.3 Bishop**

* Bishop moves exactly two points in any diagonal direction
* They must stay on their half of the board.
* Bishop cannot jump, its movement is blocked if there is a piece on the intervening point.

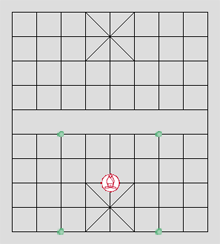


Fig 4.5 Bishop possible moves (BrainKing, 2005)

### **4.3.4 Knight**

* Knight can move one point in any direction horizontally or vertically, plus one diagonal move.
* If the first point of the horizontal or vertical move is blocked by a piece, then the Knight may not move in that direction
* Knight cannot jump over occupied places
* Knight in the Fig 4.7 cannot move to the points marked by the red X’s; it is blocked by the Black pawn

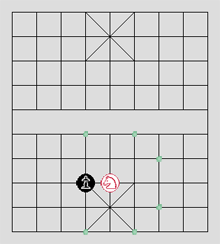


Fig 4.6 Knight possible moves (BrainKing, 2005)

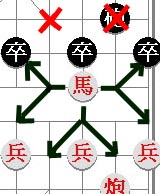


Fig 4.7 Knight possible moves (AncientChess.Com, 2007)

### **4.3.5 Rook**

* Moves any number of spaces vertically or horizontally until it meets another piece or the edge of the board.

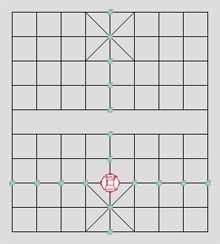


Fig 4.8 Rook possible moves (BrainKing, 2005)

### **4.3.6 Cannon**

* A cannon moves in the same way as a rook.
* If a cannon wants to capture an opponent's piece, it must be done by hoping over exactly one another piece (own or opponent's). Fig 4.9 shows a position where all possible red cannon moves (or captures) marked by green dots (note a green dot inside the black rook at the board top).
* The black knight on the left side cannot be captured because no third piece stands between it and the cannon. The top black rook can be captured since the cannon hops over the red pawn. On the other hand, the second black rook (on the right) cannot be captured since the cannon would hop over more than one piece (black pawn and red knight).

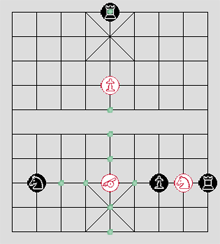


Fig 4.9 Cannon possible moves (BrainKing, 2005)

### **4.3.7 Pawn**

* Move only one space forward only (they never go diagonally and cannot move backwards).
* When a pawn crosses the river, its moving abilities extend by sideways options - it can move one space forward or horizontally, cannot move backwards
* Pawns don't promote in Chinese chess. When a pawn reaches the last row, it can continue moving only sideways.

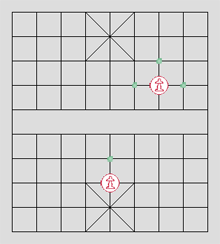


Fig 4.10 Pawn possible moves (BrainKing, 2005)

### **4.3.8 Other Rules**

* Unlike International Chess, stalemate is not a draw in Chinese Chess. Rather, stalemate is a win for the side who has stalemated the other.

## **4.2 Techniques used in modern Chess programs**

The structure of a chess program is based on the following.

* **Representation:** This refers to the way in which the board, the pieces and the movements of the pieces are described numerically so they can be used appropriately by the search and evaluation algorithms.
* **Search for moves:** This is done by using algorithm that searches for an adequate move that retains a list of all the legal movements.
* **Position Evaluation:** The evaluation module is called on continuously by the search algorithm to pass on the scores of each position.

(Rasskin-Gutman, 2009)

### **4.2.1 Game Tree**

Computer chess games are usually represented using game trees. A game tree is an instance of a tree in which the root node represent possible states of the game or positions, and arcs in the tree represent moves. Leaf nodes in the tree represent final states, where the game has been won, lost or tied (Coppin, 2004).

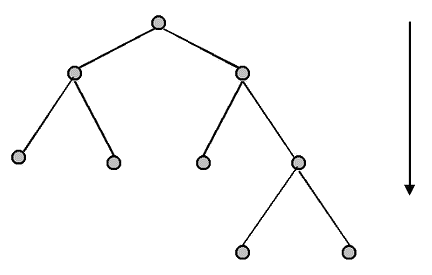


Fig 4.11 Example of a Game Tree (Ross, 2014)

### **4.2.1 Search Algorithm**

A search algorithm works by considering various possible action sequences. The possible action sequences starts at the initial state from a search tree with the initial state at the root. The branches are actions and the nodes correspond to states in the state space of the problem (Russell & Norvig, 2010).

### **4.2.2 Minimax Algorithm**

The minimax algorithm is the most widely used game tree search algorithm. The aim of the algorithm is to maximise the lowest possible score that can be achieved. When evaluating game trees, it is usual to assume that the computer is attempting to maximise some score that the opponent is trying to minimise. This score is usually the result of the evaluation function for a given position, so high positive score means a good position for the computer and a high negative score means a good position for the opponent (Coppin, 2004).

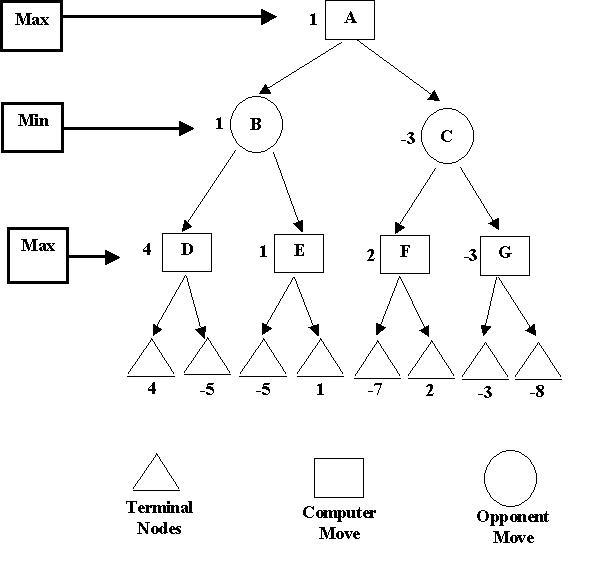


Fig 4.12 Example of Game Tree using Minimax Algorithm (Kendall, 2001)

In Fig 4.12, the computer starts from the ends of the branches and works the numbers back up the tree. At each stage, the computer assumes its opponent will choose their best move, which would be the **minimum** (the worst one) for the computer. When the computer looks at its possible moves in the game tree, it always chooses the best one, which is represented by the **maximum** number.

### **4.2.3 Alpha-Beta Pruning (ABS)**

Alpha-Beta Pruning is an adversarial search algorithm which helps to decrease the complexity of nodes in the game tree thus allowing an efficient search to be performed. It allows the computer to ignore or prune branches of the search tree that would give least favourable results (Computer History Museum, 2005) and computing the correct minimax decision without looking at every node in the game tree (Russell & Norvig, 2010).

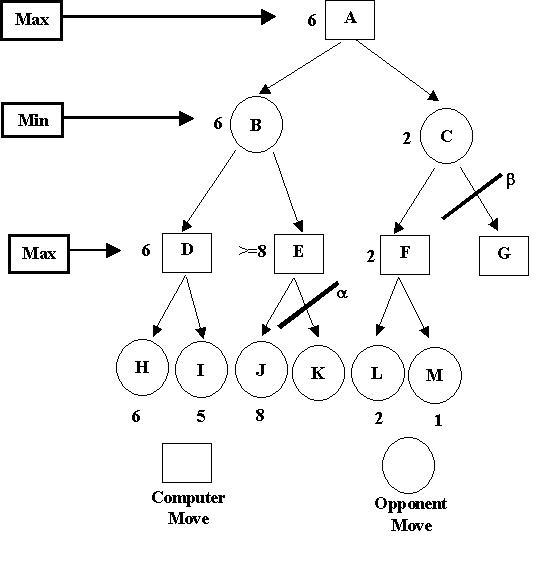


Fig 4.13 Example of Game Tree using Alpha-Beta Pruning (Kendall, 2001)

### **4.2.4 Evaluation Functions**

Evaluation Functions is used to examine a particular position of the board and estimate how well the computer is doing, or how likely it is to win from this position. Evaluation Functions is also known as Static Evaluators because they are used to evaluate a game from just one static position (Coppin, 2004).

# **Chapter 5: Methodology and Design**

## **5.1 Review of research undertaken**

The research undertaken shows how significant Artificial Intelligence and Machine Learning are in computer games. It also showed the structure, techniques and algorithms used to create a computer Chinese Chess program. Like standard computer chess programs, computer Chinese Chess programs uses a game tree to represent the state of the game or a particular position. A game tree is a tree structure with nodes and edges. A node represents the position and the edges represents the moves. When determining possible moves or position, a searching algorithm of the game tree is used to determine the best possible move. The minimax algorithm is most commonly used algorithm in games which requires decision making or determining the best move. However, due to the complexity of Chinese chess game (the complexity of Chinese chess is between Standard Chess and Go), it is quite often impossible to search the whole game tree. This is where other techniques such as the alpha-beta pruning may be implemented to help reduce the searching complexity and to allow a more efficient searching of the game tree.

## **5.2 Research Question**

An Exploration of Artificial Intelligence and Search Algorithm in Chinese Chess. This study involved developing computer Chinese Chess program using minimax search algorithm. The aim was to create an artificial intelligence that uses the minimax search algorithm to determine best possible move.

## **5.3 Methodology**

The context of this study was to develop a computer Chinese Chess game using the minimax algorithm. It involved the following:

* Created the Game board as a 2D array
* Listed out possible legal movements for each chess pieces
* Set up each pieces on game board
* Prepared game tree
* Applied minimax algorithm and alpha-beta pruning technique to determine the best possible moves.

## **5.4 Vision Document**

### **5.4.1 Purpose**

The purpose of this project is to create and develop a computer Chinese Chess game using Unity game engine.

### **5.4.2 Scope**

To create an Artificial Intelligence that can play the game and to be able to determine the next possible and capturing moves.

### **5.4.3 Stakeholder and User Description**

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Responsibilities** |
| Game Board | Game board for the game | To allow chess pieces to be placed on and move. |
| King | Game Piece for the game | To be use in the game |
| Guard | Game Piece for the game | To be use in the game |
| Bishop | Game Piece for the game | To be use in the game |
| Knight | Game Piece for the game | To be use in the game |
| Rook | Game Piece for the game | To be use in the game |
| Cannon | Game Piece for the game | To be use in the game |
| Pawn | Game Piece for the game | To be use in the game |

### **5.4.4 Requirements**

Using the **MoSCoW** method.

|  |  |  |
| --- | --- | --- |
| **Item ID** | **Description** | **M = Must Have, S = Should Have, C = Could Have, W = Won’t Have** |
| 000 | Game board creation | M |
| 001 | Possible movements and capturing moves for King piece | M |
| 002 | Possible movements and capturing moves for Guard piece | M |
| 003 | Possible movements and capturing moves for Bishop piece | M |
| 004 | Possible movements and capturing moves for Knight piece | M |
| 005 | Possible movements and capturing moves for Rook piece | M |
| 006 | Possible movements and capturing moves for Cannon piece | M |
| 007 | Possible movements and capturing moves for Pawn piece | M |
| 008 | Pieces set up at start of game | M |
| 009 | Exceptions handling | S |
| 010 | Game menu | C |
| 011 | GUI | C |
| 012 | Multiplayer | W |
| 013 | Game Difficulties | W |

### **5.4.5 Risk Analysis**

|  |  |  |
| --- | --- | --- |
| **Risk** | **Risk Description** | **Actions** |
| Game Rules | Knowledge on the game rules | Research on game rules |

### **5.4.6 Solutions (Tools and Platform)**

|  |  |
| --- | --- |
| Game Engine | Unity3D |
| IDE | Visual Studio |
| Coding Language | C# |
| Source Control | GitHub |
| Sprite Editor Software |  |

## **5.5 Functional Specification**

### **5.5.1 User Stories**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **User** | **Description** | **Priority**  **H – High**  **M – Medium**  **L - Low** | **Estimate (Hours)** | **Acceptance Criteria** |
| 000 | Game Board | As a game board, I want to create a game board for the game | H | 6 | Have a game board for the game pieces |
| 001 | King | As a game piece, I want to be able to move around the board to capture other game pieces | H | 8 | List of possible moves and capturing moves defined |
| 002 | Guard | As a game piece, I want to be able to move around the board to capture other game pieces | H | 8 | List of possible moves and capturing moves defined |
| 003 | Bishop | As a game piece, I want to be able to move around the board to capture other game pieces | H | 8 | List of possible moves and capturing moves defined |
| 004 | Knight | As a game piece, I want to be able to move around the board to capture other game pieces | H | 8 | List of possible moves and capturing moves defined |
| 005 | Rook | As a game piece, I want to be able to move around the board to capture other game pieces | H | 8 | List of possible moves and capturing moves defined |
| 006 | Cannon | As a game piece, I want to be able to move around the board to capture other game pieces | H | 8 | List of possible moves and capturing moves defined |
| 007 | Pawn | As a game piece, I want to be able to move around the board to capture other game pieces | H | 8 | List of possible moves and capturing moves defined |

### **5.5.2 Project Plan**

|  |  |  |
| --- | --- | --- |
| **Week** | **Description** | **Estimate (Hours)** |
| 27/11/17-03/12/17 | * Game Piece Script (Pawn), listing out all possible moves * Continue with updating FYP document (Methodology and Design part) | 8 |
| 04/12/17-13/12/17 | * Start other game piece scripts, listing out all possible moves for each game pieces * Finish document (Research, Methodology and Design) * Research Search Tree | 8 |
| 14/12/17-January 2018 | * Start other game piece scripts, listing out all possible moves for each game pieces * Research Search Tree | 8 |

## **5.6 Design**

### **5.6.1 System Use Case Diagram**

#### **5.6.1.1 Create Game Board Use Case**

**Use Case Diagram*:***

Game Board

**Use Case Narrative**

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | Create Game Board | |
| **Use Case Id** | 001 | |
| **Priority** | H | |
| **Source** | 5.5.1 User Story ID 000 | |
| **Primary Business Actor** | User, Game Board | |
| **Other Participating Actors** |  | |
| **Description** | To set up game board | |
| **Preconditions** |  | |
| **Trigger** | Game Start | |
| **Typical Scenario** | **Actor Action** | **System Response** |
| **Create Board** | Step 1: User invokes game start option | Step 2: create game board |
| **Alternate Scenarios** | **Actor Action** | **System Response** |
|  |  |  |
| **Conclusions** | Game board is created and display on screen | |
| **Post conditions** | Games pieces will be display on their respective location | |
| **Business Rules** |  | |
| **Implementation Constraints** |  | |

#### **5.6.1.2 King Movement Use Case**

**Use Case Diagram*:***

King

**Use Case Narrative**

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | **King Movement** | |
| **Use Case Id** | 002 | |
| **Priority** | H | |
| **Source** | 5.5.1 User Story ID 001 | |
| **Primary Business Actor** | King | |
| **Other Participating Actors** | Game Board | |
| **Description** | Moving the King piece | |
| **Preconditions** | Game start | |
| **Trigger** |  | |
| **Typical Scenario** | **Actor Action** | **System Response** |
| **Move** | Step 1: King piece moves to a location on the board | Step 2: update King piece new location, display King piece in new location on the board |
| **Alternate Scenarios** | **Actor Action** | **System Response** |
| **Invalid Location** |  | Step 2: System identifies that the King piece is moving to an invalid location.  Step 3: Display appropriate message and King piece must select a different location to move to. |
| **Conclusions** | King piece new location on the board has been updated | |
| **Post conditions** |  | |
| **Business Rules** | King piece cannot move outside of palace | |
| **Implementation Constraints** |  | |

#### **5.6.1.3 Guard Movement Use Case**

**Use Case Diagram*:***

Guard

**Use Case Narrative**

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | **Guard Movement** | |
| **Use Case Id** | 003 | |
| **Priority** | H | |
| **Source** | 5.5.1 User Story ID 002 | |
| **Primary Business Actor** | Guard | |
| **Other Participating Actors** | Game Board | |
| **Description** | Moving the Guard piece | |
| **Preconditions** | Game start | |
| **Trigger** |  | |
| **Typical Scenario** | **Actor Action** | **System Response** |
| **Move** | Step 1: Guard piece moves to a location on the board | Step 2: update Guard piece new location, display Guard piece in new location on the board |
| **Alternate Scenarios** | **Actor Action** | **System Response** |
| **Invalid Location** |  | Step 2: System identifies that the Guard piece is moving to an invalid location.  Step 3: Display appropriate message and Guard piece must select a different location to move to. |
| **Conclusions** | Guard piece new location on the board has been updated | |
| **Post conditions** |  | |
| **Business Rules** | Guard piece cannot move outside of palace | |
| **Implementation Constraints** |  | |

#### **5.6.1.4 Bishop Movement Use Case**

**Use Case Diagram*:***

Bishop

**Use Case Narrative**

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | **Bishop Movement** | |
| **Use Case Id** | 004 | |
| **Priority** | H | |
| **Source** | 5.5.1 User Story ID 003 | |
| **Primary Business Actor** | Bishop | |
| **Other Participating Actors** | Game Board | |
| **Description** | Moving the Bishop piece | |
| **Preconditions** | Game start | |
| **Trigger** |  | |
| **Typical Scenario** | **Actor Action** | **System Response** |
| **Move** | Step 1: Bishop moves to a location on the board | Step 2: update Bishop piece new location, display Bishop piece in new location on the board |
| **Alternate Scenarios** | **Actor Action** | **System Response** |
| **Invalid Location** |  | Step 2: System identifies that the Bishop piece is moving to an invalid location.  Step 3: Display appropriate message and Bishop piece must select a different location to move to. |
| **Conclusions** | Bishop piece new location on the board has been updated | |
| **Post conditions** |  | |
| **Business Rules** | Bishop pieces are not allowed to cross the river, they must stay on their half of the board | |
| **Implementation Constraints** |  | |

#### **5.6.1.5 Knight Movement Use Case**

**Use Case Diagram*:***

Knight

**Use Case Narrative**

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | **Create Game Board** | |
| **Use Case Id** | 005 | |
| **Priority** | H | |
| **Source** | 5.5.1 User Story ID 004 | |
| **Primary Business Actor** | Knight | |
| **Other Participating Actors** | Game Board | |
| **Description** | Moving the Knight piece | |
| **Preconditions** | Game Start | |
| **Trigger** |  | |
| **Typical Scenario** | **Actor Action** | **System Response** |
| **Move** | Step 1: Knight piece moves to a location on the board | Step 2: update new location, display Knight piece in new location on the board |
| **Alternate Scenarios** | **Actor Action** | **System Response** |
| **Invalid Location** |  | Step 2: System identifies that the Knight piece is moving to an invalid location.  Step 3: Display appropriate message and Knight piece must select a different location to move to. |
| **Conclusions** | Knight piece new location on the board has been updated | |
| **Post conditions** |  | |
| **Business Rules** |  | |
| **Implementation Constraints** |  | |

#### **5.6.1.6 Rook Movement Use Case**

**Use Case Diagram*:***

Rook

**Use Case Narrative**

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | **Rook Movement** | |
| **Use Case Id** | 006 | |
| **Priority** | H | |
| **Source** | 5.5.1 User Story ID 005 | |
| **Primary Business Actor** | Rook | |
| **Other Participating Actors** | Game Board | |
| **Description** | Moving the Rook piece | |
| **Preconditions** |  | |
| **Trigger** | Game Start | |
| **Typical Scenario** | **Actor Action** | **System Response** |
| **Move** | Step 1: Rook piece moves to a location on the board | Step 2: update Rook piece new location, display Rook piece in new location on the board |
| **Alternate Scenarios** | **Actor Action** | **System Response** |
| **Invalid Location** |  | Step 2: System identifies that the Rook piece is moving to an invalid location.  Step 3: Display appropriate message and Rook piece must select a different location to move to. |
| **Conclusions** | Rook piece new location on the board has been updated | |
| **Post conditions** |  | |
| **Business Rules** |  | |
| **Implementation Constraints** |  | |

#### **5.6.1.7 Cannon Movement Use Case**

**Use Case Diagram*:***

Cannon

**Use Case Narrative**

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | **Cannon Movement** | |
| **Use Case Id** | 007 | |
| **Priority** | H | |
| **Source** | 5.5.1 User Story ID 006 | |
| **Primary Business Actor** | Cannon | |
| **Other Participating Actors** | Game Board | |
| **Description** | Moving the Cannon piece | |
| **Preconditions** |  | |
| **Trigger** | Game Start | |
| **Typical Scenario** | **Actor Action** | **System Response** |
| **Move** | Step 1: Cannon piece moves to a location on the board | Step 2: update Cannon piece new location, display Cannon piece in new location on the board |
| **Alternate Scenarios** | **Actor Action** | **System Response** |
| **Invalid Location** |  | Step 2: System identifies that the Cannon piece is moving to an invalid location.  Step 3: Display appropriate message and Cannon piece must select a different location to move to. |
| **Conclusions** | Cannon piece new location on the board has been updated | |
| **Post conditions** |  | |
| **Business Rules** |  | |
| **Implementation Constraints** |  | |

#### **5.6.1.8 Pawn Movement Use Case**

**Use Case Diagram*:***

Pawn

**Use Case Narrative**

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | **Create Game Board** | |
| **Use Case Id** | 008 | |
| **Priority** | H | |
| **Source** | 5.5.1 User Story ID 007 | |
| **Primary Business Actor** | Pawn | |
| **Other Participating Actors** | Game Board | |
| **Description** | Moving the Pawn piece | |
| **Preconditions** |  | |
| **Trigger** | Game Start | |
| **Typical Scenario** | **Actor Action** | **System Response** |
| **Move** | Step 1: Pawn piece moves to a location on the board | Step 2: update new location, display Pawn piece in new location on the board |
| **Alternate Scenarios** | **Actor Action** | **System Response** |
| **Invalid Location** |  | Step 2: System identifies that the Pawn piece is moving to an invalid location.  Step 3: Display appropriate message and Pawn piece must select a different location to move to. |
| **Conclusions** | Pawn piece new location on the board has been updated | |
| **Post conditions** |  | |
| **Business Rules** |  | |
| **Implementation Constraints** |  | |

### **5.6.2 Activity Diagram**

Start

Black’s turn

Red’s turn

Red’s turn

Checkmate

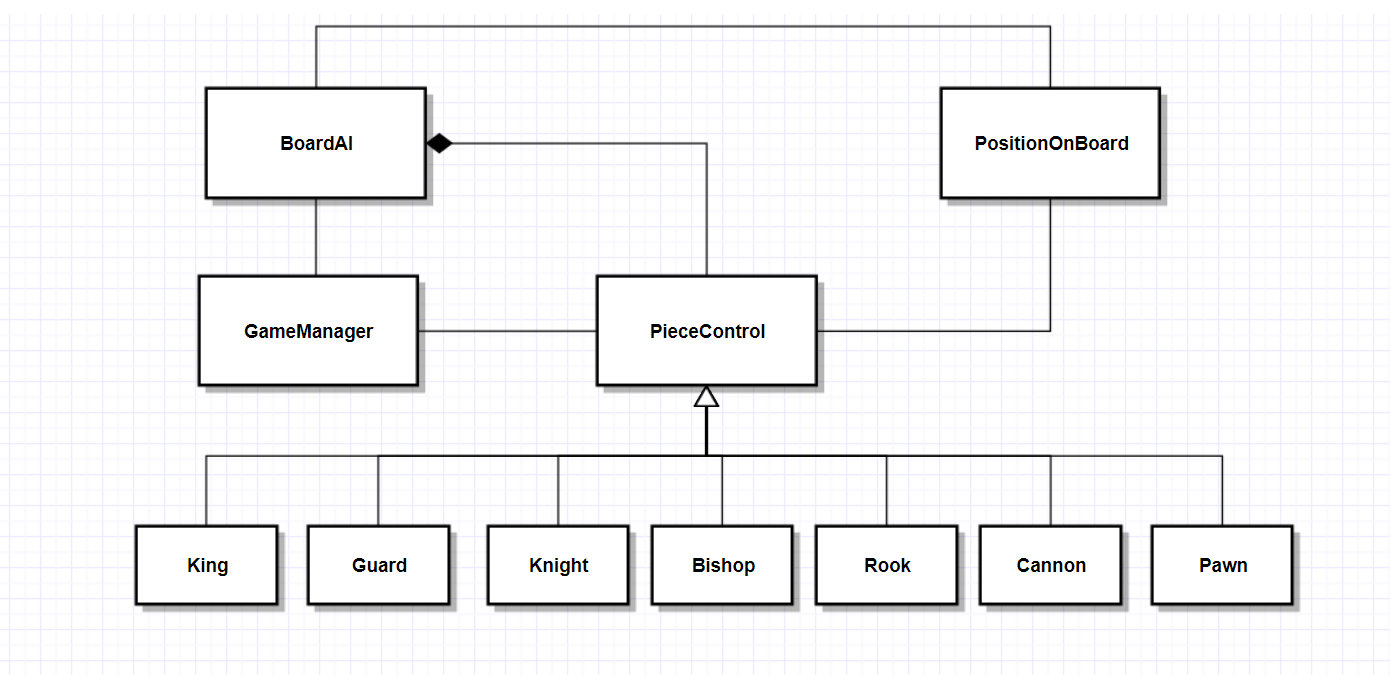
Black’s turn

Checkmate

Red wins

Black wins

### **5.6.3 Key Class Diagram**



### **5.6.4 VOPC**

### **5.6.5 Sequence Diagrams**

#### **5.6.5.1 Sequence Diagram for Start Game**

User GameManager BoardAI

Start Game

Start Game

#### **5.6.5.2 Sequence Diagram for Moving a Piece**

User GameManager BoardAI Piece

select Piece get list of moves get list of moves

update board give move list give move list

destination for piece

is legal move is legal move

return legal move return legal move

if valid, move piece

# **Chapter 6: Implementation**

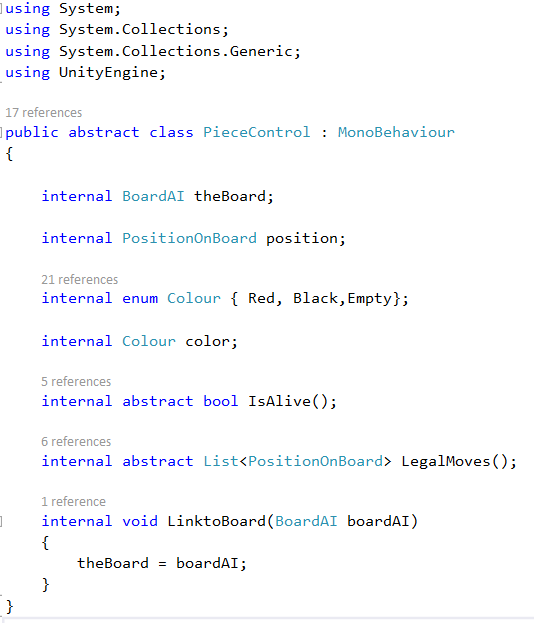
## **6.1 Prototype and Sprints**

### **6.1.1 Sprint 1**

|  |  |  |
| --- | --- | --- |
| **Prototype** | **Start Date** | **Finish Date** |
| 1 | 01/10/2017 | 13/12/2017 |

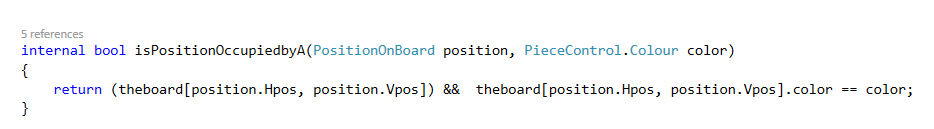
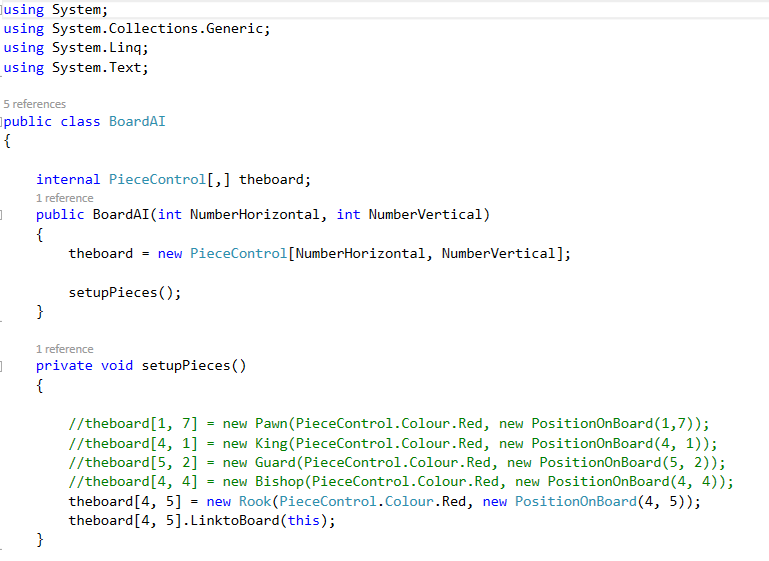
|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Create abstract class PieceControl | Complete |
| 2 | Create BoardAI class | Complete |
| 3 | Create PositionOnBoard class | Complete |
| 4 | Create Pawn class, listed out all possible moves | Complete |

This sprint started with the creation of the PieceControl abstract class. The PieceControl Class was created to act as a common class to all game pieces’ class. Each game pieces (King, Guard, Bishop, Knight, Rook, Cannon and Pawn) had its own classes to handle its movement. Each classes inherits all the abstract methods of the PieceControl Class.



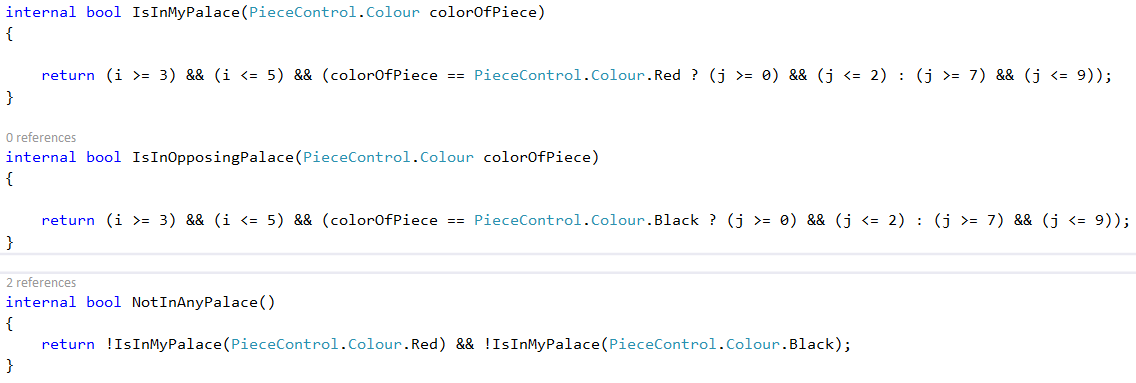
Code snippet 1: PieceControl Class

The BoardAI Class was created to handle each chess pieces set up on the game board.



Code Snippet: isPositionOccupiedbyA() method in the BoardAI Class

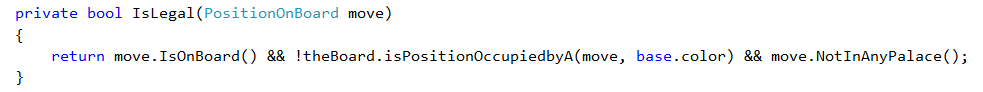
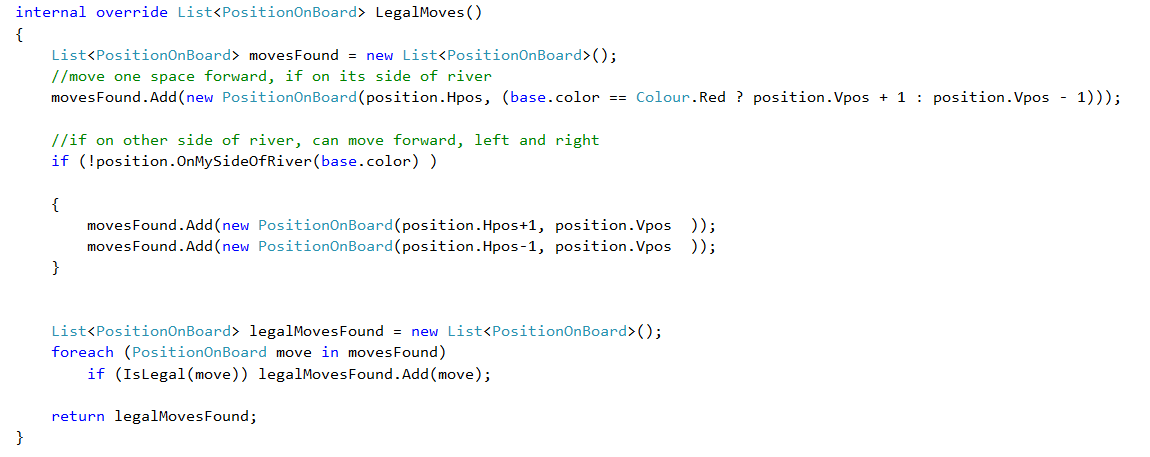
The PositionOnBoard Class was created to store the number of horizontal and vertical spaces on the game board. In this case it was 9 (horizontal) x 10 (vertical). This Class includes methods such as OnMySideOfRiver(), which specified the location of the river for both side (0 - 4 (incl) for Red, 5-9 (incl) for black), the isOnBoard() method was created to specified the number of spaces of the game board, if a piece was place outside of that, it would not be a valid move. The IsInMyPalace(), IsInOpposingPalace() method specified the location of the palace for both side. The NotInAnyPalace() method returns true if the piece is not in either side of the palace.



Code Snippet: IsInMyPalace(), IsInOpposingPalace() and NotInAnyPalace() method in the PositionOnBoard Class

During this part of the sprint, the Pawn Class was also created. The Pawn Class extends the PieceControl Class, and was used to handle all possible movements of each Pawn pieces. The LegalMoves() method in the Pawn Class was used to store all possible moves of the Pawn piece into a list and returns a list of possible legal moves. The isLegal() method was used to check the following:

* The Pawn piece’s move is on the board
* The position of the Pawn piece’s move is not already occupied by another piece
* The Pawn piece’s move is not in any palace



Code Snippet: Pawn Class

### **6.1.2 Sprint 2**

|  |  |  |
| --- | --- | --- |
| **Prototype** | **Start Date** | **Finish Date** |
| 2 | 22/01/2018 | 09/02/2018 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Create King class to handle legal movements of the King piece | Complete |
| 2 | Create Guard class to handle legal movements of the Guard piece | Complete |

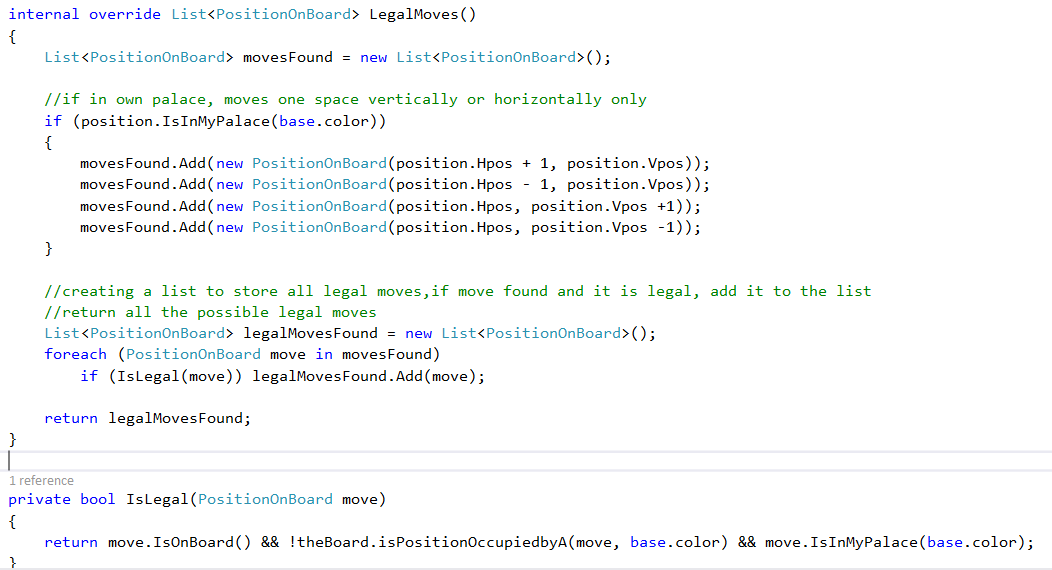
In this sprint, the King and Guard Classes were created to handle movements of each of the King and Guard pieces. Same as the Pawn Class in sprint 1, the King and Guard Classes extends PieceControl Class. The LegalMoves() method was used to store all possible moves for the King and Guard pieces into a list.

For the King class, the isLegal() method was used to check the following:

* The King piece’s move is on the board
* The position of the King piece’s move is not already occupied by another piece
* The King piece is within its palace



Code Snippet: King Class extends PieceControl Class



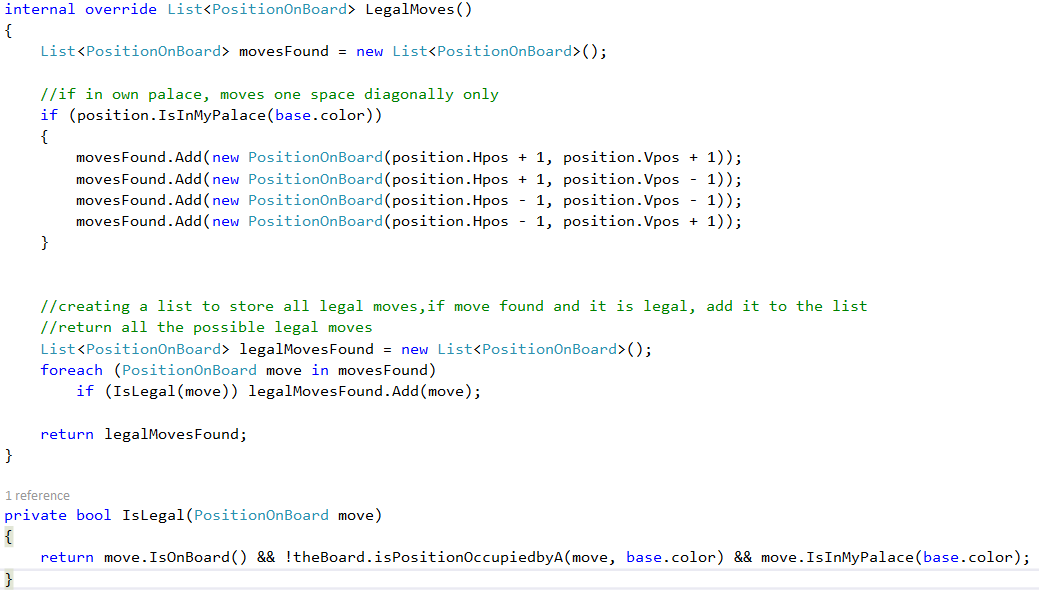
Code Snippet: King Class

For the Guard Class, the isLegal() method was used to check the following:

* The Guard piece’s move is on the board
* The position of the Guard piece’s move is not already occupied by another piece
* The Guard piece is within its palace

C:\Users\Suki\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Screenshot (144).png

Code Code Snippet: Guard Class



### **6.1.3 Sprint 3**

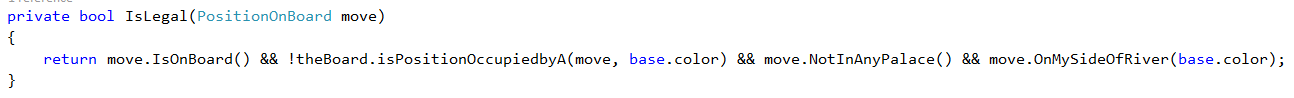
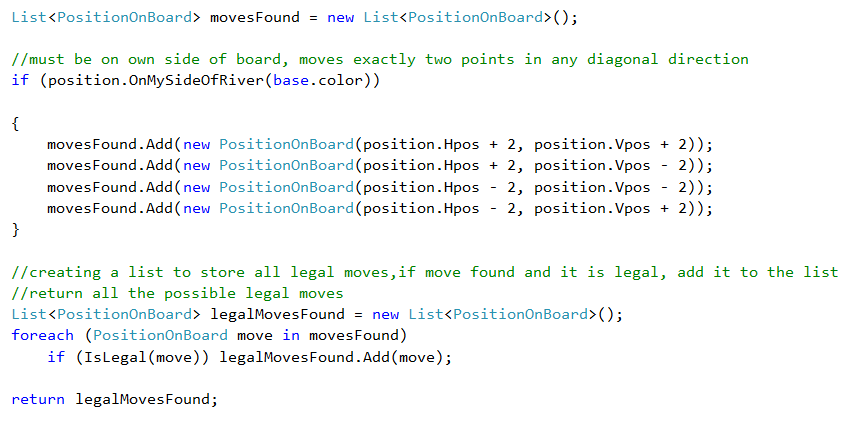
|  |  |  |
| --- | --- | --- |
| **Prototype** | **Start Date** | **Finish Date** |
| 3 | 12/02/2018 | 26/02/2018 |

|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Create Bishop class to handle legal movements of the Bishop piece | Complete |
| 2 | Create Rook class to handle legal movements of the Rook piece | Complete |
| 3 | Create Cannon class to handle legal movements of the Cannon piece | Complete |

In this sprint the Bishop, Rook, and Cannon Classes were created to handle movements of each of the Bishop, Rook and Cannon Pieces. Same as the Pawn class in sprint 1, the Bishop, Rook and Cannon Classes extends PieceControl Class. The LegalMoves() method was used to store all possible moves for the Bishop, Rook and Cannon pieces into a list.

For the Bishop Class, the isLegal() method was used to check the following:

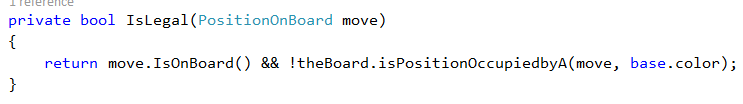
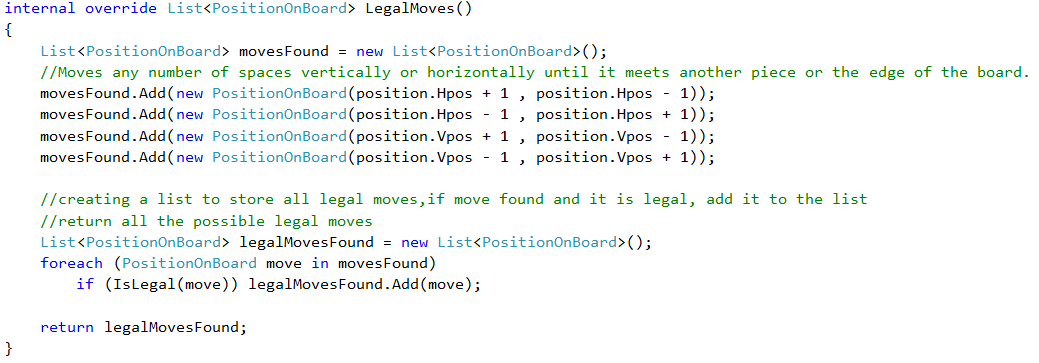
* The Bishop piece’s move is on the board
* The position of the Bishop piece’s move is not already occupied by another piece
* The Bishop piece’s move is not in any palace
* The Bishop piece is on its side of the river



Code Snippet: Bishop Class

For the Rook Class, the isLegal() method was used to check the following:

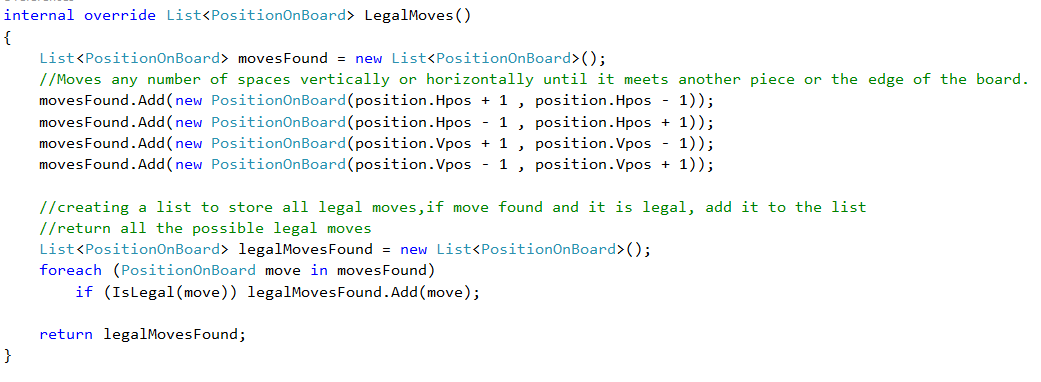
* The Rook piece’s move is on the board
* The position of the Rook piece’s move is not already occupied by another piece

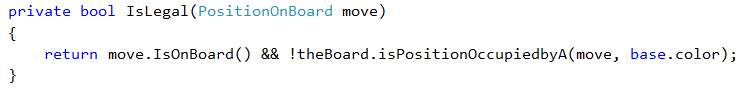


Code Snippet: Rook Class

For the Cannon Class, the isLegal() method was used to check the following:

* The Cannon piece’s move is on the board
* The position of the Cannon piece’s move is not already occupied by another piece





Code Snippet: Cannon Class

### **6.1.4 Sprint 4**

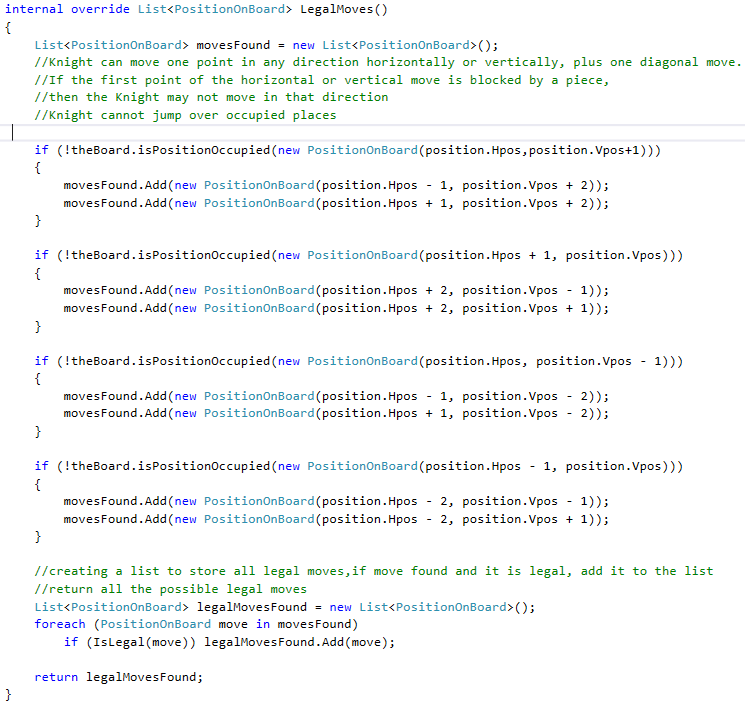
|  |  |  |
| --- | --- | --- |
| **Prototype** | **Start Date** | **Finish Date** |
| 4 | 05/03/2018 | 23/03/2018 |

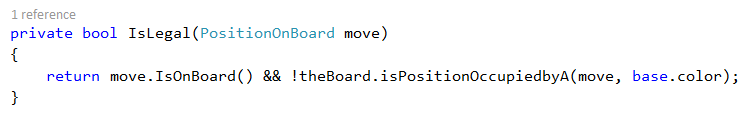
|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Create Knight class to handle legal movements of the Knight piece | Complete |
| 2 | Set up each game piece on the board | Complete |
| 3 | Add a CurrentBoard() method in BoardAI | Complete |
| 4 | Set up BlankSpace Class to deal with blank spaces on the board | Complete |
| 5 | Assign pieceID to each piece | Complete |

In this sprint the Knight Class was created to handle movements of all the Knight Pieces. Same as the Pawn Class in sprint 1, the Knight Class extends PieceControl Class. The LegalMoves() method was used to store all possible moves for the Knight pieces into a list. In addition, the isPositionOccupied() method was created to check if there was a piece occupied, if there is a piece occupied, the LegalMoves() method checks for other possible moves for the Knight piece.

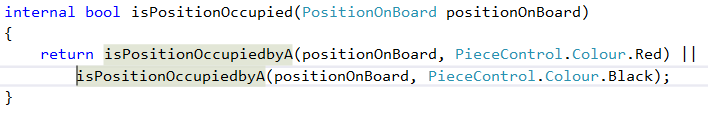
For the Knight Class, the isLegal() method was used to check the following:

* The Knight piece’s move is on the board
* The position of the Knight piece’s move is not already occupied by another piece



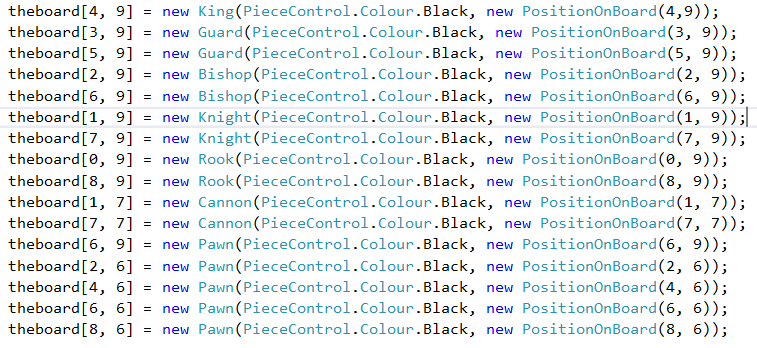


Code Snippet: Knight Class

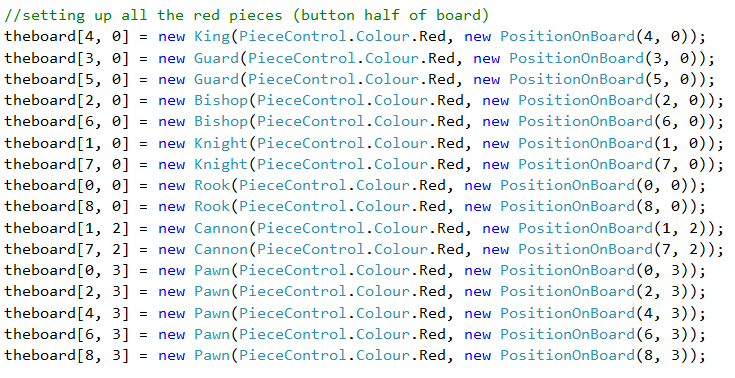


Code Snippet: isPositionOccupied() method in BoardAI Class

After listed out all possible movements for each pieces, the next step was setting up each pieces on the game board. In the BoardAI Class, the setupPieces() method was created to set up each pieces on the game board.

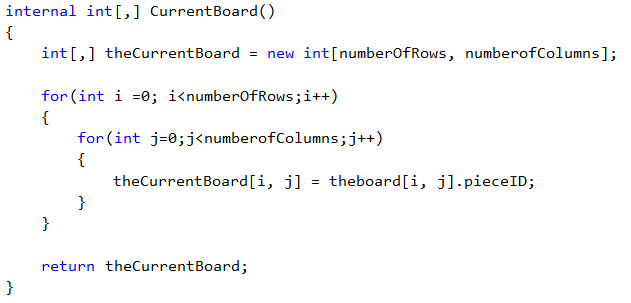


Code Snippet: Setting up all black pieces



Code Snippet: Setting up all red pieces

To get the current state of the board, the CurrentBoard() method was created in BoardAI Class. This method stores the current board as a 2D array integer. It also contained two for loops which loops through the board to get the ID of the pieces. This method returns the current board as integer array.

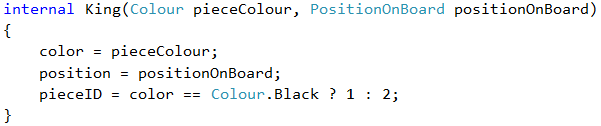


Code Snippet: CurrentBoard()

In order to know which piece is returned from the CurrentBoard() method, each pieces was assigned with a pieceID.

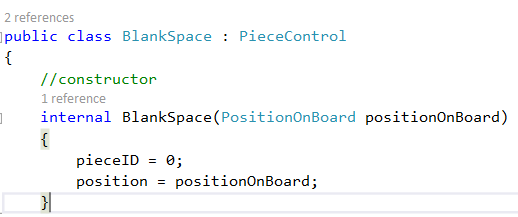
|  |  |  |
| --- | --- | --- |
| **Piece Name** | **PieceID (Black Piece)** | **PieceID (Red Piece)** |
| Blank Space | 0 | 0 |
| King | 1 | 2 |
| Guard | 3 | 4 |
| Bishop | 5 | 6 |
| Knight | 7 | 8 |
| Rook | 9 | 10 |
| Cannon | 11 | 12 |
| Pawn | 13 | 14 |

Inside the constructor of each pieces’ class, the pieceID was assigned. The conditional expression (also known as ?: operator in C#) was used to determine if the King piece is a black piece, assign pieceID 1 to it, if it’s a red King piece, assign pieceID 2. The same was applied to the other pieces.



Code Snippet: Assigning pieceID to the King Piece

When the pieces were set up on the board, there were many blank spaces left on the board. This may cause issues to the CurrentBoard() method in determining whether there is a piece occupied at the position when looping through the board. To avoid this issue a BlankSpace Class was created to deal with all the blank spaces on the board. All blank spaces were assigned pieceID of 0.



Code Snippet: BlankSpace Class

### **6.1.5 Sprint 5**

|  |  |  |
| --- | --- | --- |
| **Prototype** | **Start Date** | **Finish Date** |
| 5 | 09/04/2018 | 23/04/2018 |

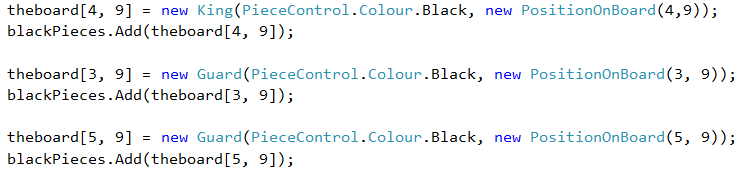
|  |  |  |
| --- | --- | --- |
| **Task Number** | **Details** | **Status** |
| 1 | Creating lists to store each pieces to the board | Complete |
| 2 | Moving the piece, updating the board |  |
| 3 | Creating game tree |  |

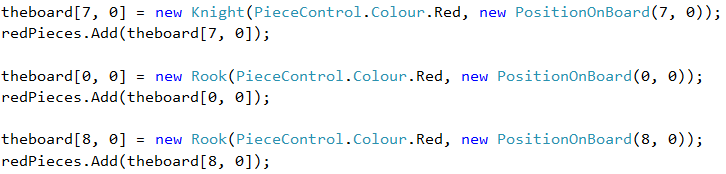
In this sprint, two lists were created to store each pieces. A list for all red pieces and a list for all black pieces. Then each pieces were added to their respective location on the board.





Code Snippet: Setting up the list for the red and black pieces





Code Snippet: Adding each pieces to their respective location on the board

# **Chapter 7: Findings and Conclusion**

# References

Agatonovic-Kustrin, S. & Beresford, R., 2000. Journal of Pharmaceutical and Biomedical Analysis. *American Association of Pharmaceutical Scientists,* 22(5), pp. 717-727.

AncientChess.Com, 2007. *How to Play Chinese Chess.* [Online]   
Available at: http://ancientchess.com/page/play-xiangqi.htm  
[Accessed 06 12 2017].

Banavar, G., 2016. *Guruduth Banavar: Cognitive Systems.* s.l.:UM Stamps.

Bengio, Y., 2016. Machines Who Learn. 01 June, 314(6), pp. 46-51.

BrainKing, 2005. *Chinese Chess (Xiangqi).* [Online]   
Available at: https://brainking.com/en/GameRules?tp=68  
[Accessed 06 12 2017].

Campbell, M., Hoane Jr, A. J. & Hsu, F.-h., 2002. Deep Blue. *Artificial Intelligence,* Issue 134, pp. 57-83.

Computer History Museum, 2005. *Chess Software Basics.* [Online]   
Available at: http://www.computerhistory.org/chess/chess-software-basics/  
[Accessed 11 04 2018].

Coppin, B., 2004. *Artificial Intelligence Illuminated.* s.l.:Jones & Bartlett Publishers.

Curley, R., 2011. *Computing: From the Abacus to the IPad.* s.l.:Britannica Educational Pub.

DeepMind Technologies Limited, n.d. *AlphaGo.* [Online]   
Available at: https://deepmind.com/research/alphago/  
[Accessed 07 10 2017].

Donnelly, P., 2014. *Basics of XiangQi Play.* [Online]   
Available at: http://www.xqinenglish.com/peterdonnellybasicsofplay.html  
[Accessed 07 10 2017].

Edgar, J., 2014. *'Captain Cyborg': the man behind the controversial Turing Test claims.* [Online]   
Available at: http://www.telegraph.co.uk/news/science/science-news/10888828/Captain-Cyborg-the-man-behind-the-controversial-Turing-Test-claims.html  
[Accessed 25 11 2017].

French, R. M., 2000. The Turing Test: the first 50 years. *Trends in Cognitive Sciences,* 4(3), pp. 115-122.

Goldberg, D. E. & Holland, J. H., 1988. Genetic Algorithms and Machine Learning. *Machine Learning (1988) 3,* 3(2-3), pp. 95-99.

Gui, W. & Jun, T., 2016. Chinese Chess Algorithm Design and Implementation in the Computer Game. *Proceedings of the 35th Chinese Control Conference,* pp. 10380-10384.

Hansen, L. & Larsen, J., 1996. Unsupervised learning and generalization. *IEEE International Conference on Neural Networks,* Issue 96, pp. 25-30.

Hsu, F.-H., 1999. IBM's Deep Blue Chess grandmaster chips. *IEEE Micro,* 19(2), pp. 70-81.

IBM 100, 2011. *A Computer Called Watson.* [Online]   
Available at: http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/watson/  
[Accessed 25 11 2017].

IBM Research, 2011. *The DeepQA Research Team.* [Online]   
Available at: http://researcher.watson.ibm.com/researcher/view\_group\_subpage.php?id=2159  
[Accessed 25 11 2017].

IBM Research, 2014. *Watson.* [Online]   
Available at: http://www.research.ibm.com/cognitive-computing/watson/index.shtml  
[Accessed 25 11 2017].

IBM Research, 2017. *TD-Gammon.* [Online]   
Available at: http://researcher.watson.ibm.com/researcher/view\_page.php?id=6853  
[Accessed 10 12 2017].

IBM, 2011. *Deep Blue.* [Online]   
Available at: http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/  
[Accessed 27 09 2017].

IBM, 2017. *IBM Watson.* [Online]   
Available at: https://www.ibm.com/power/solutions/cognitive-computing  
[Accessed 25 11 2017].

Jones, M. T., 2017. *A beginner's guide to artificial intelligence, machine learning, and cognitive computing.* [Online]   
Available at: https://www.ibm.com/developerworks/library/cc-beginner-guide-machine-learning-ai-cognitive/index.html  
[Accessed 11 10 2017].

Jones, M. T., 2017. *Train a software agent to behave rationally with reinforcement learning.* [Online]   
Available at: https://www.ibm.com/developerworks/library/cc-reinforcement-learning-train-software-agent/index.html  
[Accessed 11 12 2017].

Kehoe, D., 2015. *Designing Artificial Intelligence for Games (Part 1).* [Online]   
Available at: https://software.intel.com/en-us/articles/designing-artificial-intelligence-for-games-part-1  
[Accessed 22 10 2017].

Kendall, G., 2001. *Introduction to Artificial Intelligence.* [Online]   
Available at: http://www.cs.nott.ac.uk/~pszgxk/courses/g5aiai/005gameplaying/game-playing.htm  
[Accessed 14 10 2017].

Li , Y., Liu , X., Zhang , H. & Xi , Y., 2016. Study on system modeling and optimization of static evaluation based on the computer game of draughts. *Chinese Control and Decision Conference (CCDC),* pp. 4283-4287.

Maglogiannis, I., Karpouzis, K., Wallacce, B. & Soldatos, J., 2007. *Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in EHealth, HCI, Information Retrieval and Pervasive Technologies.* s.l.:IOS Press.

Marsland, S., 2015. *Machine Learning: An Algorithmic Perspective.* 2nd ed. s.l.:CRC Press.

Mitchell, T. M., 1997. *Machine Learning.* s.l.:McGraw-Hill Science/Engineering/Math.

Rasskin-Gutman, D., 2009. *Chess Metaphors : Artificial Intelligence and the Human Mind.* Cambridge: The MIT Press.

Ross, D., 2014. *Game Theory.* [Online]   
Available at: https://plato.stanford.edu/entries/game-theory/  
[Accessed 14 10 2017].

Russell, S. J. & Norvig, P., 2010. *Artificial intelligence : a modern approach.* 3rd ed. s.l.:Prentice Hall.

Samuel, A. L., 1959. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal,* 3(3), pp. 210-229.

Schölkopf, B., 2015. Artificial intelligence: Learning to see and act. 518(7540), pp. 486-487.

Silver, D. et al., 2016. Mastering the game of Go with deep neural networks and tree search. *Nature,* Volume 529, pp. 484-489.

Tesauro, G., 1995. Temporal Difference Learning and TD-Gammon. *Comunications of ACM,* 38(3), pp. 58-68.

Trice, A., 2015. *The Future of Cognitive Computing.* [Online]   
Available at: https://gamesdeveloper.today/the-future-of-cognitive-computing/  
[Accessed 09 12 2017].

Turing, A. M., 1950. Computing Machinery and Intelligence. *Mind,* 49(236), pp. 433-460.

Vardi, M. Y., 2014. Would Turing Have Passed the Turing Test?. *Communications of ACM,* 57(9), p. 5.

Wang, J. & Li, S., 2008. *Representing evaluation of computer Chinese chess by artificial neural network using genetic algorithm.* Yantai, Shandong, China, Chinese Control and Decision Conference (CCDC), pp. 1226-1232.

Yang , Z. et al., 2017. Research and Implementation of Static Evaluation Algorithm for Checkers. *Chinese Control And Decision Conference (CCDC),* pp. 7665-7670.

Yellow Mountain Imports, 2010. *How to Play Xiangqi / Chinese Chess / 象棋.* [Online]   
Available at: https://www.ymimports.com/pages/how-to-play-xiangqi-chinese-chess  
[Accessed 11 10 2017].

Yen, S.-J., Chen, J.-C., Yang, T.-N. & Hsu, S.-C., 2004. Computer Chinese Chess. *ICGA Journal,* p. 18.

Yu, W.-G., 2012. Designing a high-performance Chinese chess computer player for middle game position. *Proceedings of the 2012 International Conference on Machine Learning and Cybernetics,* pp. 571-57.