Palestine Polytechnic University

Deanship of Graduate Studies and Scientific Research

Master of Informatics

Using Reinforcement Learning to Learn Seega Board Game

Submitted by:
Mahdi Attawna

Thesis submitted in partial fulfillment of requirements of the degree Master of Science in Informatics

Aug, 2020

The undersigned hereby certify that they have read, examined and recommended to the Deanship of Graduate Studies and Scientific Research at Palestine Polytechnic University the approval of a thesis entitled: **Using Reinforcement Learning to Learn Seega Board Game**, submitted by **Mahdi Attawna** in partial fulfillment of the requirements for the degree of Master in Informatics.

Graduate Advisory Committee:

Dr. Hashem Tamimi (Supervisor), Palestine Polytechnic University.

Signature:____________________ Date:_______________

Dr. Alaa Halawani, Palestine Polytechnic University.

Signature:____________________ Date:_______________

Dr. Jihad El-Sana, University of the Negev.

Signature:____________________ Date:_______________

Thesis Approved

Dr. Murad Abusubaih

Dean of Graduate Studies and Scientific Research

Palestine Polytechnic University

Signature:____________________ Date:_______________

DECLARATION

I declare that the Master Thesis entitled **"Using Reinforcement Learning to Learn Seega Board Game"** is my original work, and hereby certify that unless stated, all work contained within this thesis is my own independent research and has not been submitted for the award of any other degree at any institution, except where due acknowledgement is made in the text.

Mahdi Lutfi Attawna

Signature:____________________ Date:______________

STATEMENT OF PERMISSION TO USE

الملخص

تعد ألعاب الكمبيوتر أحد المجالات الشائعة التي تستخدم التعلم الآلي والذكاء الاصطناعي. تحتوي العديد من هذه الألعاب على عدد كبير من المواضع المحتملة التي تجعلها كبيرة جدًا حتى بالنسبة لأجهزة الكمبيوتر الفائقة ان تقوم بحلها وتعلمها. السيجة هي لعبة لوحية مصرية ثنائية اللاعبين شبيهة بالشطرنج ولكن لديها قواعد أكثر صعوبة ، ولها مرحلتان ، في المرحلة الاولى يضع اللاعبون الحجارة على اللوح في الاماكن الفارغة، في المرحلة الثانية يقوم اللاعبون بتحريك حجارتهم رأسيًا أو أفقيًا لالتقاط حجر الخصم.

في هذا العمل ، استخدمنا التعلم المعزز في تدريب اثنين من برامج الكمبيوتر وهما برنامج التحريك وبرنامج التموضع لإتقان لعبة السيجة وتعلم قواعدها بدون تدخل بشري، قمنا بتعليم برنامج التموضع كيفية وضع أحجار اللاعب في الأماكن المناسبة على اللوح، وتم تعليم برنامج التحريك كيفية تحريك الأحجار بطريقة احترافية لالتقاط أحجار الخصم والفوز في اللعبة.

قمنا بعرض نتائج التجارب التي تم إجراؤها ومناقشة أداء البرامج التي تم تعليمها أثناء هذا العمل.

Abstract

Computer games are one of the popular fields which use machine learning and artificial intelligence. Many of these games have a large search space, which makes them too vast for even supercomputers to brute force. Seega is an ancient Egyptian two-player board game similar to chess but has more difficult rules. It has two stages, In the first stage, the players position their stones on the board in strategic manner. In the second stage, the players move their stones vertically or horizontal to capture the opponent stones. In this work, we used deep reinforcement learning to train two co-operative agents to master the game of Seega and learn the game rules. We compared our proposed approach with the classical mini-max algorithm. We found that our approach is much more practical to be adopted to be used in Seega in terms of computational time and needed resources.

Dedicated to my children *Khaled* and *Tawq*.

Acknowledgements

I would like to take this opportunity to acknowledge all those who helped me during this thesis work. I would like to thank my supervisor Dr. Hashem Tamimi for introducing me to the world of machine learning, his valuable suggestions, outstanding and insightful guidance during this thesis work which facilitated my achievement of this dissertation.

Finally, I must express my very profound gratitude to my parents and my wife for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer games are one of the popular fields which use machine learning and Artificial Intelligence. AI become an essential part of the game industry which attracted the interest of computer scientists, software developers, and machine learning researchers to dive into it. Many of the developed games have a large search space which makes them too vast for even supercomputers to brute force. This encourages the researcher to develop and propose different methods and techniques that overcome this problem.

There are different types of computer games. Board games have gained the attraction of the researchers since many years. Using heuristics and rules was one of the methods that made computer games possible; specially for games with small search spaces. Minimax and alpha-beta pruning are good examples of such algorithms.

For games with huge search spaces, such algorithms will need high computation time beyond the player's patience or even beyond their age. Machine learning was introduced to solve this issue by providing less computationally expensive algorithms that can learn from the prepared datasets of expert players. But such datasets are not available for many games such as the game of Seega, which makes the use of traditional supervised machine learning not a choice.

The problem of the absence of datasets for this game makes the reinforcement learning based on self-play games the best method to train our agent. Reinforcement Learning has been improved over the last decade. Many algorithms and methods developed in this area, especially the methods that use deep learning.

Seega is an interesting game. Unlike chess, the game has two stages. In the first stage, the players position their stones on the board in a strategic manner. In the second stage, the players move their stones vertically or horizontally to capture the opponent's stone.

Using a mini-max for each stage separately will not guarantee to win the game, because the first mini-max will not deal with heuristic on the second stage. On the other hand, using one mini-max for the two stages will need a very large search space.

Therefore, we will introduce a new solution to the Seega game using deep reinforcement learning. The proposed approach has one agent for each stage of the game and does not require a prepared dataset since it gets trained simultaneously by playing with another computer agent.

1.1 Main Contributions

In this thesis, we developed two computer agents that can play the Seega game. The first agent can play stage-1 of the game (*Positioning Agent*). The second agent can play the stage-2 of the game which we call *Moving Agent*. We used reinforcement learning to train the two agents on self-play games without human data or guidance. This work used the Monte Carlo tree search to play and simulate games, and deep convolution neural networks to train the agents.

The training process is divided into two steps. We trained stage-2 then stage-1 as follows: First, we trained the *Moving Agent* using self-play games on randomly-generated boards; which trained the agent to select the best moves of the stones to win the game. Then we trained the *Positioning Agent* using self-play games starting from an empty board to select the best cell index to place the player piece in the perspective of the current player. The trained *Moving Agent* is also used during the training of *Positioning Agent* to predict the winner of the game when the players finish positioning all their stones instead of playing the full game.

1.2 Thesis Structure

This thesis is composed of five chapters. Chapter 1 gives an introduction, discusses the problem statement, and explains the main contribution in this thesis. Chapter 2 contains a brief historical background and explains some fundamental concepts related to the Seega game. Chapter 3 explains some concepts used in this thesis and discusses the related studies. Chapter 4 explains the methodology used in the work, it shows the system architecture and describes in detail the proposed architecture of the developed system and the implementation of the theoretical model used on the matter. Chapter 5 presents experiments and results that describe several case studies made, showing result analysis and the solutions obtained. Finally, Chapter 6 summarizes the respective work, the following conclusions about the results obtained and the approaches used, together with several future work propositions.

Chapter 2

The game of Seega

In this chapter, a detailed discussion about the history and rules of Seega game with examples to clarify the challenges in this work are presented. Then we will discuss some of the related studies achieved on Seega game and explain other reinforcement learning researches that have been done in the field of computer games and board games.

2.1 Seega Game

Seega is a two-player board game that is popular in many countries of middle-east and Africa such as Egypt, Sudan, Libya, Somalia, Jordan, and some parts of Palestine. People in these countries used to play this game for hours during the day. The game has many similarities with other board games such as chess. It needs a lot of focus and thinking to win the match.

2.1.1 Seega History

The history of the game is unknown but some of Seega boards were found carved in many different Egyptian temples built since the 1300s BCE. Figure 2.1 shows Seega game board, carved on the wall of Elkab Temple of Amenhotep III, Elkab, Egypt.

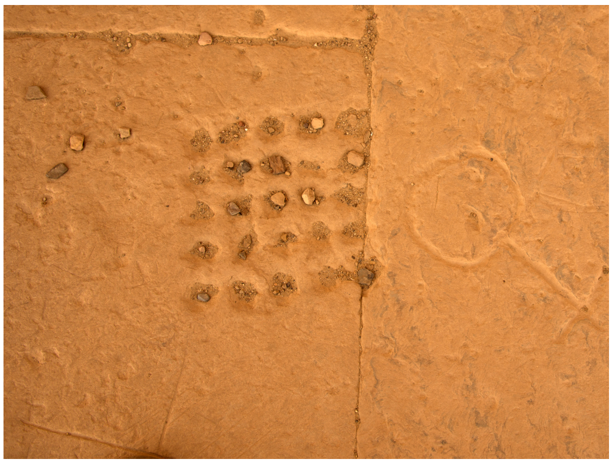


Figure 2.1: Seega game board, carved on the wall of Elkab Temple of Amenhotep III, Elkab, Egypt. Photo: Bruce Allardice, July 2, 2013. [1]

Seega has very complex rules, some researchers consider it harder than chess [11]. The game rules and literature first appearance was in in a book by Edward William Lane, called “An Account of the Manners and Customs of the Modern Egyptians” in 1830 [12], later in 1892, More details about its rules and strategies were discussed by ”H Carrington Bolton” in his book “Games Ancient and Oriental, and How to Play Them” [13].

2.1.2 Seega Components

Similar to most board games, Seega has two components: The board and stones, Figure 2.2 shows an example of Seega board with its components.

1. **The board:** the game comes in many different variations, a 5×5 grid and 24 playing stones of two different colors, 12 stones per player. a 7×7 board with 48 stones, 24 stones per player, and a 9×9 board with 80 stones, 40 stones per player. all the versions share the same rules.

2. **Stones:** depends on the size of the board, each player has a known number of stones called Kelab, they should be different in color or shape for each player.

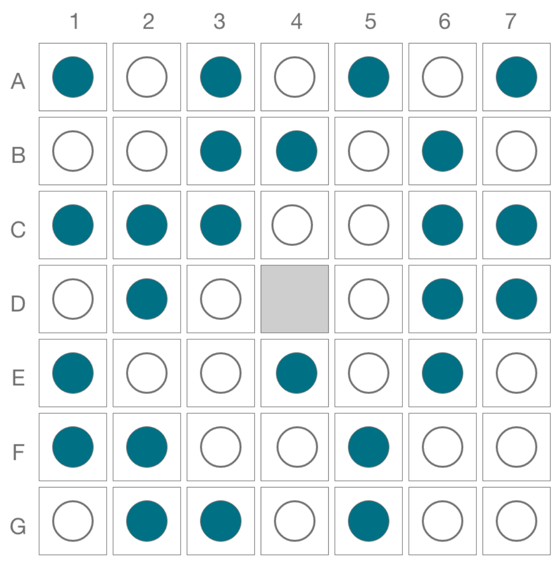


Figure 2.2: Seega 7x7 game board example, each player has 24 tile with same color (black/white).

2.1.3 Seega Stages and Rules

Seega could have different rules and variations, but the core rules are shared among all variation in all countries, Following are the main rules:

- Seega is a two-player game.
- Each player owns a fixed number of stones that have the same color or shape that is different from the opponent's stones.
- The objective of each player is to capture as many as he can from the opponent player's stones.
- The game ends when one player has less than two stones on the board.
- At the end of the game, the player with more stones is the winner.
- The game has two distinct phases: The positioning phase and the moving phase.
- In the *Positioning phase*, each player takes a turn and places two stones on any empty cells on the board except the central cell. They continue adding stones in turn until they fill all board cells (except the center). This will lead the game to phase two. Figure 2.3 shows an example of the actions taken by the players in the positioning stage.
- In the *Moving phase*:
  - In turn, each player starts to move his stones on the board to surround the other player stones.
  - The movement of the piece will be from an occupied cell to an empty adjacent cell, e.g, in figure 2.2 move $E4$ to $D4$.
  - Stones can only move horizontally or vertically, and not diagonally.
  - The movement will be one cell a time.

- When the player moves a piece, if any opponent stones are surrounded by the player stones (horizontally or vertically) as a result of the move it will be captured and removed from the board, e.g: in figure 2.2 when moving $E4$ to $D4$ the white stones at $C4$, $D3$, and $D5$ will be removed.
- In the same turn, The player can continue to capture more opponent stones with the same stone, eg: after moving $E4$ to $D4$ and removing the surrounded white stones, then the player can continue and move from $D4$ to $C4$ since $C5$ can be removed.

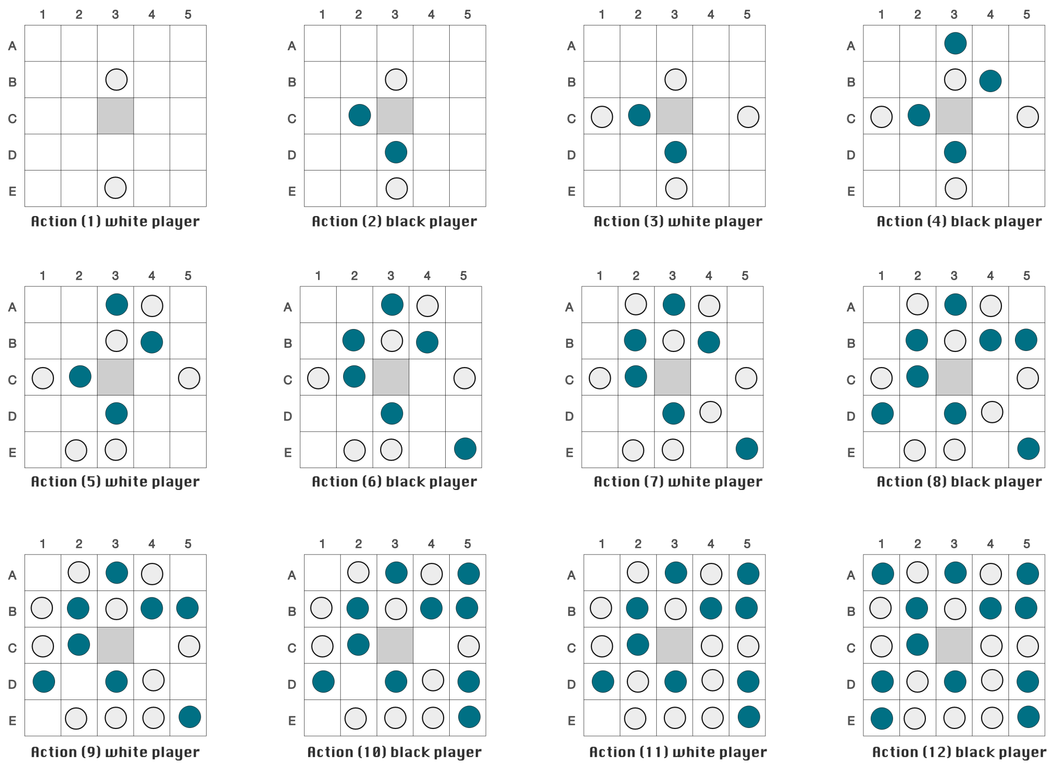


Figure 2.3: Example of playing positioning stage of the game.

Chapter 3

Background and Literature Review

In this chapter, we will explain some of the machine learning concepts that are are related to this work. We will discuss the different types of machine learning algorithms, explain the reinforcement learning components and methods, and clarify the Monte Carlo tree search and convolutional neural network (CNN).

3.1 Introduction to Reinforcement Learning

Reinforcement Learning (RL) is a sub-type of machine learning, which is ideal to be used in sequence decision making. The learning model is discovered by the agent without prior examples or training data. RL is based on two main components, *Agent* and *Environment.* The *Agent* explores the *Environment* by sending actions and getting feedback from the *Environment* called *Reward.* The *Reward* gives the *Agent* a note about how good the action was. The goal of the

Agent is to learn the best model which is called *Policy*, that maximizes the total rewards it earns. The *Policy* is simply a mapping from states to action and the *Agent* will learn it by trial and error [14].

As described in [14], RL has five main elements: an environment, an agent, a policy, a reward signal, and a value function. Figure 3.1 shows the basic elements of Reinforcement Learning and shows how they interact with each other.

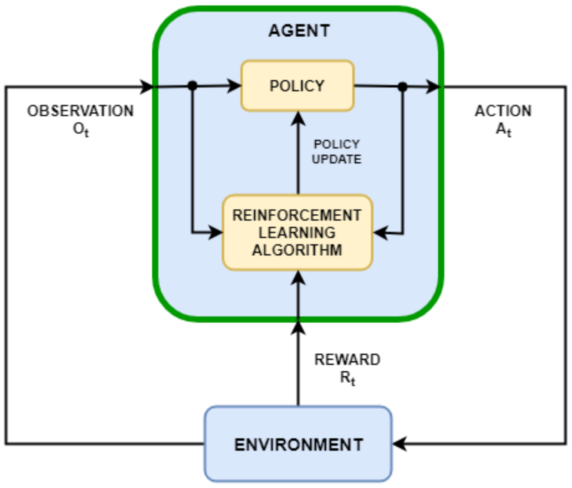


Figure 3.1: Reinforcement Learning basic elements [2].

- **Environment:** In RL, the environment is the world which the agent will explore to learn it's rules and policies. In other words, the environment contains the problem that the agent is trying to solve. At any time t, the environment will have a state, s, that agent will observe, and according to this observation, the agent will act by sending an action, a. After each action by the agent, the environment should return a value that represents the reward to the agent.

- **Agent:** The agent is the RL algorithm used to explore the environment and learn how to act at a certain observation (state s).

- **Policy** The policy can be described as a mapping from the states of the environment to action, it is used by the agent to take the right action when it observes the state of the environment. The policy is the core of Reinforcement learning. It is the knowledge that the agent collects during the learning process. It can be as simple as a lookup table or can be more complex and include advanced searching algorithms.
- **Rewards/Goal:** The reward is a single value sent from the environment immediately to the agent to define what is the good and bad action. The goal of the agent is to maximize the total rewards it receives during the learning process. The agent will receive a positive reward when its action is considered good behavior, and it will receive a negative value from the environment when its action is considered bad behavior.
- **Value function:** The value function is sometimes used to determine what is good for the agent in the long run. it can be described as the total amount of the rewards that can be expected to be collected by the agent in the future based on the current state.

3.2 Markov Decision Process

Markov Decision Process (MDP) is the mathematical formulation of RL problem. An MDP satisfies the Markov property, which mean that the current state completely characterises the state of the world. MDP can be used to show how much reward is collected through a particular sequence of actions that we sampled.

MDP can be defined by a tuple of objects (S,A,R,P, γ), where :

- S: is a set of possible states,
- A: is a set of possible actions.
- R: is a distribution of reward given (*state,action*) pair, it is a mapping from (*state,action*) to reward. it tell us what is the immediate reward we expect to get from state S at the moment.
- P: is a transition probability or distribution over next state given (*state,action*) pair.
- γ: is a discount factor, it means how much the agent cares about future rewards.

Algorithm 1 describes the steps of MDP. The goal of the algorithm is to find the optimal policy π* that maximizes the total reward.

Algorithm 1: Markov Decision Process (MDP) algorithm

```
- Initialization the policy $\pi$;
- At time step t=0, initialize environment state $s_0$ and $P(s_0)$ ;
while $t$ != End do
    - Agent selects action $a_t$, based on the policy $\pi$ ;
    - Environment creates reward $r_t$;
    - Environment creates next state $s_{t+1}$;
    - Agent receives reward $r_t$ and next state $s_{t+1}$.
end
```

3.3 Q-value function and Bellman equation

The Q-value function is used calculate the expected total cumulative rewards when taking action a on state s and following the policy, it is similar to value function but it also takes the action into the consideration.

During the learning process, the policy will be changed by time and the experience the agent gets, this will give us different value function each time. finding the optimal value function which gives the max value compared to all other value functions is challenging.

Many RL algorithms such as Qlearning [15] and TD Learning [16] rely on the foundation of MDP.

3.4 Classes of RL Algorithms

There are two categories of reinforcement learning algorithms based on how the agent learns the optimal policy. These categories are Model-free Learning Method and Model-based Learning.

In *Model-free Learning* category, the agent will learn from experience by interacting with the environment directly and collecting the rewards (positive/negative). Based on the collected reward it updates its policy or value function. This category of RL can be applied to any RL problem since it does not need any model of the environment, but it needs much more time to learn. Examples of algorithms in this class are Q-Learning [15], and SARSA.

In *Model-based Learning*, the algorithm tries to learn the transition probability between states. It uses Model-free in its early stages of learning to build the model, then it uses the model to predict the rewards without interacting with the environment, this will make the learning process take less time compared to model-free methods. Examples of algorithms in this class are Dyna-architecture [17] and Dyna-Q algorithms.

3.5 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is an algorithm that can select the best action out of a set of actions available for a specific state. This search algorithm depends on probabilities at its core. It uses RL principles in combination with the classical search techniques to improve the performance and overcome the computation limitations found in other tree search algorithms such as mini-max.

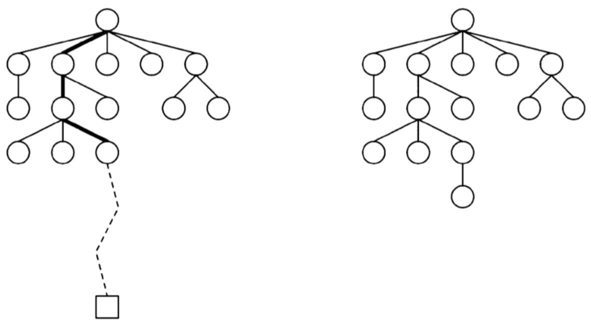


Figure 3.2: The basic MCTS process [3]

This algorithm uses *exploration-exploitation* strategy, It exploits the best actions and strategies that are found till the current iteration, and also continue to explore the space for alternative decisions that can be better than the current best actions and strategies and replace the current best.

Over the last few years, MCTS proved to be successful in many fields especially in computer games. It succeeded in playing general games, specific games, and complex games (eg: GO, and chess). Also, it succeeded in real world fields such as real-world planning, optimization, and control problems. [18]. This great success made it an important part of the AI researcher's toolkit.

In the MCTS search tree, the nodes are the building blocks. They represent the states of the problem as shown in Figure 3.2. The algorithm is repeatedly trying to improve its policy by exploring these nodes.

The process of this algorithm has four main steps that are repeated until a good policy is delivered or the max number of iteration is reached. The steps of MCTS are explained is shown in Figure 3.3.

- *Selection*: In this step, the MCTS algorithm tries to select a node on the tree that has the highest possibility of winning. It traverses the current tree using a strategy starting from the root node. This specific strategy selects the nodes that have the highest estimated value using the evaluation function. When MCTS reaches a leaf node (The nodes that are not expanded till this time) it jumps into the expansion step which is described next.

- *Expansion*: In this step, MCTS algorithm starts from the selected node in the previous step and expands the tree by adding its children. This step is meant to increase the options further in the game.

- *Simulation*: In this step, MCTS algorithm will simulate to play the game until it reaches the end of the game and collects the result (win/lose). In the early iterations of the algorithm, the simulation will be random and RL could be used to improve it.

- *Back Propagation*: In this step, The algorithm will use the value obtained from the previous step to update the values of all parent nodes back to the root.

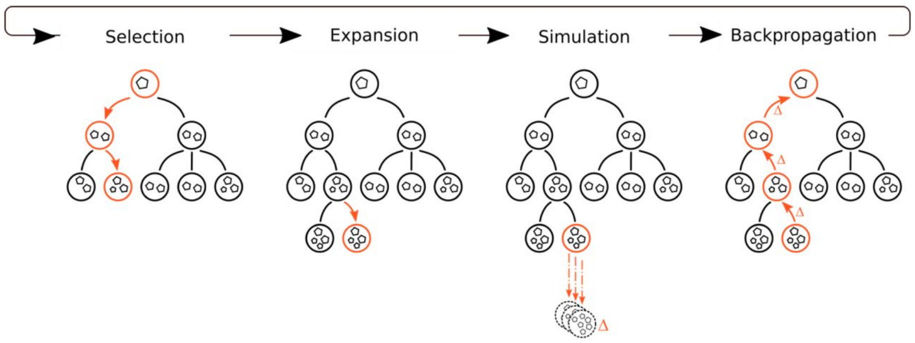


Figure 3.3: Steps of Monte Carlo Tree Search [4]

As a result, instead of visiting and evaluating many many paths to find the best path, as in the case of mini-max, MCTS uses RL and the policy to choose the best path from the current state in the tree.

3.6 Convolutional Neural Network

Convolutional Neural Network(CNN) is a special type of neural networks mostly used in areas such as image recognition and classification. Many algorithms and applications successfully used CNN such as face recognition [19], sentiment analysis [20], action recognition, and much more. Many different architectures of CNNs were proposed in the last few years such as LeNet, AlexNet, VGGNet, GoogLeNet, ResNet, ZFNet.

In CNN, the basic idea is to extract the features in the input record, then use the extracted features to understand and classify the data, This reduces the input data into small features using two operations (layers) called ‘convolution’ and ’pooling’.

CNN architecture overview

As shown in figure 3.4, CNN has four components or layers:

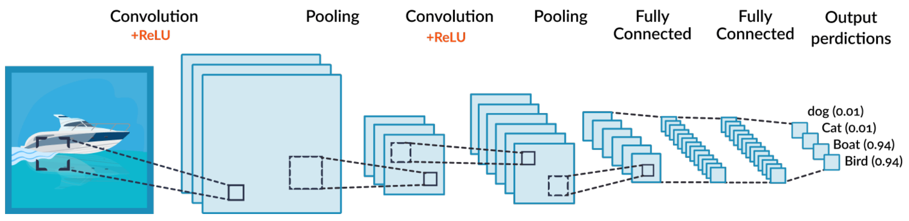


Figure 3.4: Architecture overview of Convolutional Neural Network (CNN) [5]

- **Convolution layer**: The objective of this layer is to extract the low-level features from the image such as edges. It is done by using *Filter* (sometimes called Kernel), a filter is passed over the image, and each time we move the filter we perform a matrix multiplication operation between the filter and the portion of the image over which the filter is hovering. The results are summed up into one number that represents all the pixels the filter observed. In the end, this process generates a matrix that is much smaller in size than the original input image.

- **Activation layer**: In this layer, an activation function used to format the matrix introduced in the previous step, such activation function is a *Rectified Linear Unit (ReLu)*. The ReLU function returns x for all values of $x > 0$, and returns 0 for all values of $x \leqslant 0$ as shown in figure 3.5.

- **Pooling layer**: The goal of this step is to down-sample and reduce the size of the matrix which makes the training of the network much faster by focusing on the most important information in each feature of the image. The most common type of pooling layer is a max-pooling; A filter is passed

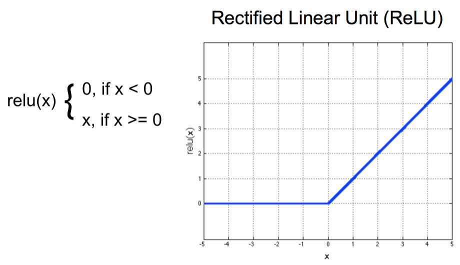


Figure 3.5: Rectified Linear Unit (ReLu) [5].

over the matrix and selects the max number out of each group of values. figure 3.6 show an example of max-pooling.

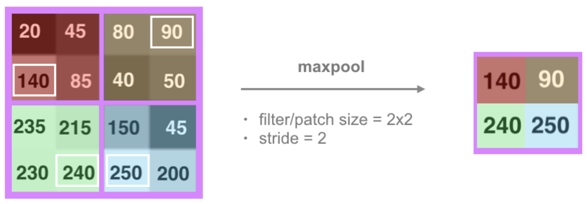


Figure 3.6: Example of max-pooling layer [5].

- **Fully connected layer**: At the end of the CNN, a traditional neural network is used to classify or give the probabilities of labels, it takes the output of the previous layers as input, and output a list of probabilities for different possible labels attached to the input.

In this work, we utilize both the MCTS algorithm and CNN to train the agents on how to play the Seega game. Since CNN is a supervised learning algorithm, and since we don't have a prepared dataset for the Seega game we will rely on MCTS to generate enough datasets and provide them to the CNN. The RL is used to improve the policy in MCTS. In the next chapter, we will explain how we used MCTS and CNN in mode details.

3.7 Related work

3.7.1 AI in board games

Chess is is the most widely-studied game in the history of artificial intelligence. Many types of research were done using different methods to develop a computer program that can defeat the human in this game. The most interesting methods were these methods that use deep learning such as David et. al. 2016 [6]. In this work, the researchers proposed a new deep learning model refereed to as *DeepChess*. This model is trained using supervised learning on a large data-set that contains millions of chess games, the training is done without any knowledge of the game rules. In their paper, the researcher developed an evaluation function that takes a chess position as input and returns a heuristic value or score as output. This evaluation function will be used to compare positions. They used this concept to propose a novel training method to train a model that receives two chess positions as input and predicts which position is better. The proposed training methodology consists of two stages. In the first stage, they trained a deep auto-encoder (refer to as *Pos2Vec*) using a data-set of millions of samples. This auto-encoder is used as a feature extractor which converts a given chess position into a vector of features. In the second stage, they trained a neural network to predict which of the two input positions will result in a win. The Architecture of these networks is shown in figure 3.7. In this work, the team used alpha-beta search algorithm[21] to select the right position from the search tree. They used a modified version of alpha-beta that does not require any position scores for performing the search, instead, it depends on *DeepChess* to compare and select the right position.

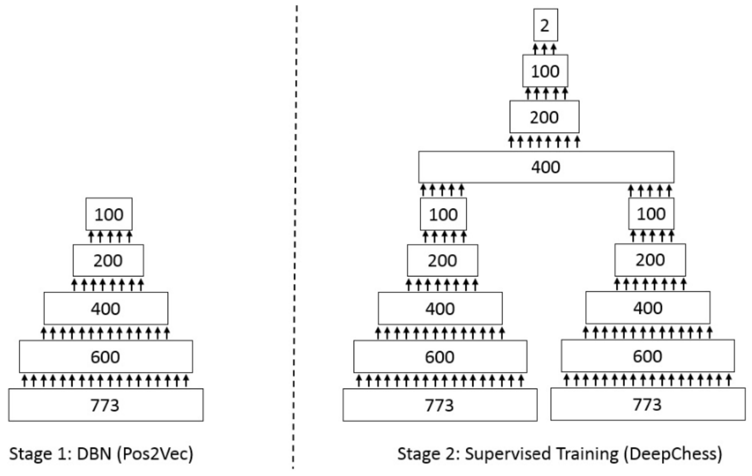


Figure 3.7: Architecture illustration of DeepChess [6].

Another game that got the interest of the researchers is the game of *GO*. GO is very old two-player board game that originated from China and is still played today. The game consists of a 19x19 board, and each player controls one color of stones (black/white) and places them on the board. Each player will try to surround his opponent's stones with his stones or surround the empty cells to make points of territory. Figure 3.8 show an example board from Go game.

Lately, some research tried to teach the computer to play the Go game, such research is the work of Silver et. al. 2016 [7]. The team in a company called deepmind (later acquired by Google) developed a program that masters the *Go* game and beat the human. AlphaGo became the first program to defeat a world champion in the game of *Go*. They used Monte-Carlo search tree [22] to simulate and play thousands of Go games. To do the simulation and compare the players progress, the researchers developed two neural networks, the first one is *value network* that evaluates the board positions and calculates a value to rate the board, and the second one is the *policy network* to select the best moves.

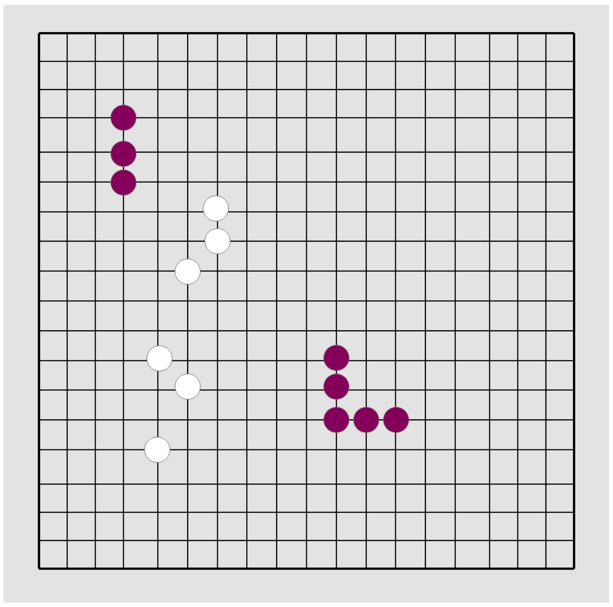


Figure 3.8: Go game board example, each player can position his tile with the same color (black/white).

Then they needed to develop a new search algorithm that combines Monte-Carlo simulation with the *value* and *policy* networks. In this work, the researchers used data collected from expert human players to train the *policy network* using supervised learning (SL). The SL trained *policy network* used in the early stages of training the player, then they trained the *policy network* using reinforcement learning (RL), which improves the SL policy network by optimizing the outcome of games of self-play. In the end, they used the reinforcement learning (RL) *policy network* to train the *value network* by playing many games against itself. This results with a *value network* that can predict the winner of the games. Figure 3.9 shows the neural network training pipeline and architecture used in the AlphaGo program.

Later in 2017, the team improved there work in new research [23]. They trained an RL agent to play the Go game by itself, unlike there previous work [7]

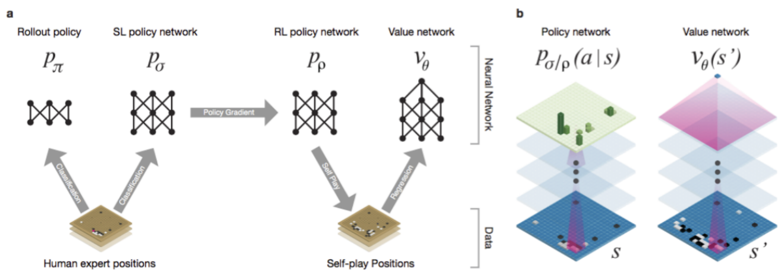


Figure 3.9: The neural network training pipeline and architecture used in the AlphaGo program.[7]

in which the network was trained on data collected from human players. They introduced a new algorithm called *AlphaGo Zero* which uses one neural network to replace both *policy network* and *value network* used in [7]. The network *AlphaGo Zero* trained solely by self-play RL, starting from random play, without any supervision or use of human data. this methodology no longer depends on *Monte- Carlo* search. It uses another simpler search tree that is built within the same neural network to evaluate positions and select moves. As a result, this new model *AlphaGo Zero* defeated the previous versions of *AlphaGo*, which were trained on a data-set of human data using handcrafted features.

The same team continued the improvement of the algorithm in 2018. In this paper [24], the researchers proposed a new algorithm called *AlphaZero* that considered a general version of there previous algorithm *AlphaGo Zero* that was introduced in [23]. This new algorithm can play games of chess and shogi [25] as well as Go using the same network architecture for all three games. It is trained using self-play games which started randomly without any human data or any handcrafted evaluation functions.

AlphaZero uses a deep neural network to replace the traditional handcrafted features and game-specific rules. This neural network takes the board position as input and returns a vector of probabilities of moves for each action. It also returns the expected estimated result of the game from a position. The weights in this deep neural network are trained using reinforcement learning from self-play games, starting from randomly initialized parameters.

This algorithm uses a general-purpose tree search algorithm based on Monte Carlo tree search (MCTS) algorithm that searches a series of simulated self-play games which in result return a vector representing a probability distribution over moves.

3.7.2 Work on Seega game

Few researchers tried to develop a computer program that can play Seega game. In 2013, abdelbar et. al [11] was the first attempt to solve the problem of Seega game. They proposed a methodology that is based on mini-max algorithm to search and select the best move for each board state. They defined a set of features for the board, some of these features are atomic and the others are compound and calculated from the atomic features. Based on these features they can calculate the heuristic of the board for each player at any given time. In their paper, they used 5×5 board.

They proposed 13 features for the board. These features have different weights. a particle swarm optimization algorithm [26] is used to generate the weight vector for these features for stage-1 and stage-2 of the game.

The proposed features are:

- Corners (f_1): Corner domination.
- Boarders (f_2): Border domination.

- Clustering (f_3 .. f_6): This feature counts adjacent player stones horizontally and vertically, (f_3, f_5) are the vertical for player1 and player2 and (f_4, f_5) are the horizontal for player1 and player2 and
- Massdist (f_7, f_8): In this feature, the center of mass is calculated for each player then the magnitude of the difference is returned for horizontal (f_7) mass-distance and for vertical (f_8) mass-distance.
- Entrapment (f_9, f_{10}): This feature gives a measure of how a color dominates the outer part of the board, (f_9) is used for horizontal and (f_{10}) for vertical.
- Material (f_{11}): This feature computes the difference between the number of black and white stones on the board.
- Phase two starts (f_{12}): This feature reflects how many captures player1 will make on the first move in phase two.
- Black can start (f_{13}): This feature returns 0 if the four squares around the middle square are occupied by White, and returns 1 otherwise.

These features are used by higher-level configurations through addition and/or multiplication. The authors proposed two different equation using these features for phase-one and phase-two of the game as follow:

For phase one, [$s_1 = (\overrightarrow{w_1})^T \overrightarrow{c_1}$]

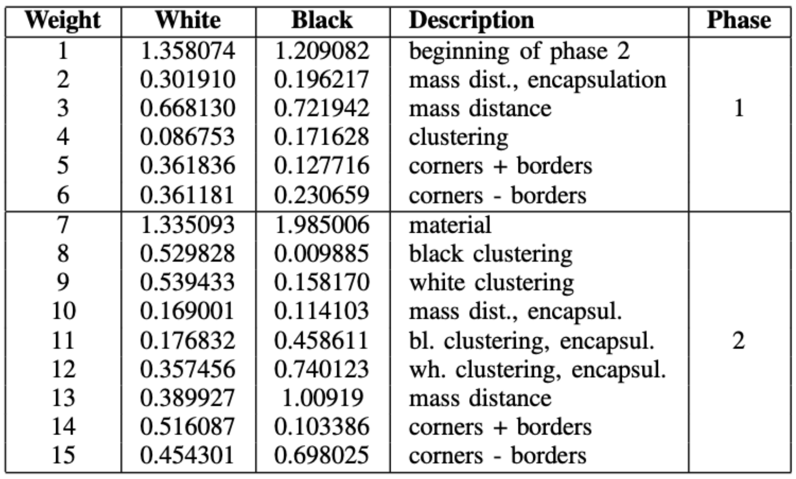
and for phase two: [$s_2 = (\overrightarrow{w_2})^T \overrightarrow{c_2}$]

where s_i is the board score, $\overrightarrow{c_i}$ is the vector of compound features, and $\overrightarrow{w_i}$ is the weight vector.

The $\overrightarrow{w}$ vectors that is used in phase-one and phase-two is calculated and optimized using Co-Evolutionary Particle Swarm Optimization algorithm.

Later, in 2004, the same team continued their work in [8]. In this work, the researchers expanded and improved the program, they used the same methodology and same set features that proposed in [11] on a larger board 7x7. A small change was done on the swarm configuration to improve the performance and reduce the computation time. As a result, they proposed a set of weights for each feature in phase-1 and phase-2 as shown in table 3.10.

Figure 3.10: Table shows the weight vectors for white and black players [8].

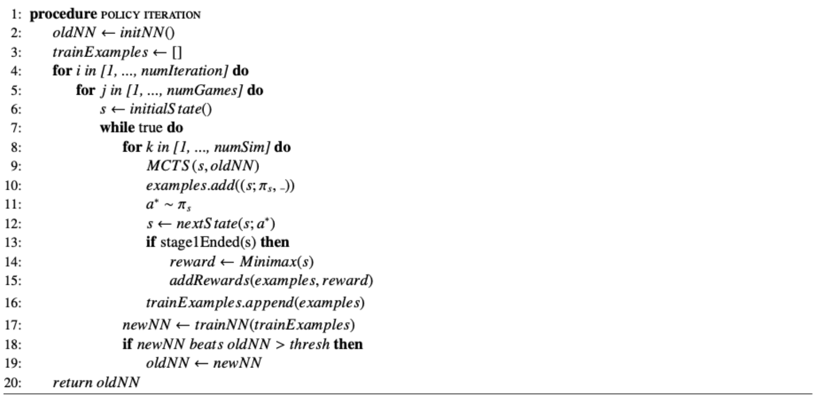


Weight	White	Black	Description	Phase
1	1.358074	1.209082	beginning of phase 2	
2	0.301910	0.196217	mass dist., encapsulation	
3	0.668130	0.721942	mass distance	1
4	0.086753	0.171628	clustering	
5	0.361836	0.127716	corners + borders	
6	0.361181	0.230659	corners - borders	
7	1.335093	1.985006	material	
8	0.529828	0.009885	black clustering	
9	0.539433	0.158170	white clustering	
10	0.169001	0.114103	mass dist., encapsul.	
11	0.176832	0.458611	bl. clustering, encapsul.	2
12	0.357456	0.740123	wh. clustering, encapsul.	
13	0.389927	1.00919	mass distance	
14	0.516087	0.103386	corners + borders	
15	0.454301	0.698025	corners - borders	

They tested this program by playing against human players who started learning and practicing the game of Seega for few weeks, the program behavior during playing showed that it has a strategy in positioning and moving the stones and not random. The program won some games and lost others.

In 2020, Aljaafreh et. al in their work [9] developed a computer program that can play the first phase of Seega game using mini-max and deep RL. Their methodology is divided into two steps. In step-1 they trained an agent to play the *Positioning stage* of the game. They used the methodology proposed in *AlphaZero* [23] to train the agent by implementing self-play reinforcement learning.

In this approach MCTS is used to run and simulate self-play games. The data collected from MCTS is used in training the neural network. In step-2, they used min-max to play *Moving stage* of the game, they run mini-max algorithm on the board generated in step-1. The depth of the mini-max tree that is explored is limited; max number of actions to finish the game is 50 according to Seega expert. Figure 3.11 shows an overview of Policy Iteration Algorithm used in this work.



```
procedure POLICY ITERATION
    oldNN ← initNN()
    trainExamples ← []
    for i in [1, ..., numIteration] do
        for j in [1, ..., numGames] do
            s ← initialState()
            while true do
                for k in [1, ..., numSim] do
                    MCTS(s, oldNN)
                    examples.add((s; $\pi_s$, _))
                    $a^* \sim \pi_s$
                    s ← nextState(s; $a^*$)
                    if stage1Ended(s) then
                        reward ← Minimax(s)
                        addRewards(examples, reward)
                    trainExamples.append(examples)
                newNN ← trainNN(trainExamples)
                if newNN beats oldNN > thresh then
                    oldNN ← newNN
    return oldNN
```

Figure 3.11: Policy Iteration Algorithm used in [9]

3.8 Tools and Frameworks

Machine learning is a complex discipline. This encouraged many big companies such as Google, Microsoft, IBM and AWS to offer machine learning frameworks and API's to make it easy for developers and data scientists to experiment ML in its different steps including the process of acquiring data, training models, serving predictions, and refining future results.

Many good frameworks are available nowadays with, different concepts and architectures that fit most needs of the people in this field. Examples include Theano [27], Scikit Learn [28], Caffe framework [29], but the most popular framework used in development and production nowadays is Google's TensorFlow [30] which we used in this thesis.

Tensorflow

Figure 3.12: Tensorflow logo.

TensorFlow is an open-source library for numerical computations and large-scale machine learning developed by the AI team inside Google to be used in their products such as search engines, images search, and other services. This library offers easy and standard ways to build machine learning models especially neural networks and deep learning models. It also contains many pre-built ML algorithms that can run on CPU and GPU. In this framework the developers and researchers can use the most popular language in this field which is Python to write their program, while the framework executing those applications in high-performance C++.

TensorFlow allows developers to create their program using dataflow graphs. Each node in the graph represents a mathematical operation (eg: $wx + b$), and each edge between nodes is a multidimensional data array called tensor which most the time represents the weights.

Figure 3.13 shows the architecture of the framework, Tensorflow provides high-level APIs: Keras and Estimator for creating deep learning models. Then, tf.data and other APIs for data pre-processing. At the lowest level, each Tensorflow operation is implemented using a highly efficient C++ code.

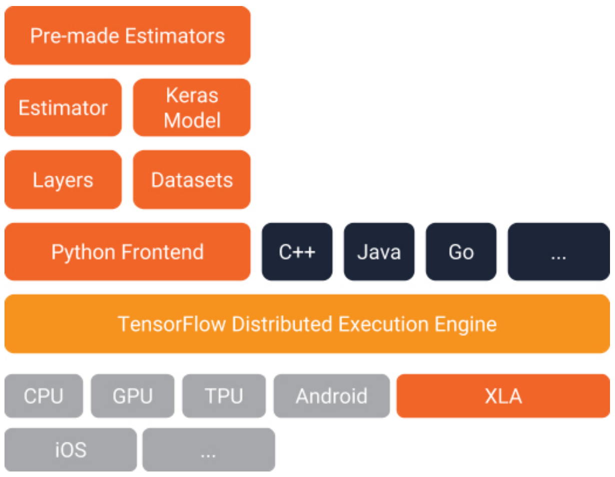


Figure 3.13: Tensorflow 2.0 architecture [10].

Most of the time, we use high-level APIs only, but when we need more flexibility like handling the tensors directly, we can use lower-level APIs like tf.nn, tf.GradientTape(), etc. Tensorflow provides extensive libraries like TensorBoard for visualization, Tensorflow Hub for downloading and reusing pre-trained models [10].

Keras

Keras[31] is a python library that provides high-level API to write deep learning applications, it allows developers and researchers to design, develop, train, evaluate, and run most sorts of deep neural networks. In this thesis, we used Keras as front-end and Tensorflow as back-end to perform the experiments.

Chapter 4

Methodology

In this chapter, we will discuss the methodology used in the study, the stages by which the methodology was implemented, and the research design. First, we will introduce the representation of the environment we used, then we will explain the architecture of the environment and the neural networks used. Finally, we will discuss the tools and evaluation methods used in this study.

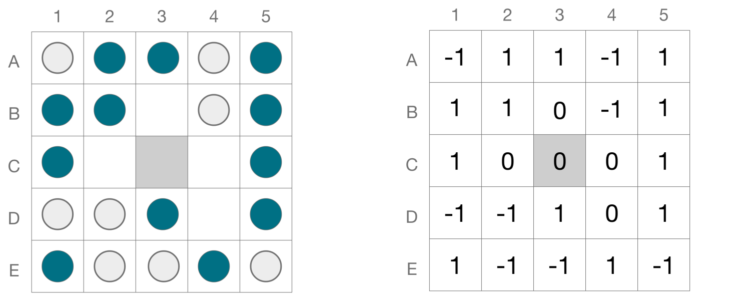
As we discussed earlier in this thesis, most of the previous researches and work on the development of automated computer gaming. Especially in the field of board games are based on training using data-sets of human play, but in our case, such data set is not available for the game of Seega. This makes the reinforcement learning on self-play games the best method to train our agents.

Seega game has two stages, each stage has different rules, input, and output. For this reason, we will develop two separate models for each stage of the game; namely *Moving Agent* and *Positioning Agent.* The training process will be divided into two steps, first, we will train the *Moving Agent* using self-play games on a randomly-generated board; in which we will train the agent to select the best moves of the pieces to win the game and predict the winner of the game. Then in the second step, we will train the *Positioning Agent* using self-play games starting from an empty board to select the best cell to place the pieces in the perspective of the current player. The trained *Moving Agent* will also be used during the training of *Positioning Agent* to predict the winner of the game when the players finish positioning all their pieces instead of playing the full game.

4.1 Environment Representation

The Board:

The environment of the game is basically the board; which will represent the state of the game s at a given time t, we used a two-dimensional array $(n \times n)$ to store the game state, we used the value 1 for player-1 pieces, the value -1 for player-2 piece's, and 0 for the empty cell of the board. Figure 4.1 shows an example of game board with its internal representation values. In this example $n = 5$.



	1	2	3	4	5
A	-1	1	1	-1	1
B	1	1	0	-1	1
C	1	0	0	0	1
D	-1	-1	1	0	1
E	1	-1	-1	1	-1

A: the board UI as shown to the players

B: The internal representation of the board,
- "0" for empty cell,
- "1" for player1 pice,
- "-1" for player2 pice.

Figure 4.1: The Seega board representation.

The Actions:

The actions are different for each stage, In the *Positioning Agent* the actions will be the index of the cell in the form of (x, y), and the value will be +1 for player-1 and -1 for player-2. We converted the (x, y) into a linear value *index* to be used in the output layer of the neural network using the following equation.

$$index = x \times n + y \tag{4.1}$$

In the *Moving Agent*, the actions are in the form of $(index, direction)$, the *index* is the selected cell position and calculated by applying equation 4.1 on cell (x, y), $direction \in [0, 1, 2, 3]$ is the direction of the moving, 0=UP, 1= DOWN, 2= LEFT, and 3= RIGHT.

4.2 RL Architecture

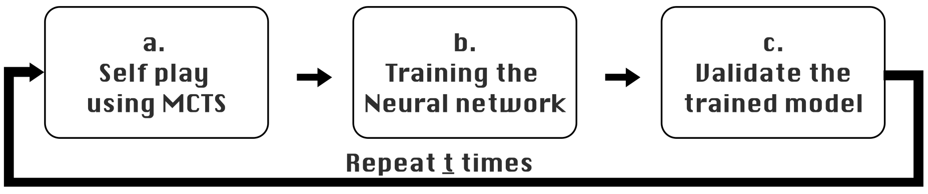


Figure 4.2: Overview of Self-play reinforcement learning in Seega.

Our work is inspired by the ideas from AlphaZero [24]. We applied them during the training of the *Moving Agent* and the *Positioning Agent*. Figure 4.2 show the general architecture of the process of reinforcement learning used in the training of the two models.

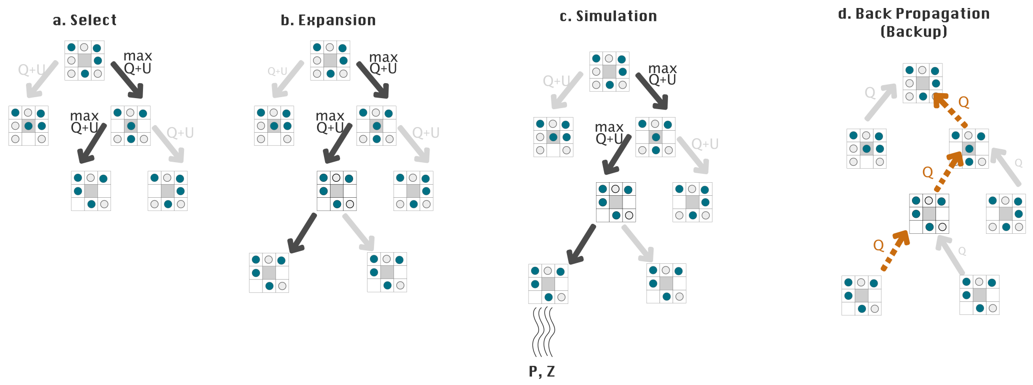


Figure 4.3: MCTS Steps implemented on Seega game.

The process has a loop of three steps. **Step a** is the core of the reinforcement learning in which MCTS interacts with the environment and plays many self-play games. MCTS internal steps are shown in figure 4.3. It starts with a random policy to select the best action a for the current state s_t, and with time the algorithm will explore and simulate more states to improve the accuracy of its policy. MCTS will continue playing until it reaches the end of the game and a winner z is found at the end. The algorithm will store all visited states s_t with its selected best action a_t and winner z to be used in the training of the deep neural network. Figure 4.4 show the building blocks of the process done in this step. MCTS algorithm is discussed in detail in section 3.5.

In **step b**, a neural network is trained using the data collected in step a. The goal of this step is minimize the error between the predicted values from the network and the correct values from the MCTS. The training data is stored in the form (s_t, $\overrightarrow{\pi}$, z) where s_t is the game state, $\overrightarrow{\pi}$ is the probabilities of each action for a given state, and z is the winner of the game. The trained network will be used to predict the probabilities of each action for a given state P_t and the winner of the game z.

In **step c**, the newly-trained model will be validated, it will play against the previously trained model for the number of k games. If it wins 60% of the games it will be accepted and used in the next iteration, if not, it will be rejected and the old model will be used in the next iteration.

The trained network will be used as a policy in MCTS in the next iteration. MCTS has a policy that determines the best path or action to take at a given state, this policy will improve by time by using the trained and accepted NN model as a policy, this will result in better examples in the next iterations.

This general process will be used in training the two agents (*Moving Agent*, and *Positioning Agent*) for this game with some customization that will be discussed later in this chapter.

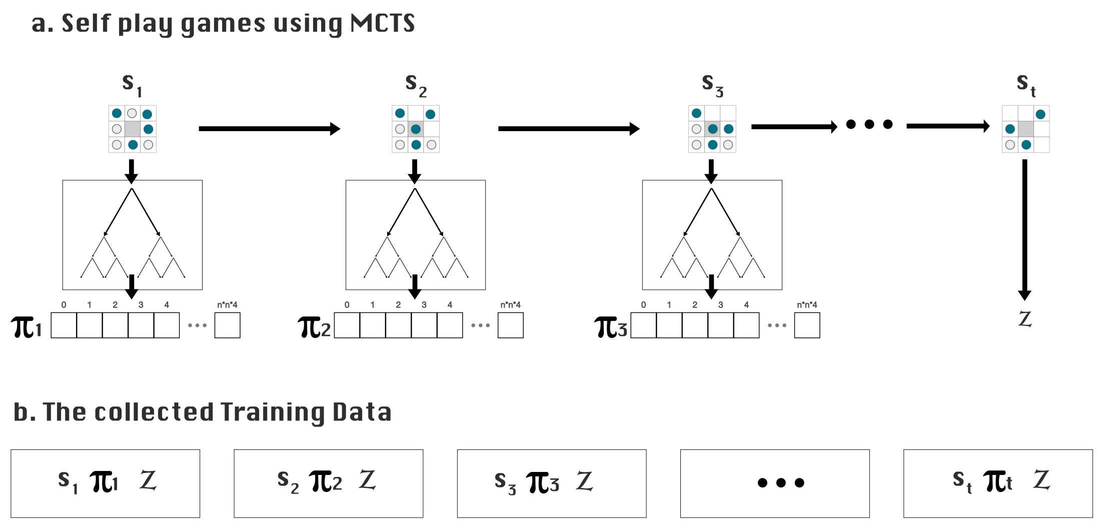


Figure 4.4: **MCTS self-play**, **a** MCTS play until the end of the game. **b** the self play data is collected to be used to train the NN. where s_t is the board state, π is the probabilities of actions at each state, z is the winner of the game.

4.2.1 Training The Moving Agent

The goal of this step is to develop a neural network f_0 and tune its θ parameters to be able to play stage-2 of the game and predict the best move to take in a given state s_t. This neural network will take as input a board state s_t, and outputs two values:

- **Policy** $\vec{P}_\theta(s_t)$: A vector of probabilities over all possible moves for the state s_t.
- **Value** $v_\theta(s_t)$: The continuous value $v_\theta(s_t) \in [-1, 1]$ is the expected winner of the game given this state s_t, this value will be used later in training of *Positioning Agent*.

Training steps:

At the beginning, the network weights θ is initialized randomly, The Monte Carlo tree search algorithm (MCTS) is used to simulate and run a lot of self-play games and provide training examples to the neural network to be trained on.

The MCTS generated examples have the form $(s_t, \vec{\pi}, z_t\ )$ where s_t is the starting state of the game (current state), $\overrightarrow{\pi}$ is the improved policy estimate after running MCTS, z_t is the winner of the game, and t is the iteration.

During the training and at each epoch, the input state s_t is provided to MCTS and the neural network to get the results which include the probabilities of actions p and the winner of the game z, after they finish the process we compare the MCTS search probabilities π_t and the neural network policy vector $\vec{P}_\theta(s_t)$ and update the weights θ of the neural network to decrease the difference between its result and the MCTS output and minimize the error between the predicted winner v_t form the neural network and the game-winner z from self-play MCTS.

Gradient descent is used to adjust the *neural network* weights based on a loss function L that sums over mean-squared error and cross-entropy losses respectively, The following equation is used to compute the loss function L

$$L = (z - v)^2 - \pi^T logp + c||\theta||^2 \tag{4.2}$$

where z is the winner of the game from MCTS, v is the predicted winner from the neural network, p is The vector of move probabilities, and c is a parameter controlling the level of L2 weight regularisation (to prevent over-fitting).

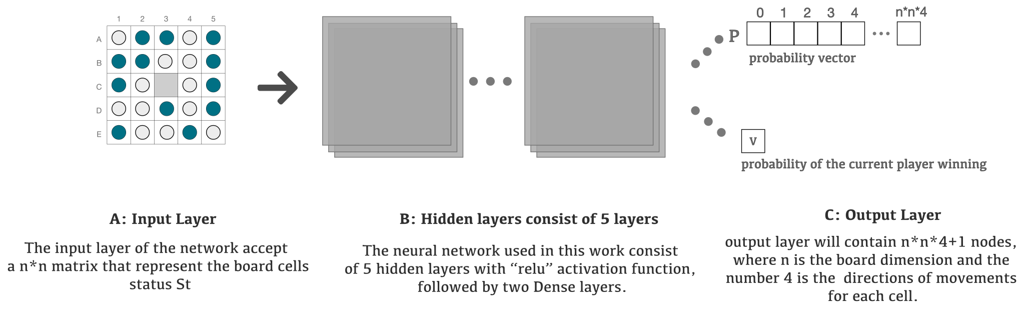

1 2 3 4 5
A B C D E
P
0 1 2 3 4 ... n*n*4
probability vector
v
probability of the current player winning
A: Input Layer
The input layer of the network accept a n*n matrix that represent the board cells status St
B: Hidden layers consist of 5 layers
The neural network used in this work consist of 5 hidden layers with "relu" activation function, followed by two Dense layers.
C: Output Layer
output layer will contain n*n*4+1 nodes, where n is the board dimension and the number 4 is the directions of movements for each cell.

Figure 4.5: The architecture of deep neural network used in training the *Moving Model*.

As shown in figure 4.5, the neural network used in this work consists of 5 layers CNN network with *relu* activation function, followed by two Dense layers.

The input layer of the network accepts a $n \times n$ matrix that represents the board state s_t, the output layer will contain $n \times n \times 4$ nodes, where n is the board dimension and the number 4 is the directions of movements for each cell,

$direction \in [0,1,2,3]$. For example, if we used 5×5 board then the input layer will have 25 nodes, and the output layer will have 25×4 nodes. The index of each node *nid* in the output layer is calculates using the following equation which generates the index based on the action $(index, direction)$ pair.

$$nid = index + direction \tag{4.3}$$

Later, we will use the node *nid* with the highest probability to be the best action for the state s,

The node index *nid* is used as the action, when the player executes the action it is needed to convert *nid* to the form $(index, direction)$, *nid* can be used to extract the board cell index and the direction using the following equations:

$$index = \left\lfloor \frac{nid}{4} \right\rfloor \tag{4.4}$$

$$direction = nid \mod 4 \tag{4.5}$$

For example, suppose that the network predicted node $nid = 45$ to be best action, then board cell index $index = \lfloor \frac{45}{4} \rfloor$, and $direction = 45 \mod 4$, this mean we need to move cell index=11 to Right. The index also can be formatted as (x, y) using the following equation:

$$x = \left\lfloor \frac{index}{n} \right\rfloor, \quad y = index \mod n \tag{4.6}$$

4.2.2 Training The Positioning Agent

At this stage we will use the same approach used in the training of *Moving Agent*, we will use MCTS to simulate self-play games and generate a lot of games data to be used to train the neural network.

The goal of this step is to train a neural network to play the stage-1 of Seega game in which the agent needs to select the best positions to place its pieces on the board that leads to winning the game at the end of the moving stage.

As shown if figure 4.6, The neural network used in this step consist of 5 CNN network with *relu* activation function, followed by two Dense layers.

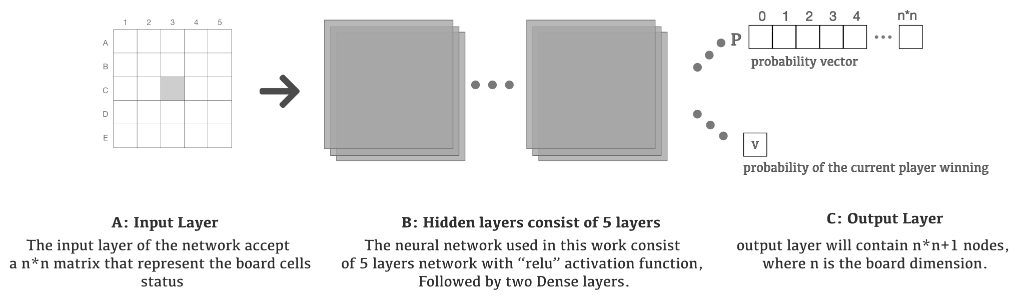

1 2 3 4 5
A
B
C
D
E
0 1 2 3 4 ... n*n
P
probability vector
V
probability of the current player winning
A: Input Layer
The input layer of the network accept a n*n matrix that represent the board cells status
B: Hidden layers consist of 5 layers
The neural network used in this work consist of 5 layers network with "relu" activation function, Followed by two Dense layers.
C: Output Layer
output layer will contain n*n+1 nodes, where n is the board dimension.

Figure 4.6: The architecture of deep neural network used in training the *Positioning Model.*

The input layer of the network accepts a $n \times n$ vector that represents the board cells status s_t, the output layer will contain $n \times n$ nodes. The index of each node *nid* in the output layer represents the cell index in the board.

We used the same process shown in figure 4.2 to train this agent, some modification is added in this step in the implementation of the neural network which is in the output layer, the output layer consists of $n \times n$ node as shown in figure 4.6.

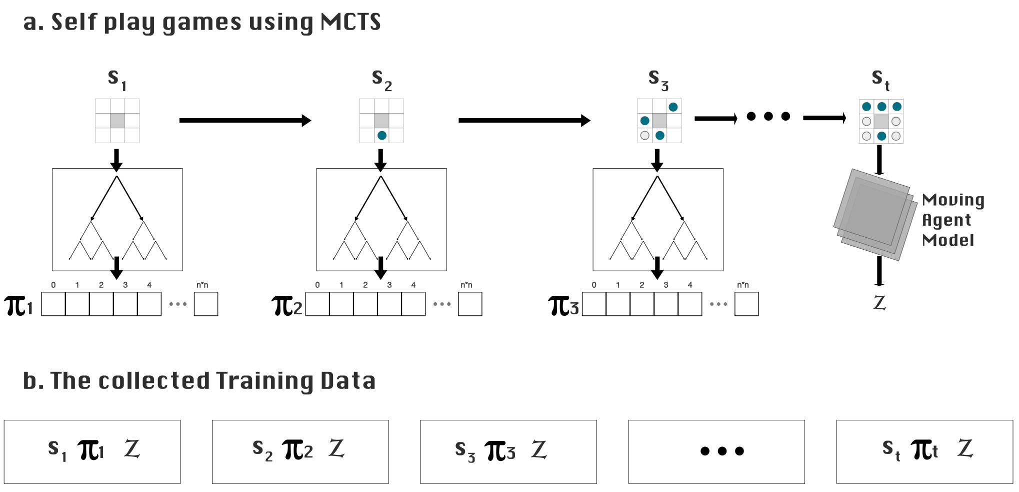


Figure 4.7: **MCTS self-self play**, **a** MCTS play until the board is full of pices, then the board is passed to the neural network which previously trained to predict the winner z of the game. **b** the self play data is collected to be used to train the NN. where s_t is the board state, π is the probabilities of actions at each state, z is the winner of the game.

Another modification is done in MCTS, As shown in figure 4.7. The algorithm will run self-play game starting from an empty board, when the algorithm reaches a board state s_t in which all cells are filled, it will use the previously trained *Moving Agent* to predict the winner of the game. This step will improve the speed of training the model since we do not need to play the moving stage of the game to calculate the winner.

The data collected in this step will be used to train the Agent as discussed earlier in this chapter.

4.3 Evaluation Criteria

Elo is a rating system that is named after its creator Arpad Elo. It was first introduced in 1960 as a rating system for chess [32]. Later it became widely used to rate players in other games such as football[33], video games[34], and other board games like Go[35]. This system is used to rank players, also it can be used to predict the winning probabilities for the players.

Elo system is easy to understand and to use and it is a standard rating all over the world. The system mainly has two steps:

Step 1: Score Prediction, the Elo system can predict the winner and expected score of a match based on the difference in rating between two players. For example, if A score is higher than B, then A is more likely to win. Elo system uses a logistic function to transform the difference between A and B rating into a "score estimation", the result of the logistic function E will be between 0 and 1. The greater value means more probability to win. The following equations are used to calculate the estimated match result for player-1 E_A and player-2 E_B.

$$E_A = \frac{1}{1 + 10^{(R_B - R_A)/400}} \tag{4.7}$$

$$E_B = \frac{1}{1 + 10^{(R_A - R_B)/400}} \tag{4.8}$$

where E_A is the estimated result for player-A, R_A is the current rating for player-A, and R_B is the current rating for player-B. The estimated values will be used in the next step to update the rating for the players.

Step 2: Update the players Rating: After the end of the match, we compare the estimated result E_A with the actual results R_A, if $R_A > E_A$ this means that the system undervalues the player A so we need to increase his rating. But if $R_A < E_A$ then the system overvalues the player A we need to decrease his rating. The same comparison is done for player B. In the Elo system, we use the following equation to update the players rank:

$$R'_A = R_A + K(S_A - E_A) \tag{4.9}$$

$$R'_B = R_B + K(S_B - E_B) \tag{4.10}$$

where R'_A, R'_B are the updated rank for the players, R_A, R_B are the old ratings of the players before the match, S_A, S_B are the score of the match for the players, E_A, E_B the the predicted score for the players(calculated in step1 of Elo system), and K (called K-factor) is a constant used to determine the speed with which the player rate is adjusted after each game. for example in chess $K = 16$.

The Initial Score for the players can be any value such as *zero*, but most developers avoid *zero* value and use a standard number such as 500 to prevent negative rating for the players.

In this work, we will use Elo rate the agents in Seega, it will be used during the training and evaluating of the agent. After each iteration of the training, we will calculate the Elo rating for the new agent by playing many games against the agent from the previous iterations, based on the results of these games we will give the new and old agents the Elo rating.

Chapter 5

Experiments and Results

Summary

This chapter will show the experimental results of our proposed Reinforcement Learning approach for the game of Seega. First section 5.1 describes the computing environment and experimental settings that were used to conduct this work. Then, section 5.2 shows the results of training the *Moving Agent*, Finally, in section 5.3 we will describe the results of training the *Positioning Agent*.

5.1 Computing environment

All the machine learning algorithms, that were used in our experiments, were implemented using Python 3.7 under Ubuntu 20.04 with intel core i7-8700U CPU 3.20GHz, 32GB RAM memory shipped with Nvidia GeForce GTX 1050 GPU card that comes with 640 NVIDIA CUDA Cores, 2 GB GDDR5 and 7 Gbps Memory Speed.

Python is widely used in the field of machine learning. Many libraries are developed and tested in this language. In our experiments, we have used Keras on top of Tensorflow 2.1 library in order to implement the neural networks. Tensorflow core engine is built using C-language which make it very efficient and can utilize the GPU, it provide the developers a high-level python interface to develop and configure many different types of neural networks, more details about this tool in chapter-4.

5.2 Moving Agent Training Results

This section will show the results of training our *Moving Agent* which is trained to select the best move that leads to winning. We implemented the proposed reinforcement learning methodology which was described in the previous chapter to train our *Moving Agent*. The Training took approximately 2.5 weeks, started from completely random behavior, and continued without human intervention.

We developed the convolution neural network which its architecture consist of 5 layers ($25 \times 512 \times 512 \times 512 \times 101$) and total parameters is 8,197,221. During the training, the MCTS algorithm played 15 iterations and generated around 1.7 million of self-play games. At the end of each iteration, we saved a checkpoint of the agent and evaluate the new neural network checkpoint against the best-reached network $f\theta$ so far. As shown in the table 5.1, the newly trained model played against the best model reached so far 70 games, if the new model wins 0.6 of the games we accept it and replace the best model.

The plot in figure 5.1 shows the performance of the agent from each iteration of learning of the stage-2 of the game. Elo ratings were computed from the result

Table 5.1: The training results for the Moving Agent.

iteration	samples	new-model wins	old-model wins	accepted?
1	58,542.00	35	35	NO
2	105,231.00	33	37	NO
3	142,020.00	32	38	NO
4	210,393.00	37	33	NO
5	236,467.00	28	42	NO
6	274,192.00	37	33	NO
7	314,434.00	42	28	YES
8	453,873.00	41	29	NO
9	653,658.00	41	29	NO
10	812,582.00	44	26	YES
11	993,996.00	43	27	YES
12	1,153,714.00	42	28	YES
13	1,292,100.00	43	27	YES
14	1,473,442.00	45	25	YES
15	1,639,451.00	42	28	YES

of playing evaluation games between different versions of the agent.

To test the agent in a real-world game, we generated a test board with its stones arranged on selected cells, this generated board has many available stones to move, the goal of the test was to show that the agent can select the best action a from a list of allowed actions A. Figure 5.2 **a** shows the testing board provided to the agent to predict the best move, in this example the allowed actions for the agent; assuming that the agent owns the black stone, are $A = [(B4, UP), (C4, LEFT), (D3, UP), (D3, DOWN), (E4, LEFT)]$.

When the example board provided to the agent to play its predicted best action a, the agent selected the action $(D3, UP)$ among all other allowed actions. Figure 5.2 **b** shows the board after the agent play it best predicted action which is $(D3, UP)$, this action enabled the agent to capture its opponent stone at $C2$ and in figure5.2 **c** the agent continue moving the same stone from $C3$ to $D3$ and

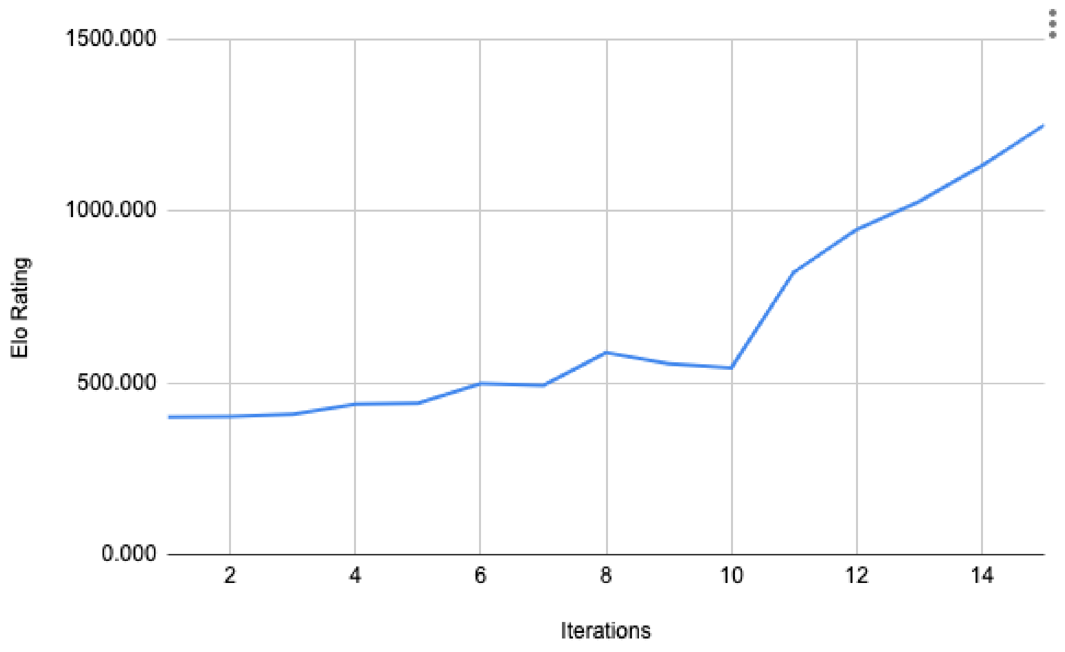


Figure 5.1: Elo rating per iteration for the *Moving Agent*

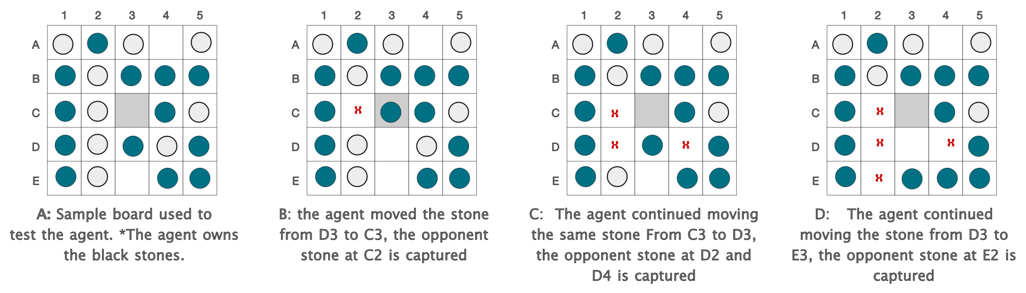

A: Sample board used to test the agent. *The agent owns the black stones.
B: the agent moved the stone from D3 to C3, the opponent stone at C2 is captured
C: The agent continued moving the same stone From C3 to D3, the opponent stone at D2 and D4 is captured
D: The agent continued moving the stone from D3 to E3, the opponent stone at E2 is captured

Figure 5.2: Example game to test the *Moving Agent*

captured two opponent stones at $D2$ and $D4$, and finally figure 5.2 **c** show the agent move the same stone from $D3$ to $D3$ and captures the opponent stone at $E2$, this make the total captured stone equal 4 from the same action.

The resulting board clearly shows that agent action a selected based on the knowledge it learned and it could lead the agent to win the game.

5.2.1 Positioning Agent Training Results

In this section we will discuss the results of training our *Positioning Agent* which trained to select the best board index to place its stone which could in result lead to win the game.

We used and implemented the same methodology used in training the *Moving Agent* in the previous section. The Training took approximately 1.5 weeks, started from completely random behavior, and continued without human intervention. The Players started from an empty board, and in turn, they placed two pieces at a time until all their stone placed at the board, to get the winner of the game we used the *Moving Agent* which trained in the previous section to predict the winner instead of playing the stage-2 of the game, this saved a lot of time and processing during the training. We developed the convolution neural network which its architecture consists of 5 layers ($25 \times 512 \times 512 \times 512 \times 25$) and total parameters is 5,251,217. Each CNN layer has 512 filter with a kernel size is 3, and *ReLu* is used as activation function.

During the training, the MCTS algorithm played 110 iterations and generated around 264,000 games of self-play. Same as in the previous model, at the end of each iteration, we saved a checkpoint of the agent and evaluate the new neural

iteration	**samples**	**new-model wins**	**old-model wins**	**accepted?**
1	2,400.00	35	35	NO
2	14,400.00	38	32	NO
3	16,800.00	45	25	YES
4	19,200.00	36	34	NO
5	21,600.00	35	35	NO
6	26,400.00	40	30	YES
7	28,800.00	29	41	NO
8	40,800.00	31	39	NO
9	43,200.00	41	29	YES
10	45,600.00	31	39	NO
11	48,000.00	39	31	YES
12	50,400.00	32	38	NO
13	52,800.00	33	37	NO
14	55,200.00	36	34	NO
15	57,600.00	39	31	YES

Table 5.2: The results of training the Positioning Agent.

network checkpoint against the best-reached network $f\theta$ so far. Table 5.2 shows a sample of the results which got during the training. You can find the full table in the appendix. It also shows for each iteration the number of samples and whether the new model is accepted or rejected. The newly trained model played against the best model reached so far 70 games, if the new model wins 0.6 of the games we accept it and replace the best model.

The plot in figure 5.3 shows the performance of the agent for each iteration of the learning process. Elo ratings were computed from the result of playing evaluation games between different versions of the agent.

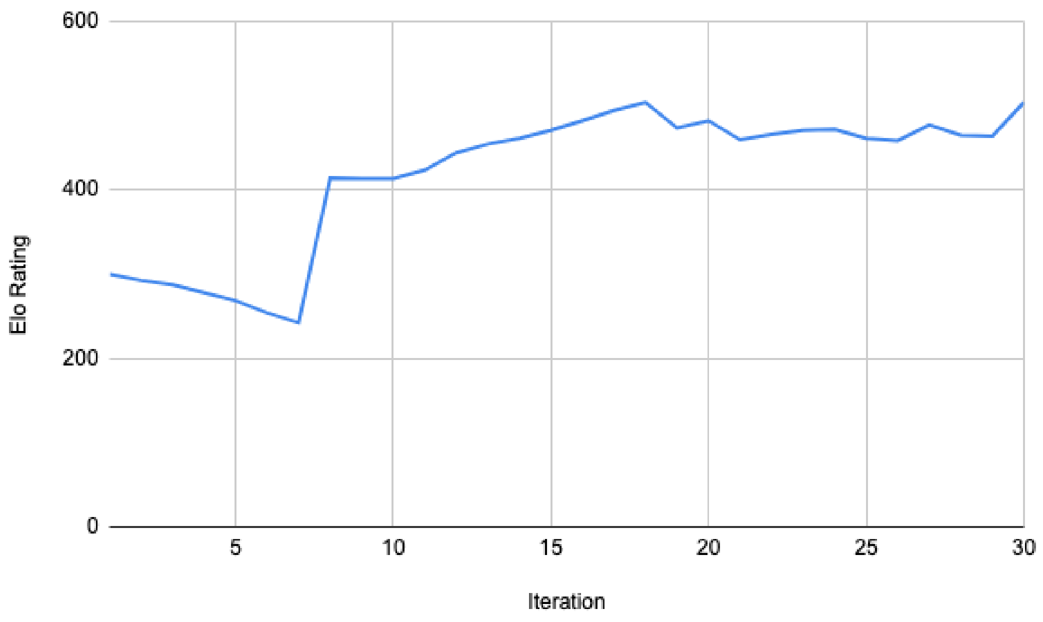


Figure 5.3: Elo rating per iteration for the *Positioning Agent*

Depth (level)	**Avg time(ms)**	**Avg time (minutes)**
2	66	0.0011
4	873	0.01455
6	48642	0.8107
8	2900940	48.349

Table 5.3: The computation time needed for mini-max at different levels of tree depth.

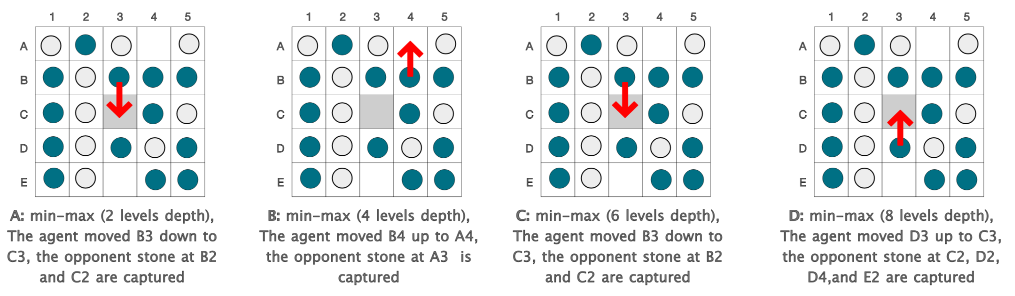
5.3 Minimax Results

We developed a mini-max version of *Moving agent* to test the computing time and the playing skills of the agent. Table 5.3 show the time needed to calculate and predict the best action using mini-max, we calculated the average time by running the algorithm with same setup 10 times, we tried mini-max with 4 setups, started with a 2-levels search depth of the tree, then increased the depth until 8-levels.

Depth (level)	move	# stones captured
2	(B3,Down)	2
4	(B4,Up)	1
6	(B3, Down)	2
8	(D3,UP)	4

Table 5.4: The predicted best actions by mini-max at different levels of tree depth.

We also tested the skills of mini-max to compare it with the agent we trained previously. The same board which used to evaluate the RF agent is used with mini-max. Table 5.4 shows the selected action and the number of opponent stones captured at each mini-max setup.



A: min–max (2 levels depth), The agent moved B3 down to C3, the opponent stone at B2 and C2 are captured

B: min–max (4 levels depth), The agent moved B4 up to A4, the opponent stone at A3 is captured

C: min–max (6 levels depth), The agent moved B3 down to C3, the opponent stone at B2 and C2 are captured

D: min–max (8 levels depth), The agent moved D3 up to C3, the opponent stone at C2, D2, D4,and E2 are captured

Figure 5.4: The actions taken by mini-max at different levels of tree depth.

Figure 5.4 illustrate the actions predicted by mini-max after running the algorithm with different tree depth. **A:** show min-max with 2 levels depth. The algorithm moved the stone from $B3$ down to $C3$ which resulted in capturing two of opponent stones at $B2$ and $C2$. **B:** show min-max with 4 levels depth, the algorithm moved the stone from $B4$ Up to $A4$ and the opponent stone at $A3$ is captured. In **C:** min-max with 6 levels depth is used, the algorithm moved its stone from $B3$ down to $C3$, the two opponent stone at $B2$ and $C2$ is captured. **D:** min-max using 8 levels depth, the algorithm moved the stone from $D3$ up to

$C3$, this action captured 4 of the opponent stone at $C2$, $D2$, $D4$, and $E2$.

5.4 Discussion

In this section, we will discuss the results of the proposed approach, The two agents are trained with random policy and without any human intervention. the goal for the two agents is to learn the rules of the game which prevent them from taking wrong actions such as placing the stone at the center cell of the board. Also to learn to take the best actions that leads to win the game and defeat the opponent player.

The results for both agents shown that they learned the rules of the game and their action based on the knowledge that they learned.

In the first few iteration shown in table 5.1 and table 5.2 depicts that the agents were defeated by the initial random policy since they didn't learned from enough samples. We adapted the 0.6 as threshold to accept or reject the newly trained model. To accept the agent it should win at least 60% of the games, this threshold prevent us from accepting a model that won randomly by chance.

Also, we used the Elo ranking system to evaluate the agents, as shown in figure 5.1 and figure 5.3, the agents skills are improved over time.

When compared to mini-max, the mini-max algorithm needed 48 minutes to get the same result of our trained model which needed 13 ms. As shown in table 5.3 the time needed to predict the action is dramatically increased when the depth of search is increased which makes this methodology not acceptable in the real world game. When using a small value search depth, the results of the algorithm will be naive and could not lead to winning the game since the algorithm explored only a small part of the tree.

Chapter 6

Conclusions

In this thesis, we used the Reinforcement Learning method to develop a computer program that can play Seega Game; which is one of the oldest board games. This game is similar to chess but with more complex rules. Seega has two stages; In the first stage, the players position their stones on the board in a strategic manner. In the In the second stage, the players move their stones vertically or horizontal to capture the opponent's stones.

We developed two computer agents that can play Seega game, the first agent can play stage-1 of the game namely *Positioning Agent*, the other agent can play the stage-2 of the game namely *Moving Agent*. We used reinforcement learning to train the two agents on self-play games without human data or guidance.

In this work, we proposed using the Monte Carlo tree search to play and simulate games, and deep convolution neural networks to train the agents. We have divided the training process into two steps; In the first step, we trained the *Moving Agent* using self-play games on randomly-generated boards; which trained the agent to select the best moves of the piece's to win the game. In the second step, we trained the *Positioning Agent* using self-play games starting from an empty board to select the best cell to place the piece's in the perspective of the current player. The trained *Moving Agent* is also be used during the training of *Positioning Agent* to predict the winner of the game when the players finish positioning all their pieces instead of playing the full game. During the training and after each iteration we evaluated the newly trained model and compared it with the best-selected model so far by playing many games against it, the newly trained model is accepted as the best-model if it wins 60% of the games, the Elo rating system is used to calculate the points for each model. This evaluation step improved the training process since the accepted model is used in the next iterations to generate better playing samples to train the neural network.

To evaluate our methodology, we developed another computer program that plays Seega game using the classical tree search algorithm mini-max. We used a board with the same arrangement of its stones to compare the playing skills of the trained model and the mini-max algorithm.

The experiment results show that the two agents learned the rules of the game, and their actions clearly show the knowledge they got during the training. This makes our methodology suitable to be used in real-world games in terms of accuracy of playing and the low computing power and time needed to use the trained models compared with the traditional mini-max algorithm.

This work is good evidence of using reinforcement learning to solve board games similar to Seega. especially those that lake the pre-collected training date data.

6.1 Future Work

Although the provided analysis and methodologies are quite good and constitute a powerful proof of the skills of the trained agents, there are some improvements that can still be made. Different experiments have been left for the future because of a lack of time.

We suggest the following future extensions to our work:

- We suggest running the same experiments on a server with more computer power and for more period of time. This could lead the agents to get more knowledge and improve their playing skills.
- Try another neural network architectures to train the two agents.
- Another work that can be done is to develop an actual game with a graphical user interface (UI) that can use the trained model to play against the actual people.

Appendix A

Result tables

iteration	samples	new-model wins	old-model wins	accepted?
1	2,400.00	35	35	NO
2	4,800.00	33	37	NO
3	7,200.00	36	34	NO
4	9,600.00	35	35	NO
5	14,400.00	38	32	NO
6	16,800.00	35	35	NO
7	16,800.00	45	25	YES
8	19,200.00	36	34	NO
9	21,600.00	35	35	NO
10	24,000.00	28	42	NO
11	26,400.00	40	30	YES
12	28,800.00	29	41	NO
13	31,200.00	32	38	NO
14	33,600.00	30	40	NO
15	36,000.00	29	41	NO
16	38,400.00	29	41	NO

17	40,800.00	31	39	NO
18	43,200.00	41	29	YES
19	45,600.00	31	39	NO
20	48,000.00	39	31	YES
21	50,400.00	32	38	NO
22	52,800.00	33	37	NO
23	55,200.00	36	34	NO
24	57,600.00	39	31	YES
25	60,000.00	38	32	NO
26	62,400.00	42	28	YES
27	64,800.00	39	31	YES
28	67,200.00	37	33	NO
29	69,600.00	46	24	YES
30	72,000.00	34	36	NO
31	74,400.00	32	38	NO
32	76,800.00	36	34	NO
33	79,200.00	31	39	NO
34	81,600.00	30	40	NO
35	84,000.00	23	47	NO
36	86,400.00	36	34	NO
37	88,800.00	32	38	NO
38	91,200.00	35	35	NO
39	93,600.00	37	33	NO
40	96,000.00	38	32	NO

41	98,400.00	28	42	NO
42	100,800.00	40	30	YES
43	103,200.00	35	35	NO
44	105,600.00	35	35	NO
45	108,000.00	36	34	NO
46	110,400.00	35	35	NO
47	112,800.00	31	39	NO
48	115,200.00	43	27	YES
49	117,600.00	36	34	NO
50	120,000.00	34	36	NO
51	122,400.00	40	30	YES
52	124,800.00	43	27	YES
53	127,200.00	38	32	NO
54	129,600.00	38	32	NO
55	132,000.00	39	31	YES
56	134,400.00	45	25	YES
57	136,800.00	40	30	YES
58	139,200.00	22	48	NO
59	141,600.00	37	33	NO
60	144,000.00	35	35	NO
61	146,400.00	40	30	YES
62	148,800.00	33	37	NO
63	151,200.00	37	33	NO
64	153,600.00	37	33	NO

65	156,000.00	27	43	NO
66	158,400.00	30	40	NO
67	160,800.00	37	33	NO
68	163,200.00	31	39	NO
69	165,600.00	42	28	YES
70	168,000.00	27	43	NO
71	170,400.00	32	38	NO
72	172,800.00	32	38	NO
73	175,200.00	37	33	NO
74	177,600.00	32	38	NO
75	180,000.00	35	35	NO
76	182,400.00	33	37	NO
77	184,800.00	30	40	NO
78	187,200.00	36	34	NO
79	189,600.00	32	38	NO
80	192,000.00	34	36	NO
81	194,400.00	32	38	NO
82	196,800.00	35	35	NO
83	199,200.00	36	34	NO
84	201,600.00	39	31	YES
85	204,000.00	49	21	YES
86	206,400.00	28	42	NO
87	208,800.00	31	39	NO
88	211,200.00	26	44	NO

89	213,600.00	453	-383	YES
90	216,000.00	24	46	NO
91	218,400.00	31	39	NO
92	220,800.00	30	40	NO
93	223,200.00	39	31	YES
94	225,600.00	32	38	NO
95	228,000.00	31	39	NO
96	230,400.00	29	41	NO
97	232,800.00	37	33	NO
98	235,200.00	30	40	NO
99	237,600.00	37	33	NO
100	240,000.00	32	38	NO
101	242,400.00	34	36	NO
102	244,800.00	35	35	NO
103	247,200.00	29	41	NO
104	249,600.00	34	36	NO
105	252,000.00	30	40	NO
106	254,400.00	27	43	NO
107	256,800.00	31	39	NO
108	259,200.00	32	38	NO
109	261,600.00	31	39	NO
110	264,000.00	36	34	NO
111	266,400.00	29	41	NO

References

[1] "Seega,(al-sija) - ancient games - playing the board games of the ancient world." https://www.ancientgames.org/seega/. (Accessed on 09/25/2020). x, 6

[2] "Reinforcement learning for control systems applications - matlab & simulink." https://www.mathworks.com/help/reinforcement-learning/ug/reinforcement-learning-for-control-systems-applications.html. (Accessed on 09/25/2020). x, 11

[3] H. Baier and P. D. Drake, "The power of forgetting: Improving the last-good-reply policy in monte carlo go," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 303–309, 2010. x, 15

[4] M. Koch, T. Duigou, and J.-L. Faulon, "Similarity-guided monte carlo tree search for bio-retrosynthesis," *bioRxiv*, p. 800474, 2019. x, 17

[5] "Convolutional neural network tutorial: From basic to advanced - missinglink.ai." https://missinglink.ai/guides/convolutional-neural-networks/

convolutional-neural-network-tutorial-basic-advanced/. (Accessed on 09/25/2020). x, 18, 19

[6] O. E. David, N. S. Netanyahu, and L. Wolf, “Deepchess: End-to-end deep neural network for automatic learning in chess,” in *International Conference on Artificial Neural Networks*, pp. 88–96, Springer, 2016. x, 20, 21

[7] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, “Mastering the game of go with deep neural networks and tree search,” *nature*, vol. 529, no. 7587, p. 484, 2016. x, 21, 22, 23

[8] A. M. Abdelbar, S. Ragab, and S. Mitri, “Co-evolutionary particle swarm optimization applied to the 7/spl times/7 seega game,” in *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No. 04CH37541)*, vol. 1, pp. 243–248, IEEE, 2004. xi, 26

[9] A. Aljaafreh and N. Al-Oudat, “Development of a computer player for seejeh (aka seega, siga, kharbga) board game with deep reinforcement learning,” *Procedia Computer Science*, vol. 160, pp. 241–247, 2019. xi, 26, 27

[10] S. Sharma, “Explained: Deep learning in tensorflow — chapter 1 - towards data science.” https://towardsdatascience.com/explained-deep-learning-in-tensorflow-chapter-1-9ab389fe90a1, 11 2019. (Accessed on 04/26/2020). xi, 29

[11] A. M. Abdelbar, S. Ragab, and S. Mitri, “Applying co-evolutionary particle swam optimization to the egyptian board game seega,” in *Proceedings of the*

first Asian-Pacific workshop on genetic programming, no. CONF, pp. 9–15, 2003. 6, 24, 26

[12] E. W. Lane, *An account of the manners and customs of the modern Egyptians*. Oxford University Press, 2012. 6

[13] E. Falkener, *Games Ancient and Oriental, and how to Play Them: Being the Games of the Greek, the Ludus Latrunculorum of the Romans and the Oriental Games of Chess, Draughts, Backgammon and Magic Squares*. Longmans, Green and Company, 1892. 6

[14] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," 2011. 11

[15] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992. 14

[16] R. S. Sutton, A. G. Barto, *et al.*, *Introduction to reinforcement learning*, vol. 135. MIT press Cambridge, 1998. 14

[17] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *ACM Sigart Bulletin*, vol. 2, no. 4, pp. 160–163, 1991. 15

[18] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012. 16

[19] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A

convolutional neural-network approach," *IEEE transactions on neural networks*, vol. 8, no. 1, pp. 98–113, 1997. 17

[20] A. Severyn and A. Moschitti, "Twitter sentiment analysis with deep convolutional neural networks," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 959–962, 2015. 17

[21] D. E. Knuth and R. W. Moore, "An analysis of alpha-beta pruning," *Artificial intelligence*, vol. 6, no. 4, pp. 293–326, 1975. 20

[22] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *International conference on computers and games*, pp. 72–83, Springer, 2006. 21

[23] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017. 22, 23, 26

[24] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018. 23, 32

[25] H. Iida, M. Sakuta, and J. Rollason, "Computer shogi," *Artificial Intelligence*, vol. 134, no. 1-2, pp. 121–144, 2002. 23

[26] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings*

of ICNN'95-International Conference on Neural Networks, vol. 4, pp. 1942–1948, IEEE, 1995. 24

[27] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016. 27

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. 27

[29] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014. 27

[30] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org. 28

[31] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and Ten-*

sorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media, 2019. 29

[32] M. E. Glickman and A. C. Jones, "Rating the chess rating system," *CHANCE-BERLIN THEN NEW YORK-*, vol. 12, pp. 21–28, 1999. 40

[33] L. M. Hvattum and H. Arntzen, "Using elo ratings for match result prediction in association football," *International Journal of forecasting*, vol. 26, no. 3, pp. 460–470, 2010. 40

[34] M. Myślak and D. Deja, "Developing game-structure sensitive matchmaking system for massive-multiplayer online games," in *International Conference on Social Informatics*, pp. 200–208, Springer, 2014. 40

[35] R. Coulom, "Computing "elo ratings" of move patterns in the game of go," *ICGA journal*, vol. 30, no. 4, pp. 198–208, 2007. 40

[36] K. Kourou, T. P. Exarchos, K. P. Exarchos, M. V. Karamouzis, and D. I. Fotiadis, "Machine learning applications in cancer prognosis and prediction," *Computational and structural biotechnology journal*, vol. 13, pp. 8–17, 2015.

[37] M. Ikonomakis, S. Kotsiantis, and V. Tampakas, "Text classification using machine learning techniques.," *WSEAS transactions on computers*, vol. 4, no. 8, pp. 966–974, 2005.

[38] P. Singhal and N. Raul, "Malware detection module using machine learning algorithms to assist in centralized security in enterprise networks," *arXiv preprint arXiv:1205.3062*, 2012.

[39] H. C. Bolton, “Seegà, an egyptian game,” *Journal of American Folklore*, pp. 132–134, 1890.

[40] L. Shao, J. Han, P. Kohli, and Z. Zhang, *Computer vision and machine learning with RGB-D sensors*, vol. 20. Springer, 2014.

[41] J. Heaton, N. G. Polson, and J. H. Witte, “Deep learning in finance,” *arXiv preprint arXiv:1602.06561*, 2016.

[42] C. Biancalana, F. Gasparetti, A. Micarelli, A. Miola, and G. Sansonetti, “Context-aware movie recommendation based on signal processing and machine learning,” in *Proceedings of the 2nd Challenge on Context-Aware Movie Recommendation*, pp. 5–10, 2011.

[43] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[44] I. Ghory, “Reinforcement learning in board games,” *Department of Computer Science, University of Bristol, Tech. Rep*, vol. 105, 2004.

[45] J. Kober, J. A. Bagnell, and J. Peters, “Reinforcement learning in robotics: A survey,” *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[46] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, *et al.*, “Model-based reinforcement learning for atari,” *arXiv preprint arXiv:1903.00374*, 2019.

[47] Z. Liu, M. Zhou, W. Cao, Q. Qu, H. W. F. Yeung, and V. Y. Y. Chung, "Towards understanding chinese checkers with heuristics, monte carlo tree search, and deep reinforcement learning," *arXiv preprint arXiv:1903.01747*, 2019.

[48] Sergio Guadarrama, Anoop Korattikara, Oscar Ramirez, Pablo Castro, Ethan Holly, Sam Fishman, Ke Wang, Ekaterina Gonina, Neal Wu, Efi Kokiopoulou, Luciano Sbaiz, Jamie Smith, Gábor Bartók, Jesse Berent, Chris Harris, Vincent Vanhoucke, Eugene Brevdo, "TF-Agents: A library for reinforcement learning in tensorflow." `https://github.com/tensorflow/agents`, 2018. [Online; accessed 25-June-2019].

[49] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[50] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[51] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[52] S. Fujimoto, H. Van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *arXiv preprint arXiv:1802.09477*, 2018.

[53] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[54] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.

[55] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” *arXiv preprint arXiv:1801.01290*, 2018.