Development of an Artificial Intelligent Software Agent using Artificial Intelligence and Machine Learning Techniques to play Backgammon Variants

Aristotelis-Antonios Kotsias, Georgios Tagkoulis
Abstract

Artificial Intelligence has seen enormous progress in many disciplines in the recent years. Particularly, digitalized versions of board games require artificial intelligence application due to their complex decision-making environment. Game developers aim to create board game software agents which are intelligent, adaptive and responsive. However, the process of designing and developing such a software agent is far from straightforward due the nature and diversity of each game. The thesis examines and presents a detailed procedure of constructing a software agent for backgammon variants, using temporal difference, artificial neural networks and backpropagation. Different artificial intelligence and machine learning algorithms used in board games, are overviewed and presented. Finally, the thesis describes the development and implementation of a software agent for the backgammon variant called Swedish Tables and evaluates its performance.

Keywords

Artificial Intelligence; Neural Network; Software Agents; Reinforcement Learning; Backgammon; Swedish Tables
Abstract


Nyckelord

Artificiell intelligens; Neuralt nätverk; Mjukvara agenter; Förstärkande lärande; Backgammon; Svenska tabeller
# Table of Contents

1 Introduction ................................................................................................................. 1
1.1 Background .............................................................................................................. 1
1.2 Problem .................................................................................................................. 2
1.3 Purpose ................................................................................................................... 3
1.4 Goal ........................................................................................................................ 3
  1.4.1 Benefits and Sustainability .............................................................................. 3
1.5 Methodology / Methods ......................................................................................... 3
1.6 Stakeholder .............................................................................................................. 5
1.7 Delimitations .......................................................................................................... 5
1.8 Outline (Disposition) ........................................................................................... 5

2 Analysis of Artificial Intelligence Methods used in Games .......................... 6
  2.1 AI Methods used in Games .................................................................................. 7
    2.1.1 Ad-Hoc Behaviour Authoring .................................................................. 7
    2.1.2 Tree search ................................................................................................. 7
    2.1.3 Evolutionary Computation ....................................................................... 9
    2.1.4 Supervised Learning ............................................................................... 9
    2.1.5 Reinforcement Learning .......................................................................... 10
    2.1.6 Unsupervised Learning ......................................................................... 11
  2.2 Software Agents ................................................................................................. 12
  2.3 How to construct a Software Agent to play Backgammon Variants .......... 13
  2.4 Related Work ...................................................................................................... 14

3 Methodologies and Methods .............................................................................. 15
  3.1 Data Collection Methods ................................................................................... 15
    3.1.1 Interviews ................................................................................................. 15
    3.1.2 Observations ............................................................................................. 16
    3.1.3 Documents Collection ........................................................................... 16
    3.1.4 Case Study ............................................................................................... 17
  3.2 Data Analysis ....................................................................................................... 17
    3.2.1 Coding ....................................................................................................... 18
    3.2.2 Grounded Theory ..................................................................................... 18
    3.2.3 Narrative analysis, Hermeneutic and Semiotic ....................................... 19
  3.3 Quality Assurance ............................................................................................... 19
  3.4 Software Development Methods ....................................................................... 19
    3.4.1 Scrum ........................................................................................................ 21

4 Software Agent Prototype - Implementation and Description .............. 23
  4.1 Project Requirements .......................................................................................... 23
  4.2 System Design ...................................................................................................... 23
    4.2.1 Multi-layer Perceptron Architecture as a function approximation and  
      Backpropagation algorithm for weight tuning .............................................. 24
    4.2.2 TD(λ), Learning Rate and Decay Parameters ........................................ 25
    4.2.3 Hybrid Algorithm used for Training the Software Agent Prototype ....... 26
  4.3 System Architecture ............................................................................................ 27
  4.4 Sprint Iterations throughout the Project ............................................................ 27
  4.5 The implemented Software Agent ..................................................................... 28

5 Training and Testing of the Software Agent ...................................................... 30
5.1 Learning Methodology ................................................................. 30
5.2 Training ..................................................................................... 31
5.3 Testing ....................................................................................... 32

6 Conclusions and Future Work ..................................................... 37
6.1 Discussion of the results ............................................................ 38
6.2 Areas of Improvement and Future Work .................................... 38

References ...................................................................................... 41
List of Figures

Figure 1: Waterfall Model................................................................. 21
Figure 2: Lifecycle of Sprint............................................................. 22
Figure 3: Neural Network................................................................. 25
Figure 4: Workflow of the System..................................................... 27
Figure 5: Chart combining the results of Table 2 and Table 3.............. 34
Figure 6: Chart illustrating the results of Table 4 ............................... 36
Figure 7: Chart combing the results of Table 5 and Table 6 ................. 36
List of Tables

Table 1: Description of sprints................................................................. 28
Table 2: Wins over the bot-player_3 when the training was focused on the "current" and on the "other" player............................................................ 33
Table 3: Wins (out of 1000 games) of the trained SA over the bot-player_3 when \( \lambda=0.7 \) and \( \alpha=1 \) ........................................................................ 33
Table 4: Wins (out of 1000 games) of the trained SA over the bot-player_3 when \( \lambda=0.7 \) and \( \alpha=0.7 \) ........................................................................ 34
Table 5: Wins (out of 1000 games) of the trained SA over the bot-player_3 when \( \lambda=0.7 \) and \( \alpha=0.3 \) ........................................................................ 35
Table 6: Wins (out of 1000 games) of the trained SA over the bot-player_3 when \( \lambda=0.7 \) and \( \alpha=0.3 \) ........................................................................ 35
Table 7: Wins (out of 1000 games) of the trained SA over the bot-player_3 when \( \lambda=0.7 \) and \( \alpha=0.3 \) ........................................................................ 35
## List of acronyms and abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>SA</td>
<td>Software Agent</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>FSM</td>
<td>Finite State Machine</td>
</tr>
<tr>
<td>BT</td>
<td>Behaviour Trees</td>
</tr>
<tr>
<td>DFS</td>
<td>Depth First Search</td>
</tr>
<tr>
<td>BFS</td>
<td>Breadth First Search</td>
</tr>
<tr>
<td>MCTS</td>
<td>Monte Carlo Tree Search</td>
</tr>
<tr>
<td>SL</td>
<td>Supervised Learning</td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial Neural Network</td>
</tr>
<tr>
<td>MLP</td>
<td>Multi-layer Perceptron</td>
</tr>
<tr>
<td>RL</td>
<td>Reinforcement Learning</td>
</tr>
<tr>
<td>TD</td>
<td>Temporal Difference</td>
</tr>
</tbody>
</table>
1 Introduction

Artificial Intelligence (AI) is a field which has seen great popularity and development in recent years. Day by day, more researchers are working on AI in some form while more non-researchers are interested in the field of AI. Innovative algorithms together with the increasing computational power due to hardware advancements have led to the rapid progress in AI [1]. Successful uses of AI are daily experienced in our lives through many applications such as pattern and speech recognition, self-driving cars and computer games.

Computer games -especially board games- are an interesting and popular field for AI application and research. For many years game developers have used AI to create interactive games which are intelligent, adaptive and responsive. For example, Backgammon is a game of particular interest as it is a good test of principles of AI [2]. Variants of Backgammon can be found in many regions around the world and although they rarely are as interesting as the standard version, they offer a pleasant break to the monotony of the game.

1.1 Background

Games are considered an ideal area for exploring and testing new ideas in AI and Machine Learning (ML). Shannon’s chess playing algorithm [3] and Samuel’s checker’s learning program [4] pioneered the field in the 40’s while the ground breaking moment was when Tesauro’s TD-Gammon [5], a computer Backgammon program, achieved to play in a level slightly below the top human players’ level of that time. More games like Go, Othello, Tic-Tac-Toe, Shogi and Backgammon variants have been studied extensively [6]–[8] because it is a common belief that advancements in the field can benefit other, more significant applications, in the future.

The reasons why games offer an ideal domain for the study of AI varies greatly. First, they are hard and interesting problems [1] due to their vast number of finite state spaces and their increased complexity as the domain of states rises. Furthermore, they provide a rich Human-Computer interaction [1] that is defined in terms of available options a player has and the ways that can interact with the medium. The popularity of the games is another reason too [1]. The more popular the games are, the more they grow in quantity and new AI techniques are required to meet the new challenges. At last, games challenge all AI areas resulting in advancements in many different areas, like Natural Processing Languages with Jeopardy! [9], Tree-Search algorithms with zero-sum games [10] and ML fields with Go [11].

Programming a board game can get different implementations. First, it can refer to providing a platform in which players can be connected and play
against each other in a digital board that resembles the actual one and the moves of the checkers comply with the rules of the game. On the other hand, it can also refer to providing a client program, in which a player can play against a Software Agent (SA); a program in the server that makes decisions to move the checkers with the ultimate target to win the game. Although there are programs that provide both functionalities the focus of this thesis is in the latter case.

Stochastic board games are particularly interesting for developing and applying AI and ML algorithms on SAs due to the large state space and the high branching factor resulting from the probabilistic dice rolls [5]. Backgammon is a two-player stochastic game that is played on a one-dimensional track where the players move their checkers (opposite or same direction) along the track based on the resulting numbers from the two dice roll. Many variants of the game exist [12] which differ in rules for moving the checkers, different checker direction and starting positions.

1.2 Problem
When programming a game, and particularly a board game, at any given state there is a set of legal moves that can be chosen from. Creating a SA that can compete in a professional level requires the selection of moves to follow a certain strategy instead of simply choosing randomly. AI methods satisfy exactly this need.

In stochastic games, especially in Backgammon, the task of programming a high-level SA is rather a difficult undertaking. While the usual practice of providing a look-up table that matches states with actions can be used in certain simplified endgame situations, this approach is not feasible for a full game because of the vast number of possible states ($10^{20}$ according to estimates [5]). Even traditional methodologies of deep searches are prohibited due to the high branching ratio that the probabilistic dice rolls introduce. At each ply (meaning moves ahead of both the SA and the opponent) there are twenty-one possible dice combinations. Together with an average twenty legal moves per dice combination the result is a branching ratio of several hundred per ply [5]. This branching ratio is, apparently, too large to reach significant depths even when the computations are performed in the fastest supercomputers.

Hence, the question that the thesis answers is:

“How to develop a SA using AI to play Backgammon variant games in a professional level?”
1.3 Purpose
The purpose of the thesis is to examine and describe the procedure for constructing a SA which can play Backgammon variants. An overview of AI, ML and Search Tree algorithms, which are implemented in board games, is presented. The thesis focuses on the combination of temporal difference algorithm, backpropagation algorithm and artificial neural networks for the design of a SA that plays the Backgammon variant Swedish Tables [13] in a professional level.

1.4 Goal
The aim of the thesis is the design and development of a SA for the Backgammon variant called Swedish Tables [13]. The SA is implemented with the use of artificial neural networks, backpropagation and temporal difference which train the SA through self-playing. Finally, the SA is embedded in the online game Sweboard [14].

1.4.1 Benefits and Sustainability
Sweboard’s online community of players is benefited from the thesis as the SA gives the illusion of intelligence to an appropriate level through its moves and therefore makes the game more challenging and more immersive to them. The three major pillars of sustainability are environment, economics and society. The degree project does not affect the two first categories as it does not have any direct or indirect connection with environment and economics. However, it affects the society in terms of brain training and improvement of thinking and strategy skills of the online players. More specific, playing against the intelligent SA makes the players to think of a strategy for making the correct decisions to move their checkers which will ensure success over time. This can be related to real world decision making and can have a positive effect on how people take decisions.

1.5 Methodology / Methods
Methodologies and methods are an essential part of any research process that guarantees proper, correct and well-founded results [15]. It is necessary to develop a strategy well ahead of the actual implementation, in order to steer the work and reach the desired goals and results.

The methodologies, as the processes followed during the research activity, are categorized to quantitative, qualitative and triangulation. Quantitative methodology refers to the research practice in which proving a phenomenon is heavily based on experiments or large data sets (that feed a
system) while the qualitative one refers to the study of a phenomenon, creation of a theory or coming up with an invention [15]. The thesis follows the Qualitative Research methodology as it implies an emphasis on the qualities of the process of how to design and develop a professional level SA for Backgammon variants. Moreover, not having a concrete hypothesis or statistically valid feedback, qualitative research is the most applicable to answer the open-ended question of the thesis.

In order to draw conclusions and establish what is true or not, research approaches, such as inductive, deductive and abductive, are used. In inductive approach, the research begins with a specific observation, continues with the detection of patterns and the formulation of a tentative hypothesis and ends up with the development of theories and propositions [15]. The deductive approach works from the more general to more specific. This means that the research begins with a theory about a topic of interest, narrows it down to a specific hypothesis and eventually verifies or falsifies the hypothesis [15]. Abductive approach uses both previous approaches to draw conclusions. A hypothesis is chosen and through an incomplete set of data/observations the simplest and best explanation is sought [15]. The research approach that is used in the thesis is inductive as the research is moving from specific observations to general propositions. More specific, data is collected and analyzed to gain an understanding of how AI has affected digitalized board games and establish different views of the AI algorithms that are used in the design of SAs. Then a tentative hypothesis about the development of a professional level SA is formulated. Finally, the thesis concludes with a proposition concerning a detailed and correct process of constructing a SA that plays Backgammon variants in professional level.

Software development methodology is an important framework that is used to plan, structure and control the process of designing and developing a system. Two popular software models are the waterfall model and the agile model. The first one, refers to the sequential process of software development while the second one is based on incremental and iterative process [16]. The thesis follows the agile method of Scrum. Taking into consideration that the team consists of two members, this makes the development process more flexible since the work can be done any time at any place.

The literature that has been studied for the needs of the thesis is separated into two sections:

- AI and Games Literature
- AI algorithms and Backgammon Literature

The purpose of this explicit discrimination relies on the fact that AI has a long-standing relation with games of different nature that is very relevant to the scope of this thesis. The algorithms that are used to solve different
problems faced in digitalized games, provide a valuable insight to the research of the thesis.

The second part of the review includes an in-depth overview of AI algorithms, with focus on Neural Networks, that are used specifically in the design of a SA able to play Backgammon variants.

1.6 Stakeholder
The thesis is carried out in collaboration with Jan Borgstedt, developer of the online game Sweboard. Jan provided an Application Programming Interface (API) of the online game with useful methods of the program that are exploited through the project work.

1.7 Delimitations
A SA that plays Swedish Tables is designed and developed during the thesis. Essential AI and ML algorithms are used to train the neural network. However, some technical details have been discarded.

The input of the neural network encodes only raw data of the board positions with no initial knowledge built of how to play good backgammon (knowledge free). The introduction of hand-crafted features relevant to good play would result in strong intermediate player.

The training of the neural network, due to computational costs, has been limited to thousands of games.

The neural network consists of one output.

1.8 Outline (Disposition)
This thesis is organized in the following way:
Chapter 2 presents a detailed analysis of AI methods and algorithms that are used in the design and development of computer games.
Chapter 3 presents the methods that are used for data collection and data analysis in qualitative research.
Chapter 4 consists of the steps that are followed during the thesis for the implementation of a SA.
Chapter 5 presents the results on how the implemented SA is performed against other opponents.
Chapter 6 presents the conclusions that are drown and describes future work.
2 Analysis of Artificial Intelligence Methods used in Games

AI and games have a long-standing and healthy relationship. AI algorithms have been improved or even invented through games while games have benefited largely through AI.

Games, from Game Theory, have either perfect or imperfect information and can be deterministic or non-deterministic [1]. In perfect information games each player is perfectly informed of all the events that have happened prior and up to that moment when makes any decision. Chess, Go and Tic-Tac-Toe are examples of games with perfect information [17]. On the contrary, imperfect information games, such as card games with hidden cards, are the games that the decisions made by each player are based on partly available information. In deterministic games, like chess, a player’s action leads to completely predictable outcomes, while in non-deterministic games the same input can result to completely different outcomes like when in Monopoly “draw a card” can lead to uncertain actions each time [18].

Backgammon, in particular, is considered a stochastic zero-sum game. Stochastic games refer to dynamic games with probabilistic transitions which can be played by one or more players [19]. On the other hand, zero sum games [10], a term found both in game theory and economics, refer to games that each player’s gain or loss is exactly balanced with gains or losses of the other participants of the game.

Two elements that are important for every AI method are representation and utility. Representation refers to the way “knowledge” is represented in AI systems and how it can be related to the way natural systems, like the brain, store and retrieve obtained knowledge. An AI system needs to use formats that a machine can understand and process to store knowledge. Usual representation types include graphs, trees and connectionism (neural networks). On the other hand, utility, a term that is found both in game theory and economics[1], is a measure of rational choice when playing a game. It is described as a function whose outcome is a decision regarding the optimal path to follow in an existing representation through search algorithms. Although utility can be found with different names in different AI methods (fitness in evolutionary computation or reward in reinforcement learning and Markov Decision Processes), the concept remains the same. [1]

In this chapter, initially some AI methods used in Games are presented. Then a general background on software agents is provided, continued by a detailed procedure on how to develop a SA that can play Backgammon variants. At last, some past work related to the subject of this thesis is described.
2.1 AI Methods used in Games

The AI methods that are commonly used in games can be divided in six categories. Ad-hoc authoring, tree search, evolutionary computation, supervised learning, reinforcement learning and unsupervised learning [1]. Each one of the categories contain algorithms that are used broadly with games to develop high-level SAs. The choice of an AI method and a particular algorithm is only depended on the nature and diversity of each game. Subsequently a description for each of the AI methods is provided.

2.1.1 Ad-Hoc Behaviour Authoring

Ad-Hoc Behaviour Authoring is the most popular method for game development and includes Finite State Machines (FSM), Behaviour Trees (BT) and utility-based AI [1].

FSM is the AI method that mostly preferred for the task of control and decision-making processes of Non-Player Characters in games. FSM has the form of a graph and provides an abstract representation of a system. More specifically, the graph contains nodes (states) that store information about a task and edges (transitions) that represent transitional relationships among the nodes. FSM is fully described by its set of states, a number of transitions (to other states) if the conditions are fulfilled and a set of actions that need to be followed in a state.

BT function in a similar way to FSM with the difference that they are composed from behaviors instead of states. BT has a tree structure with a root node and a number of parent and corresponding child nodes that represent behaviors. Active, success or failure are the values that a child can return to its parent. The node types of a BT structure are the sequence (when all child behaviors of a parent return success), selector (which is used to select one of the child behaviors) and decorator (which adds complexity to a child's behavior).

The utility-based AI approach was developed because FSM and BT cannot offer great behavioral modularity in a game. In utility-based approach, instances of the game are assigned with a function value that represents the importance of the particular instance. Instance can refer to selection of a weapon in a war-alike game or a move in a board game. At any moment, given the set of all the function values, the AI agent can choose the one that would lead to maximum gains in the future.

2.1.2 Tree search

A common belief among AI developers is that AI is mostly about search; searching the best possible (according to a specific measure) plan, move, path
and so on. Tree search algorithms accommodate exactly that need and they vary greatly depending on the order and the branches of the trees explored.

A first category of search algorithms consists of the uninformed search algorithms. These algorithms search in the state space with no further information regarding the goal and the dominant algorithms are Depth-first search (DFS) and Breadth-first search (BFS). DFS algorithm explores each branch of the tree as far as possible before backtracking and trying another branch. In game situations that means for each move to check the consequences until the game is won or lost before another branch, close to the end state, is examined. BFS, on the other hand, follows a different approach and for a particular node explores all the adjacent nodes before dive into a deeper level. [20]

Another category of search tree algorithm is when there is information regarding the goal state that guides the expansion of nodes. Best-First Search is the name of this category and the dominant algorithm is A* (A start). The A* algorithm holds a list of “open” nodes that have not been explored (but they are placed next to explored ones) and have information regarding their distance from the goal state. The selection of new nodes is made based on low cost basis, where the cost is the distance from the root node added to the estimated distance to the goal node.

While uninformed and informed tree search algorithms work very well for single-player games, for two-player games (opponent players), where the actions of each player depend greatly on the actions of the opponent, they fall short. For deterministic, perfect information, adversarial two-player games the basic algorithm is called Minimax. Minimax assumes that both players, called max and min, choose their actions optimally and its main loop alternates between them in a tree structure representation. For each state, all the possible moves from the opponent are explored until the game ends. Then a utility function assigns values to the end game positions (the final leaves) and by traversing the tree up to the root, minimax chooses what moves each of the player would choose in each state until it reaches the root node. [20]

Although minimax, and many of its variations like α-β pruning, create professional level SAs for games like Tic-Tac-Toe and Chess, where the branching ratio is fairly small, for complex perfect information, imperfect information and non-deterministic games it is the Monte Carlo Tree Search (MCTS) algorithm that increases the playing strength drastically. MCTS focus only in the branches of interest of the tree from a given state thus even with high branching ratio it manages to reach considerable depth during the search process. The biggest advantage of MCST over Minimax is that it only requires the rules of a game and the terminal state evaluation (that is win, double win, loss, draw and so on) to work without having to deal with any heuristic function [1].
2.1.3 Evolutionary Computation

Evolutionary computation refers to a broad family of optimization algorithms based on randomized variation of solutions [21]. An initial set of solutions are generated and iteratively updated by either changing or combining them together. Since this process resembles the natural evolution the term evolutionary computation is coined to describe it. In contrast to the search algorithms where from a root node representing an initial state a search tree is built depending on specific actions, evolutionary computation considers only complete solutions.

Evolutionary algorithms are a subset of evolutionary computation and they include all those techniques that implement mechanisms inspired by biological evolution such as reproduction, mutation, natural selection, recombination and survival of the fittest [21]. Candidate solutions correspond to the individuals of a population and the cost function determines the environment in which the solutions are implemented.

Optimization and evolutionary algorithms include a wide variety of methods and techniques and are usually used to create game content like different levels or to find player models.

2.1.4 Supervised Learning

Supervised Learning (SL) is an AI and ML method of approximating a function that maps an input to an output according to a pair of input-output training data set [22]. In SL, initially there is a data set of input-output pairs (labelled training examples) where input is usually in the form of a vector and produces a desired learning output (or supervisory signal). The data set is then split in training and testing data set with a ratio of seven over three (although the ratio can be different depending on the developer). The training set is used to train a system, or rather learn a function, that maps inputs to specific outputs while the testing data set is used to evaluate if the trained system is able to produce valid results. When the process of training and testing from the initial data set is over, optimally whenever a new unseen input is fed to the system, it will be able to provide correctly its output. SL is inspired from concept learning, a term found in human and animal psychology [1].

There are numerous SL algorithms and each one of them has a different way to find and represent the learning function. SL algorithms include Artificial Neural Networks (ANNs), Decision Tree methods, naïve Bayes classifier, k-nearest neighbours, case-based reasoning, Random Forests and Support Vector Machines [22]. Depending on the input data different SL approaches can be applied that include classification (that is used to predict categorical class labels), regression (that is used to estimate relationships
among variables) and preference learning (that is used to predict ranks and preferences). Due to the scope of the thesis, ANNs are subsequently presented in greater detail.

Decision tree methods involve segmenting or stratifying the predictor space into a number of simple regions, called edges and leaf, based on a condition. Random Forests is a decision tree method, that collects decision trees whose result is aggregated into one final result. The advantage of Random Forests over decision trees is that they have the ability to limit overfitting without substantially increasing error due to bias. The naïve Bayes’ Classifier is a probabilistic machine learning model that makes classification using the Maximum A Posteriori decision rule and is used for text classification and spam detection. K-nearest neighbours is a classification and regression algorithm in which the data points are separated into several classes to predict the classification of a new sample point. Case-based reasoning is the process of solving new problems based on the solution of similar past problems. Lastly, given a training set where elements are marked as belonging to one or other categories, a Support Vector Machine builds a model that assigns new elements to one category or the other. [22]

ANNs are inspired by the way neurons in the human brain process information as well as learn and perform various tasks. ANN are comprised by connected artificial neurons. An artificial neuron consists of a number of inputs X, each of which is associated with a weight W. The dot product between X and W is then added to a bias weight B, forming the sum XW+B, which is fed into an activation function to produce the output of the neuron. The selection of an appropriate activation function depends on the problem itself, meaning that linear and non-linear separable problems require different approach. The most common way to form an ANN is to structure neurons in layers. Multi-layer Perceptron (MLP) is the simplest form of this structure, where neurons in each layer are not connected to each other but only to all neurons in other layers. [23]

In order to train an ANN, it requires to have a training algorithm that adjusts the weights W and B. Such a training algorithm needs a cost function and a search strategy for all the possible solutions. The backpropagation algorithm, which stands for backward propagation of errors, is the most common algorithm for training ANNs. It updates the weights that minimize the error function by computing the gradient (partial derivative) of the function with respect to each weight and adjusting them according to the weight that minimizes the error function the most. [23]

2.1.5 Reinforcement Learning
Reinforcement Learning (RL) is an AI and ML method that mimics the way humans and animals learn to take decisions by receiving positive or negative rewards from the environment [24]. Inspired by behaviourist psychology, RL algorithms are trained by the feedback they get from the environment based on how the AI agent interacts with it. More specifically, when in a particular moment the agent is at a certain state \( s \) and decides to move to a state \( s' \) by choosing an action \( a \) (from a set of permissible actions), the environment respond with an immediate reward \( r \). The iterative process of interaction between an agent and its environment finally teaches the agent to choose actions that lead to maximum sum of rewards. Thus, the agent aims to discover a policy \( (\pi) \) for selecting actions that in the long run will maximize the accumulative reward [24].

The interactions between an agent and its environment occur in discrete time steps and they can be modeled as a Markov Decision Process (MDP). A MDP is defined as a tuple of 5 element \((S, A_s, P_a(s, s'), R_a(s, s'), \gamma)\) where

- \( S \): is the finite set of states
- \( A_s \): is the finite set of actions available from a state \( s \)
- \( P_a(s, s') \): is the probability that an action \( a \) in a state \( s \) will lead to the state \( s' \)
- \( R_a(s, s') \): is the reward received for the action \( a \) that resulted the agent to transit from state \( s \) to \( s' \)
- \( \lambda \in [0,1] \): symbolizes the difference in importance between future and current rewards.

Temporal Difference (TD) [25], in particular, is an incremental RL group of procedures that are specialized for prediction. TD has the ability incrementally to use past experience in an unknown system to predict its future behavior. TD methods are driven by the difference between temporally successive predictions instead of the error between the predicted and actual outcome that is dominated in a supervised learning [25].

### 2.1.6 Unsupervised Learning

Unsupervised Learning is a ML and AI method that attempts to discover patterns or associations among the attributes of the input data without having explicit access to an output. Thus, with unsupervised learning the focus is on understanding the relationships among variables or observations instead of trying to predict values. The most known unsupervised learning techniques [1] are clustering and frequent pattern mining. Clustering is the unsupervised task of grouping together objects that are similar (according to a specific measure) to each other and differ substantially with the other groups. On the other hand, frequent pattern mining refers to the task of deriving frequent patterns and structures from data.
Clustering can be further divided in k-means, mixture models and hierarchical clustering [26]. K-means clustering refers to the process of partitioning n observations into k clusters where each observation is put to the cluster with the nearest mean. Each cluster in k-means is defined by one point, called centroid and is the mean of the samples in the cluster. The method of k-means is widely used in signal processing and data mining [26]. Mixture model, a term coined from statistics, is a probabilistic model that identify if subpopulations emerge within an overall population, therefore identifying clusters within a population. At last, hierarchical clustering seeks to build a hierarchy of clusters. There are two main strategies available for hierarchical clustering, agglomerative (bottom-up approach) and divisive (top-down approach) [1].

The two most popular types of frequent pattern mining for games are: frequent itemset mining and frequent sequence mining [1]. The main difference between the aforementioned types is whether the attributes of the data follow any particular order or not. The former type find structures among data attributes of no particular order while the latter works only if the data attributes are sequential according to an inherent temporal order [1].

2.2 Software Agents

SAs are computer programs whose behavior is analogous to a human [27], meaning that are capable of carrying out a task that is either too tedious or extremely difficult to be undertaken by people. SAs are able to function independently without any external direct intervention and are always in control of their actions. As described by Schermer [28], when a software entity displays some of the following characteristics, “i) reactive, ii) pro-active and goal-oriented, iii) deliberative, iv) continual, v) adaptive, vi) communicative, and vii) mobile”, then it can be regarded as a SA. At last, SAs can be characterized either in terms of their intelligence or in terms of their purpose [29].

The three main categories of SAs, regarding their intelligence are: adaptation, reasoning and autonomy. Adaptation refers to SAs that are context-aware, and they are able to change the way they interact with users under different circumstances. Reasoning refers to the ability of SAs to reason. There are SA that follow only simple rules while others drive their behavior using advanced AI. Autonomy, on the other hand, concerns the degree of a SA’s reliability upon a human. Some SAs rely heavily on human input before functioning, while others are completely autonomous. [29]

Regarding their purpose, SAs are characterized as generalists, transactional, informational, productivity, and collaboration. Generalist SAs are of general purpose; supporting a range of tasks and direct the users to
external resources. Typical examples in this category are Siri, Alexa and Cortana from Apple, Amazon and Microsoft respectively. Transactional are SAs that perform an automatic transaction on behalf of a user. For instance, a user may have a SA to buy or sell shares when they reach a certain price. Informational are the SAs that provide information to the users, like weather forecast, updates in stock markets and so on. Productivity SAs are these software systems that help a user, or a team of users, to accomplish tedious or rote tasks while collaboration ones are those SAs that help people to collaborate and communicate with each other.

2.3 How to construct a Software Agent to play Backgammon

Variants

Although there are many backgammon variants around the world, and it is fairly easy for everyone to come up with a new variant, the focus in this thesis is with those backgammon variants that resemble the original game into being a stochastic, zero sum game. In that situation, the aim for a SA is to score game positions, meaning from each state of the game to move to the state that will provide it with the maximum probability eventually to win the game. The most efficient way for that to happen is through a hybrid algorithm using SL together with RL. From SL, ANN are used to score game positions. At each state of the game, the SA needs to decide, from a set of legal moves, the one that will lead to the win. The ANN then, scores all the legal positions and the SA chooses the one that scores the highest expected value. The form of the ANN is an MLP trained with the backpropagation algorithm.

The ANN should have three layers. The input consists of the representation of the checkers on the board, called the “raw” features, but it can also have some “expert features”, meaning some important concepts of the game. The number of the hidden layer can be chosen with experimental results, but it is important to mention that the number of neurons in the hidden layer affect the training process, since more neurons lead to more processing time [30]. At last, the output of the ANN consists of a neuron, whose output determine the probability of a given move to win the game.

While the training of an ANN requires a SL setting, in the case of backgammon and its variants that is impossible. Because of the interaction between two different players, one cannot know in advance how the course of the game will be, and thus only 2,3 or 4 ply strategies can be devised. As explained in 2.1.5 section, the “TD has the ability incrementally to use past experience in an unknown system to predict its future behavior”. That is the reason that TD of RL is used during the training to tune the weights of the ANN.
In the training phase, when the move that scored the highest expected value is selected, it is evaluated against the expected value of the ANN in the next state, which according to the TD is considered the “correct” one. Finally according to that evaluation, and using the backpropagation algorithm the weights are tuned accordingly [30].

For a detailed description on how this procedure is applied in practice, the reader is encouraged to read chapters 4 and 5, where a case study of constructing a SA to play the backgammon variant Swedish Tables is undertaken.

2.4 Related Work

Many real-world applications require a more sophisticated approach that combine the strengths of different AI methods and result to hybrid AI algorithms. TD-Gammon [5] and Palamedes [30] used the TD method of RL, Multilayer Neural Network function approximation and self-play to develop SAs that learned to play extremely well with little knowledge of the game. TD-Gammon, developed in 1992, was the first software in the history of backgammon that was able to play on a grandmaster-level while Palamedes, which includes a group of three backgammon variant games (Plakoto, Portes and Feyga) and is played mostly in Greece and neighboring countries, won the gold medal in Portes, in the Computer Olympiad in Tilburg, in November 2011 [30].

In [5], G. Tesauro proposed a game-learning program called TD-Gammon. More specific, “TD-Gammon is a neural network that trains itself to be an evaluation function for the game of backgammon by playing against itself and learning from the outcome” [5]. TD-Gammon uses MLP architecture which consists of input, hidden and output layers together with a formula for changing the weights in order to approximate the target function. The training procedure of the program consists of feeding the input layer with the positions of the board’s checkers and estimating outputs depending on the inputs. Moreover, during the training the program represents both players, scores every legal move in each round and chooses the one with the maximum expected output for the player making the move.

Similarly in [30], the program Palamedes followed the training procedure proposed by G.Tesauro in TD-Gammon. Based on that procedure, the backgammon variants of Portes, Plakoto and Feyga are implemented and hand-crafted features of each variant were introduced. The results showed that the programs in the Palamedes surpass any other similar program.
3 Methodologies and Methods

When conducting the thesis, the scientific research methods and methodologies are very important and essential for planning and steering the work to achieve correct, reliable and valid results.

3.1 Data Collection Methods

Data collection methods are used to collect the necessary data that is required for the research. The most common data collection methods used in qualitative research are interviews, observations, documents collection and case study [15] [31].

3.1.1 Interviews

This is the most common and widely used method of data collection in qualitative research. Through interviews, data concerning participants’ opinions and beliefs about situations and problems is obtained. The gathered data is then used for understanding and interpreting the experiences that the participants have. Moreover, interviews provide large volumes of in-depth data quickly. However, they can be biased due to the fact that interviewees may provide incomplete or even false information. Qualitative interviews based on their structure are categorized in structured, unstructured and semi-structured.

Structured interview is when all questions are prepared in advance and scheduled for the specific purpose of getting certain information from the participants. Same set of questions is presented and asked to each participant. Many interviews can take place in a short period of time due to the quick conduction of structured interview. Therefore, a large sample can be obtained which can result in representative findings. However, different participants cannot be questioned different questions as the interview schedule must be followed something which makes structured interview inflexible.

Unstructured interview is “a conversational type of interview in which the questions arise from the situation” [31]. In contrast to structured interview, the set of questions is not prepared in advance something which allows new ideas to be brought up during the interview according to the answers that the interviewee provides. Consequently, there is a natural flow during the interview where the researcher (interviewer) asks questions and uses interviewee’s responses to decide on the next question.

Semi-structured interview lies between structured and unstructured interview. This category of interview includes a set of open-ended questions which is developed prior of the interview and it can be modified during the
interview process. Open-ended questions are the questions that cannot be answered with yes or no or simple word responses.

### 3.1.2 Observations
Observation method is used to collect data about cultures, processes and people in qualitative research. The researcher is benefited from the observation method in a variety of ways [32]. First, through observation method the research can develop a holistic understanding of a phenomenon by observing behaviours, nonverbal expressions and interacts between the participants. Moreover, observation method allows the researcher to verify or falsify the answers that participants gave during interviews as participants may deliberately share incorrect information, thereby the validity of the research is strengthened. Interaction analysis is a specialized approach to observation and it is categorized in kinetics, which is the study of body movements, and proxemics which is the study of how people use space [31]. However, limitations are involved while using kinetics and proxemics due to use of space and cultural relations such as nonverbal behaviour and body movements which can differ across cultures.

Four stances toward observation have been identified depending on the degree to which the research involves in the participation. The five stances are; complete participant, participant as observer, observer as participant and complete observer [31], [32].

Complete participant is the researcher who participates in a group of people but hides his role in order to avoid disturbing normal activity and obtain observations based on natural behaviour of the other members. In the participant as observer stance the members of the group are aware of the role of the researcher who is actively participating as member of the group. In the observer as participant stance the researcher may interact with other members in some degree while his role is known to them. Finally, complete observer is the researcher who is hidden from the group while observing it.

### 3.1.3 Documents Collection
Documents collection is a systematic collection method that is used from researchers in order to gain understanding of the phenomenon by reviewing and evaluating documents. Documents refer to a wide range of visual, written and physical materials such as reports, files, books and videos. The analytic procedure contains searching, finding, selecting and synthesizing the collected data from the documents. Document collection, as a collection method, is particularly applicable to qualitative research due to the variety of documents which allow the researcher to reveal, overview, understand and gain insights relevant to the research problem.
Document collection has advantages and limitations [32]. More specific, the plethora of obtainable public documents and the cost-effectiveness of it, have made document collection an attractive option for obtaining empirical data. Moreover, documents provide a broad coverage in terms of time span and quantity. Therefore, documents can provide a good descriptive information and can be used as stable sources of data. However, limitations such as incomplete selection of documents based on the research agenda can be observed. Moreover, the researcher should consider the authenticity, completeness and original purpose of the documents that are used in research in order to examine the biases and unintentional or intentional distortion of them.

3.1.4 Case Study
Case study is a comprehensive and an in-depth description of a single case such as a group, an individual, a process or a program [30]. A case study should be considered when contextual conditions related to the studying phenomenon are covered and when “how” and “why” questions need to be answered. According to Yin [31], the types of case study are explanatory, descriptive and exploratory. Explanatory case study investigates the data in depth to explain the phenomena. Descriptive case study describes the phenomena that occur within the data. Lastly, exploratory case study explores any phenomena in the data.

Case study can provide rich detailed information of complex phenomena as it allows an in-depth exploration and use of variety data sources related to the phenomena. This ensures that the studied unit is explored through many perspectives and therefore multiple angles of it are revealed and understood. However, case study has weaknesses as the in-depth search of the unit of interest inevitably leads to lack of in-bredth search.

The case study data collection method is chosen to be applied in the thesis. More specific, the in-depth description of constructing a SA for the backgammon variant Swedish tables is overtaken while different backgammon variants, such as Fevga and Plaktoto, are only mentioned without any further research or description.

3.2 Data Analysis
Data analysis methods are used to analyze the collected data [15]. Data analysis is a systematic and time-consuming process as the researcher faces plethora of information from documents, interview transcripts and data collected during the research all of which must be interpreted and examined. The analysis involves searching and arranging of collected data, synthesizing
and collecting what is important for the research. In order to create explanations and draw conclusions, the researcher must overview and organize all the collected data. According to [31], data analysis in a process that often occurs simultaneously or concurrently with data collection through a recursive, iterative and dynamic process. Data analysis methods involve efforts to understand the phenomenon under research, explain relationships and synthesize information. Commonly used methods for qualitative research are Coding, Grounded Theory, Narrative, Hermeneutic and Semiotic [15].

3.2.1 Coding
Coding analysis includes the creation of labels and strategies that can be applied on the data in order to numerate and develop the data into categories that can be interpreted and analyzed. During coding, raw data is analyzed and concepts and categories are developed. Coding of the items leads to the recognition of differences or similarities in the raw data. A commonly used approach is the iterative reading of all the data and its sorting into understandable units of meaning [31]. At the beginning, the researcher can use as many codes as needed and develop tentative categories. However, the initial codes are likely to be modified or reduced later.

After the completion of coding the data, units with same codes are placed together. The merging of the codes is done in order to reduce the large number of individual codes into set of categories. Once this step is done, the researcher should review the set of items to ensure that they belong together. Once categories have been created, it is common that they still have relationships which lead to another merging and creation of themes. Themes are one level of abstraction above categories [31]. The process of coding, categorizing and development of themes is repeated for the whole data.

3.2.2 Grounded Theory
Grounded theory is an analysis method about conceptualizing the data “where theory emerges from the data through process of rigorous and structured analysis” [33]. Grounded theory emphasizes on theoretical development as final output of the research. It consists of detailed and structured processes for the development of theory from data. A broad research question is the start of the grounded theory, which points out the general area that need to be studied during the research.

Comparative method is used in order to compare the emerged concepts and categories derived from the stages of data analysis. The basis of the emerging theory is formed from the relationship between these categories and concepts. The comparison process is continued until no new significant
concepts or categories are emerging, also known as “theoretical saturation” [33]. Finally, grounded theory ends with a validated theory [15].

3.2.3 Narrative analysis, Hermeneutic and Semiotic
Narrative analysis, according to [15] concerns literary discussion and analysis and uses papers, journals, interviews and fields notes as units of analysis in order to research and comprehend the way people create meaning. Hermeneutic analysis (meaning of text) and semiotic analysis (meaning of symbols and signs) are used for interpretation. Hermeneutic analysis allows an in-depth understanding of meanings such as texts, cultures and arts. Understanding is achieved through systematic interpretation processes. Semiotic analysis aims to study, analyze and interpret signs.

3.3 Quality Assurance
Quality assurance concerns the verification and validation of the qualitative research and deem the rigor in the quality of the data. The qualitative research applies and discuss validity, transferability, dependability and confirmability [15].

Validity concerns the believability and accuracy of the research’s findings such as conclusions, interpretations and observations. The researcher has the obligation to represent the findings of the participants as credible and accurate as possible and must provide assurances that the obligation is met. The participants can validate and confirm if the findings are accurate and credible. Transferability is the degree to which the findings of the study can be transferred to other groups or other contexts [31]. The findings can be generalized or applied to other similar contexts, populations and situations. The researcher must provide detailed, complete and accurate descriptions of the research’s context so that other researchers can apply them appropriately and make necessary judgements. Dependability corresponds to reliability and refers to the correctness and replicability of the research’s findings. Lastly, confirmability deals with the objectivity and neutrality used in the procedures and the interpretation of the findings.

3.4 Software Development Methods
Nowadays many software development methods for software project exist in order to organize the work process inside the project, according to its priorities, and achieve the goals in time. Common used software development methods are waterfall and agile methods [16] such as Scrum [34].

Waterfall method is a sequential process of software development where the project progresses sequentially from one stage to another, Figure 1.
Firstly, there is requirement specification and then analysis, design, coding, testing, installation and finally maintenance. Only when the previous stage is fully completed then the project can continue to the next one. Therefore, a considerable amount of time is spent at every stage in order to meet all of its requirements as no changes occur later. More specific, the waterfall method consists of distinct phases which follow the sequential logic.

- **Requirements**: This is the initial phase which involves understanding of what needs to be designed and what its function is. The requirements from the stakeholder are collected and the specifications of the input and output of the product are examined.

- **Analysis**: At this phase, the data and system are analyzed in order to properly develop and generate models and business logic that will be used later on.

- **Design**: The requirements are studied, and the system design is prepared. The necessary hardware and software of the system are specified, and the overall architecture of the system is defined. Therefore, a design specification is created that outlines the technical specification of the system.

- **Coding**: At this phase the actual source code of the system is written.

- **Testing**: The source code is now systematically tested in order to discover and report issues that need to be resolved.

- **Installation and maintenance**: Finally, the system is deployed in the customer environment and subsequent maintenance and support is provided to the client.

Agile methods focus on adaptability and agility during the development of the project. More specific, agile methods involve multiple iterations during each stage of the project so as to improve the output with every iteration. At each iteration, all the steps of design, coding and testing are revised [16]. The team is self-organized with cross functional structure. The project can be presented and demonstrated to the customers at the end of each iteration and their feedback can determine the course of changes in the next iteration. Therefore, the design idea is agile as it can be evolved during each iteration. This process is repeated until the delivered product meets the requirements of the customer.
3.4.1 Scrum

Scrum is an agile software development approach that is suitable for projects that are constantly altering. It defines a holistic, flexible development process where the team works as a unit to achieve its goals [34]. The project is divided into short periods of time, called sprints, where actions are iteratively taken. Sprints are considered to be complete when time period expires. At the end of each sprint there is a meeting, called retrospective, where the team members talk about their work and project’s progress is tracked. The lifecycle of each sprint is described in Figure 2, where its action is taken iteratively from the beginning of the project until the release of the product/system. Furthermore, there are short daily meetings where each member reflects upon its work and what has and hasn’t been done since the previous daily meeting. In this way, all the members are updated of the project’s progress and are aware of team’s performance [34]. Moreover, during the meetings new requirements may introduced or previous might change something.
which makes Scrum an agile methodology. Scrum consists of the following 5 phases \[35\].

- **Initiate**: This is the initial phase of the Scrum where the scrum master and stakeholders are identified, team is formed, epics are developed, prioritized backlog is created, and release planning is identified.
- **Plan and Estimate**: This phase consists of processes related to planning and estimating tasks.
- **Implement**: At this phase, the tasks and activities are executed in order to create the project’s product/system.
- **Review and Retrospect**: The deliverables are reviewed and ways to improve the project work are determined.
- **Release**: The product is delivered to the customer.

![Figure 2: Lifecycle of Sprint](image)

Throughout the entire project, the agile methodology of Scrum is followed in order to manage the project work and successfully implement all of its requirements. Specifically, Scrum optimizes the team’s efficiency because each member of the team has to answer three simple questions each day and work accordingly to the answers of the questions.

*What have I done yesterday? What am I going to do today? Which problems do I have?*

In this way, each member every day knows what has to be done and what his tasks are. Moreover, with Scrum the team is able to deliver the product faster and receive feedback at the end of each sprint something which is not the case with Waterfall where no value is given to the product until the very end. Furthermore, two persons are in charge of the project something which makes the whole development process more flexible, since the work can be done at any place any time. Lastly, Scrum is easy to use since both team members are familiar with this method as they have worked with it in the past. Therefore, for all the above reasons, Scrum is chosen as the Software Development Method followed in this project.
4 Software Agent Prototype - Implementation and Description

As a case study of this thesis, the implementation of a SA for the backgammon variant Swedish Tables is performed. In this chapter the requirements of that project, the system design and the system architecture are outlined. Specifically, the various components of the system are analyzed, and their connection is illustrated. Moreover, the work during each sprint is presented and finally the SA prototype is presented.

4.1 Project Requirements

Sweboard is an online version of the Backgammon game Swedish Tables where the users can connect and play. When a player is registered (for the first time) and logged in (every other time), he is prompted to choose an opponent from a list of available players. This list consists of other players, that are currently connected to the Sweboard and the SA called BrädeAI. Therefore, a player can choose to either play with another online player or with the BrädeAI. Only one player, at any given time, can play with the SA, so if someone already plays all the other players need to wait to finish and exit the game before it becomes available again. BrädeAI’s logic is programmed according to the game knowledge and personal experience of the game (Swedish Tables) of Jan Borgstedt. However, Jan is interested in developing a second SA for the game where artificial intelligence and neural network are used for its training.

For the identification of the system’s requirements, Skype meetings were arranged with Jan where he expressed his requirements. More specific, the first requirement is that the SA is developed by using AI algorithms and ANNs. Another requirement is that the developed SA is competed and evaluated against two random-picking move SAs and BrädeAI. Lastly, Sweboard’s repository cannot be shared with third-party individuals and for that reason only the project’s repository is available.

At last, during the research and evaluation of the system’s characteristics, a specified number of training iterations and neural network structure, due to hardware limitations, have been selected.

4.2 System Design

Due to delimitations and requirements the hybrid algorithm combining ANN of SV and TD of RL has been chosen for the development and training of the SA that plays the backgammon variant, Swedish Tables. More specifically, from the SL method, the ANN is chosen for the representation of the different
states of the backgammon variant Swedish Tables [13], together with the backpropagation training algorithm for weight tuning. But since there is no training set, the “correct answers” for the training are collected using RL, meaning that the next state, after a move initiated by the ANN, is considered the “correct” and by that state the training of the ANN is performed.

4.2.1 Multi-layer Perceptron Architecture as a function approximation and Backpropagation algorithm for weight tuning

At the heart of the system is a neural network that is organized in a MLP. Specifically, the neurons are organized into three categories which are: the input layer, the hidden layer and the output layer. The input layer consists of 192 neurons, the hidden layer consists of 100 neurons and the output layer consists of 1 neuron, as shown in Figure 3. The input layer consists of neurons which take inputs in the following way. For each position on the backgammon board, the number of checkers of one color on that position is represented by four neurons. If there are 0-3 checkers on one position, then 0-3 neurons are used while the fourth is set to 0. For instance, if there is no checker on the position then the neurons take the value 0000 and the values 1000 or 1100 or 1110 if there are one or two or three checkers respectively. If there are more than 3 checkers (e.g. n checkers) on the position, then the first three neurons are set to 1 and the fourth is set to (n-3)/2. Moreover, two additional neurons take the value n/2 (n are the number of checkers) which encodes the number of black and white checkers on the backgammon bar. The last two neurons take the value n/15 which encode the number of black and white checkers which have been removed from the backgammon board. The hidden layer consists of 100 neurons while the output layer consists of 1 neuron which outputs the probability of a given player to win.

MLP architecture with feed-forward neural network is used because it has extremely robust function approximation capabilities. In fact, given sufficient hidden units, the MLP architecture is capable of approximating any nonlinear function to arbitrary accuracy [36]. Moreover, the input follows TD-Gammon’s pattern because each position is represented in a straightforward way, making little attempt to minimize the number of units.

As it is mentioned in section 2.1.4, there are many Supervised Algorithms that are used for training an ANN. For the purposes of this thesis, the backpropagation is chosen as the suitable algorithm for training the SA prototype as it can efficiently calculate the error between the actual output of the neural network (provided by the TD method of RL) and the expected output, given by a specific input and then backpropagate and update/tune the weights of the neural network.
4.2.2 TD(λ), Learning Rate and Decay Parameters

The learning rate parameter alpha (α) and the decay parameter lambda (λ), both from the RL method, are used in the training of the neural network. More specifically, network’s weights are changed at each time step by applying the Temporal Difference Lambda (TD(λ)) algorithm. The weights change according to the (cost or error function) formula [5] which can be found in Equation 1.

Equation 1: The Cost or Error function formula

\[ w_{t+1} - w_t = \alpha(Y_{t+1} - Y_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w Y_k \]

Different values of the learning parameter alpha either converge or diverge the training of the neural network. When the α is low the training is more reliable, but more time is needed for the optimization. On the other hand, when α is high the training may not converge or even diverge. In the beginning, the training starts from a relatively high α because random weights are far from optimal while it can decrease during the training to allow more fine weights tuning. An approach followed in the thesis, is to try different values of α (e.g. 0.1, 0.01, 0.001) and pick the one that gives best loss.
without sacrificing speed of training. Moreover, another method followed in the thesis, is the splitting of the $\alpha$ parameter into two other learning parameters alpha and beta. The learning parameter alpha is applied on the weights from input layer to hidden layer while beta is applied on the weights from hidden layer to output layer.

At every time step, there is a heuristic error based on the difference between two successive predictions. “The temporal credit assignment of how an error detected at a given time step feeds back to correct previous estimates is controlled by the $\lambda$ parameter. When $\lambda=1$ or $\lambda=0$ the error feeds back without decay far in time or no feedback occurs beyond the current time step respectively. Intermediate values of $\lambda$ give a smoother correction of previous estimations” [5].

Temporal difference method is used to train the neural network. TD($\lambda$) method is used to identify and reinforce the blame or credit with respect to the outcome. More specifically, at the beginning the weights of the neural network are initialized with the zero value. At each move, the neural network evaluates all the available moves for the current playing side and picks the one with the best outcome. Since the weights are zero, the moves at the beginning are random. However, through TD($\lambda$) method the information about these trainings is maintained over time and eventually decays something which affects future training. At the end of each game, when one player wins or losses, the information is concrete, and a more correct error is computed and applied to future training with respect to older trainings.

4.2.3 Hybrid Algorithm used for Training the Software Agent Prototype

As it was mentioned earlier, this thesis implements a combination of Backpropagation and Temporal Difference for the training of a SA through self-play. This is a suitable hybrid algorithm due to the fact that Backgammon is a stochastic game. Specifically, backpropagation can calculate the error of neural network’s output and then update the weights of the neural network. However, since Backgammon is a stochastic game, the expected output of each move is not predefined or known from before and cannot be calculated with Backpropagation. For that reason, TD($\lambda$) is used to calculate the difference between the expected output and current output. Finally, the SA instead of trying to imitate humans, develops its own sense of positional judgement through self-play. The advantage of this is that the SA is not affected by human biases that could be unreliable or erroneous. Moreover, self-play training takes less time than training against a human player. Therefore, self-play is more efficient and reduces the cost of training significantly.
4.3 System Architecture

The system consists of several interconnected parts, illustrated in Figure 4. First, the necessary functions are exported using Sweboard’s API. Through the functions, the SA (is presented as “Bot” and has exactly the same meaning) has access to important information such as the rules and the stage of the game. The system uses this information to first train the SA and then test it against its opponents. More specific, when there is no trained ANN to be loaded, the system enters the training phase where it trains the SA through self-play.

![Figure 4: Workflow of the System](image)

After a pre-defined number of games, a trained ANN is generated. Subsequently, the generated ANN is loaded to the system and the SA competes against one of the opponents mentioned in the requirements. Reliability in the trained ANN is achieved through the large number of self-playing training and the continuous change of the learning rate and decay parameters which are explained in detail in following sections.

4.4 Sprint Iterations throughout the Project

The project work is divided into sprints and their tasks are presented in this section. Specifically, two sprints concern the literature study together with understanding the artificial intelligence and neural networks. One sprint involved the familiarization with the backgammon program provided by Jan Borgstedt, four sprints in the development of the SA and its integration with the main backgammon program. Skype meetings with Jan took place who helped us to understand how core functions of the program work and guide us through his program. Furthermore, one sprint involved training and testing the program with different learning parameters and gathering data. Lastly, two sprints were spent in writing and polishing the report with the guidance of our examiner. All the above are described in Table 1. At the end
of each sprint, its results are evaluated, and unfinished tasks are forwarded to the next sprint.

Table 1: Description of sprints

<table>
<thead>
<tr>
<th>Sprint(s)</th>
<th>Task(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-2</td>
<td>Literature study concerning AI and ANN</td>
</tr>
<tr>
<td>3</td>
<td>Skype meetings with Jan for the clarification of his program and his requirements</td>
</tr>
<tr>
<td>4-7</td>
<td>Development of the program &amp; integration</td>
</tr>
<tr>
<td>8</td>
<td>Testing and evaluation of the results</td>
</tr>
<tr>
<td>9-10</td>
<td>Writing and correcting the final report of the thesis</td>
</tr>
</tbody>
</table>

4.5 The implemented Software Agent

The AI method that this project degree followed, as has been mentioned before, is the TD method of RL, together with Multilayer Neural Network function approximation and self-play. Therefore, a hybrid algorithm has been selected, that combines the strengths of supervised and reinforcement learning. In the following webpage the SA that has been developed, for the purposes of this degree project, can be found: https://github.com/GiorgosTagkoulis/Thesis

The SA itself is developed in the Java programming language [37] in order to comply with the game (developed by Jan) itself [14]. Java provides an Object-Oriented way of programming [38]. The SA consists of a Neural Network package and two Java classes, named Player and Utility. The Neural Network package consists of four Java classes. Each unit of the neural network is an object of type unit, or to be more precise either InputUnit or HiddenUnit since these two classes extend the Unit interface. Having initialized the class NeuralNetwork with 192 InputUnits and 100 HiddenUnits, it is possible to take the neural network presented in Figure 3 and use it for the training of the SA (while the Neural Network is part of the SA, since the selection of the moves and the training is sole responsibility of the Player class, sometimes it is referred as a helpful package to the SA).

The AI agent playing the game of Sweboard is the Player class. It consists of the methods that during the training will train the ANN while during a game will choose the best, according to its training, moves. Utility class is used only to manipulate the board positions, in order to fit as an input to the ANN, and to return the value of the ANN. Therefore, the most critical part of the SA is the Player class. When the program starts to run, if there will not be a file called “SavedNN” in the same directory, then Player class (which means the SA) will get into training mode. That means that it will start playing with itself and after the training completion a new file named “SavedNN” will be written.
In a normal supervised learning, when the ANN estimates a value upon which a move will be selected, it would be compared with the “correct” value and the weights would be adjusted accordingly, using the backpropagation method described in chapter two. But since in Sweboard, and in Backgammon in general, there is no correct move because of the element of luck introduced by the dice, the reinforcement learning fills the gap of choosing the “correct” move. Thus, as a “correct” value of the ANN is considered the value as estimated after a player has chosen a move. And subsequently the training using the backpropagation method will be performed using the values of the input of the ANN (before the selected move), the value of the output of the ANN (after the move) and the value of the ANN for the player that played but using the board that resulted after the move (it is called expected). The backpropagation method then adjust the weights according to the Equation 1. Only in the case that a move will lead to winning or losing the game there is an actual correct value, and then the training is performed using the value 1, for a win, or 0 for lose for the expected value.
5 Training and Testing of the Software Agent

The training and testing of the ANN, and essentially the SA, consist of several steps. First, a suitable learning methodology is picked and applied for the training. Then, during the training the learning rate and decay parameters are modified repeatedly in order to achieve fine weight tuning. Lastly, for every pair of learning rate and decay parameter, a trained ANN is exported and tested for several games against the opponents, as posed by the requirements. These steps are described in detail in the following sections, together with the numerical results of the trained SA against the opponents.

5.1 Learning Methodology

As it is explained in chapter 4, the neural network consists of neurons that are connected with synapses (weights) that hold a value. The ANN’s core function is to score game positions. At any given time, the program needs to choose a move, from a set of available ones, that brings the highest probability, for the current player, to win the game. For that reason, as it was mentioned earlier, MLP are trained using the backpropagation algorithm.

The learning procedure consists of tuning the weights so that the function implemented by the ANN approximates, as close as possible, the desired target function. The updates of the weights are applied during a self-play game. At each time step, meaning before a move, the set of all available moves are collected and a calculation of the target of each board update is estimated. The move that is finally chosen, is the one that brings the highest probability for the current player to win. The strategy of selecting each player’s next move affects the quality of the trained neural network. Specifically, this can be achieved either by selecting a move that minimizes the probability of the other player winning the game or by selecting the move that maximizes the probability of the current player to win the game. After testing both of the strategies, the former one was selected as the main strategy to be followed for the training of the ANN as it produced better results (explained in more detail in section 5.3).

Therefore, for each available move for the current player, a new board is generated, and the ANN is fed with the board positions $x_1, x_2, ..., x_n$ as inputs (of the other player) which are encoded using the pattern that is explained in section 4.2.1. For each input combination there is an output $z$ (or probability $P_z$) which indicates the ANN’s expected outcome estimation for the specified input combination. Since the best move for the current player is wanted, the move that minimizes the probability (that the ANN generates) for the other player is chosen.
At last, it is worth mentioning that including handcrafting features of the game such as prioritizing hitting opponent’s checker instead of moving checkers to empty positions, can increase the overall performance of the learning system. However, only “raw” board representation as input pattern of the neural network has been followed in the thesis, without including any additional pre-computed features. For that reason, this is a knowledge-free implementation.

5.2 Training

Training an ANN requires training examples in a supervised learning setting. However, in a backgammon game play there is not exactly correct or incorrect move because there is the element of luck introduced by the dice. To overcome that difficulty, the method introduced by Tesauro [5], and explained in detail in chapter 4, is used (the TD(\(\lambda\)) algorithm and the ANN’s backpropagation algorithm to update the TD error).

As it has been mentioned in chapter four, during the training phase there is a number of different factors that play a crucial role in the final result and essentially the performance of the SA. They consist of:

- Selection of player, current or other, whose probability will be maximized or minimized respectively in the pursuit of the best move.
- Number of neurons in the hidden layers.
- Learning rate \(\alpha\).
- Decay parameter \(\lambda\).
- Wipe out or not the eligibility traces after the completion of each game or not.

Because there are a lot of “variables” to be tested, it becomes apparent that the number of different training settings is in the order of magnitude of one hundred. Since the processor power of the home laptops that have been used to train the ANN are not sufficient for that extensive testing (it is estimated that it would need four/five months to test all the alternatives), some important decisions were made at the initial stages of the training. While details about the testing of each training setting is given and explained in the next subsection, two of the bullet points are crucial to be presented here, because after the initial selection no further change had been made.

First, regarding the number of neurons in the hidden layers, they were carefully chosen because although they do not interact directly with the external environment, they influence the final outcome tremendously. Few hidden neurons could lead to underfitting, the situation when there are too few neurons in the hidden layers to adequately detect the signals in a
complicated data set [39]. On the other hand, too many neurons in the hidden layer could cause either overfitting or increase the training time of the ANN. Overfitting occurs when the ANN has a lot of information capacity compared to the training set and thus is not enough to train all of the neurons in the hidden layers. Many neurons in the hidden layer could increase the training time of an ANN at levels impossible to adequate train it. Therefore, for the case study of this thesis, the following rule was followed: “The number of hidden neurons should be between the size of the input layer and the size of the output layer.” [39].

Regarding the eligibility traces, its main role is to carry information from one training to another. While during a self-play game, it is mandatory to carry information to next moves, there are two alternatives when a game ends and the next one begins. One can wipe out the eligibility trace matrices and start fresh in the next game or keep the matrices and move the information to the next game. In preliminary tests, no major difference was spotted between the two alternatives and thus it was decided to keep the eligibility trace matrices unchanged between different games.

5.3 Testing

The SA that has been developed as part of the case study of this thesis has been tested against three different SAs or bot-players (as explained earlier SA and bot refer to the same entity), bot-player_1, bot-player_2 and bot-player_3, all provided by Jan Borgstedt. The first two bot-players are so-called naïve players while the third one is a sophisticated bot-player that resembles the BrädeAI of the Sweboard.

Regarding the naïve players, the selection of the moves at any given time step is performed with an element of luck. From the available moves at any given time step, bot-player_1 chooses the first one in the list while bot-player_2 chooses its preferred move randomly. The SA developed by this thesis, managed to outperform (100% win in 1000 games) both of these naïve players with minimum training of the ANN (trained for 10k self-play games). Therefore, all of the subsequent testing has been performed with the sophisticated bot-player_3.

Initially, the testing was focused on which player (current or other) should subsequent tests be focused on. The tests that were carried out and presented in Table 2, showed explicitly that when the chosen move minimizes the probability for the opponent to win, better overall results obtained. Lambda and alpha values were kept constant, so they do not interfere or affect the result in any case. The values of alpha and lambda were chosen after research in similar projects, as presented in 2.2.
So far, it has been presented how the number of neurons in the hidden layer was chosen, why the eligibility traces were kept as they were at the end of each self-play game and why training upon the “minimization” of the probability of win for the “other” player was preferred. Therefore, what is left is to choose values for lambda and alpha that will train a bot which can win the bot-player_3.

The best strategy, as presented in [5], is to begin the training with a large value of lambda and alpha and then decrease them while the training progresses. As it was mentioned in 4.2.2 alpha represents the learning parameter while lambda the decay parameter. At the beginning, it is desirable to keep the alpha value high, because the weights are tuned quicker (large modifications in the weights of the ANN). Similarly, regarding the decay parameter, lambda, it was decided to begin with large values (which mean the result of each time step would have impact to previous iterations as well) and decrease it when the performance in the results start to flatten. Table 3 and Table 4 illustrate the results from the neural networks, obtained during the training.

**Table 3:** Wins (out of 1000 games) of the trained SA over the bot-player_3 when λ=0.7 and α=1.

<table>
<thead>
<tr>
<th>Thousands of self-play games</th>
<th>Wins for λ=0.7 &amp; α=1 in 1000 games against bot-player_3</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>48 36 41 35</td>
<td>40.75</td>
</tr>
<tr>
<td>20</td>
<td>54 44 51 54</td>
<td>50.75</td>
</tr>
<tr>
<td>30</td>
<td>74 75 81 88</td>
<td>79.5</td>
</tr>
<tr>
<td>40</td>
<td>119 119 125 132</td>
<td>123.75</td>
</tr>
<tr>
<td>50</td>
<td>127 121 124 122</td>
<td>123.02</td>
</tr>
<tr>
<td>55</td>
<td>119 114 119 123</td>
<td>118.75</td>
</tr>
<tr>
<td>60</td>
<td>129 119 116 106</td>
<td>117.5</td>
</tr>
<tr>
<td>75</td>
<td>101 106 109 126</td>
<td>110.5</td>
</tr>
<tr>
<td>85</td>
<td>107 99 101 104</td>
<td>102.75</td>
</tr>
<tr>
<td>95</td>
<td>95 104 100 106</td>
<td>101.25</td>
</tr>
<tr>
<td>105</td>
<td>90 95 99 110</td>
<td>98.5</td>
</tr>
<tr>
<td>110</td>
<td>92 90 94 95</td>
<td>92.75</td>
</tr>
</tbody>
</table>
Table 4: Wins (out of 1000 games) of the trained SA over the bot-player_3 when $\lambda=0.7$ and $\alpha=0.7$.

<table>
<thead>
<tr>
<th>Thousands of self-play games</th>
<th>Wins for $\lambda=0.7$ &amp; $\alpha=0.7$ in 1000 games against bot-player_3</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>52 52 40 47</td>
<td>47.75</td>
</tr>
<tr>
<td>20</td>
<td>43 45 46 45</td>
<td>44.75</td>
</tr>
<tr>
<td>30</td>
<td>69 59 46 55</td>
<td>57.25</td>
</tr>
<tr>
<td>40</td>
<td>88 112 86 91</td>
<td>94.25</td>
</tr>
<tr>
<td>50</td>
<td>82 77 100 88</td>
<td>86.75</td>
</tr>
<tr>
<td>55</td>
<td>100 86 107 95</td>
<td>97</td>
</tr>
<tr>
<td>65</td>
<td>137 125 140 132</td>
<td>133.5</td>
</tr>
<tr>
<td>75</td>
<td>121 110 115 117</td>
<td>115.75</td>
</tr>
<tr>
<td>85</td>
<td>123 127 148 130</td>
<td>132</td>
</tr>
<tr>
<td>95</td>
<td>129 121 109 115</td>
<td>118.5</td>
</tr>
<tr>
<td>105</td>
<td>122 127 140 132</td>
<td>130.25</td>
</tr>
<tr>
<td>110</td>
<td>125 123 137 129</td>
<td>128.5</td>
</tr>
</tbody>
</table>

The tests of the neural network files, obtained during the training, were run four times against the bot-player_3, and its average was taken as the final result in that particular training (in thousands of times). With the average values the chart in Figure 5 was created to illustrate better the trend in the acquired results.

![Figure 5: Chart combining the results of Table 2 and Table 3](chart.png)
As it is observed in Figure 5, when lambda is 0.7 and alpha is 1, the performance of the neural network is increased until **40 thousand self-games** (when it won on average 123.75 games out of 1000), while afterwards it begins to decrease. Similarly, for \( \lambda = 0.7 \) and \( \alpha = 0.7 \), the performance of the neural network is increased until **65 thousand games** (when it won on average 133.5 games out of 1000) and afterwards it begins to decline.

Having these results, the question that arose was with which values would be better to continue with. In order to continue the training of the ANN, it was decided to get the two neural network files with the best performance so far against the bot-player_3 and continue the training with smaller values for lambda and alpha. Table 5, 6 and 7 present the obtained results.

**Table 5:** Wins (out of 1000 games) of the trained SA over the bot-player_3 when \( \lambda=0.7 \) and \( \alpha=0.3 \)
(From 40k neural network where \( \lambda=0.7, \alpha=1 \))

<table>
<thead>
<tr>
<th>Thousands of self-play games</th>
<th>Wins for ( \lambda=0.7 &amp; \alpha=0.3 ) in 1000 games against bot-player_3</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>145 127 135 132</td>
<td>134.75</td>
</tr>
<tr>
<td>150</td>
<td>139 128 142 137</td>
<td>136.5</td>
</tr>
<tr>
<td>200</td>
<td>127 119 133 124</td>
<td>125.75</td>
</tr>
<tr>
<td>300</td>
<td>116 124 98 109</td>
<td>111.75</td>
</tr>
</tbody>
</table>

**Table 6:** Wins (out of 1000 games) of the trained SA over the bot-player_3 when \( \lambda=0.7 \) and \( \alpha=0.5 \)
(From 65k neural network where \( \lambda=0.7, \alpha=0.7 \))

<table>
<thead>
<tr>
<th>Thousands of self-play games</th>
<th>Wins for ( \lambda=0.7 &amp; \alpha=0.5 ) in 1000 games against bot-player_3</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>153 142 147 139</td>
<td>145.25</td>
</tr>
<tr>
<td>150</td>
<td>135 129 123 130</td>
<td>129.25</td>
</tr>
<tr>
<td>200</td>
<td>115 129 129 132</td>
<td>124.25</td>
</tr>
<tr>
<td>300</td>
<td>107 118 122 111</td>
<td>114.5</td>
</tr>
</tbody>
</table>

**Table 7:** Wins (out of 1000 games) of the trained SA over the bot-player_3 when \( \lambda=0.7 \) and \( \alpha=0.3 \)
(From 65k neural network where \( \lambda=0.7, \alpha=0.7 \))

<table>
<thead>
<tr>
<th>Thousands of self-play games</th>
<th>Wins for ( \lambda=0.7 &amp; \alpha=0.3 ) in 1000 games against bot-player_3</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>138 149 145 133</td>
<td>141.25</td>
</tr>
<tr>
<td>150</td>
<td>143 137 133 128</td>
<td>135.25</td>
</tr>
<tr>
<td>200</td>
<td>130 123 134 115</td>
<td>125.5</td>
</tr>
<tr>
<td>300</td>
<td>87 101 92 97</td>
<td>94.25</td>
</tr>
</tbody>
</table>
From the 40k neural network file ($\lambda=0.7$ and $\alpha=1$), another 300 thousand self-playing games were played, but as the Table 5 shows, and better illustrate Figure 6, there was no major improvement. Similarly, from the 65k neural network file ($\lambda=0.7$ and $\alpha=0.7$), another 300 thousand self-playing games were played, and as Table 6 and 7 shows and Figure 7 illustrates, no major improvement was obtained.

Figure 6: Chart illustrating the results of Table 4

Figure 7: Chart combing the results of Table 5 and Table 6
The goal and the purposes of the thesis have been met. In chapter 2, the 2.3 section describes, despite all the difficulties because of the high branching ratio of the decision tree, a procedure of how to construct a SA that can play Backgammon and its variants that are stochastic, zero-sum games. Furthermore, in the section 2.1 a detailed overview of all AI and ML algorithms used in board games are presented together with information regarding which algorithms are used to specific domains. At last, chapter 4 describes in detail how using ANNs together with the TD method of RL and the backpropagation algorithm, it is possible to develop a SA that can play Swedish Tables and compete with other SAs.

More specifically, it is presented a detailed analysis for the development and training of a SA that plays Backgammon variants using as a case study the Backgammon variant Swedish Tables [13]. This is done in collaboration with Jan Borgstedt, developer of the online game Sweboard. Jan stated a series of requirements for the project and provided assistance and guidance throughout the entire project until its completion.

The goal of the thesis, providing a SA that has been trained through backpropagation and temporal difference to play Swedish Backgammon, has been achieved. The problem statement which questions how to develop a SA using artificial intelligence to play backgammon variant games in a professional level, is answered both in a general scope (in section 2.3) and using the case study of Swedish Tables (in chapter 4). Through the literature study, knowledge about fundamental artificial concepts and algorithms has been acquired. Moreover, the related work has high impact in the project description and the SA development. To become more concrete, temporal difference learning and TD-Gammon by G. Tesauro and Palamedes program by N. Papahristou and I. Refanidis provide a solid background in the development of the system’s architecture design and help to understand artificial neural networks and temporal difference and backpropagation in depth.

Using multilayer neural network, temporal difference, backpropagation and self-play the SA is trained to play Backgammon variants even with little knowledge of the game and its rules. Self-play and temporal difference are used to train the multilayer neural network as evaluation function of the Backgammon variants. Among many popular Backgammon variants, Swedish tables is chosen as a case study of the thesis and the SA is trained based on this variant. As it is shown from the results, in section 5.3, the SA after the training can always win versus naïve bots while it can partially win versus the sophisticated bot with capabilities similar to BrädeAI. The performance of the SA is affected by the number of hidden and output nodes,
number of plies, handcrafted features, number of training and the learning methodology. These factors are explained in the next section.

6.1 Discussion of the results

From an overall perspective, the chosen methods and methodologies have been suitable for the thesis research and system development. Through the thesis work and experiment, a sufficient dataset of neural networks has been collected and the requirements of validity, transferability, dependability and confirmability have been met. The neural networks files that have been collected during the training process with whom the tests in chapter 5 were ran can be found in: https://github.com/GiorgosTagkoulis/Thesis/tree/master/NN_files

A successful development and training of a SA has been achieved and conclusions have been drawn upon the results that are collected from a variety of tests scenarios. More specific, for the initial tests the (high) values of $\lambda = 0.7$, $\alpha = 1$ and $\lambda = 0.7$, $\alpha = 0.7$ were chosen. The training for these values lasted for 110 thousand self-play games. Approximately every 10 thousand games, a neural network file was collected and tested. From these tests, it was obvious that no more optimization was possible for the ANN with those values. Therefore, after collecting the best neural network file from each set of values, the training was continued with decreased values for lambda and alpha.

The new trainings lasted for 300 thousand games, in order to get a better perspective of how the ANN develop with more training. However, as Table 4, 5 and 6 illustrate, the developed neural network files did not succeed to improve. Of course, various reasons, that are discussed in the next section, can be responsible and future work is needed to obtain a professional-level SA.

6.2 Areas of Improvement and Future Work

Nevertheless, there are opportunities for further development and improvement of the program. Namely, the areas that can be improved in the future are:

- Greater depth of search
- Increase the number of neurons at hidden and output layer
- Increase the number of training
- Introduce handcrafted features (expert features)
- Different learning methodology
- Test the SA versus another opponent
More specific, greater performance can be achieved by increasing the search depth. Heuristic 1-ply search is executed by the current version of the program. This means that for each of the n legal moves of player_1, the program goes through all 21 possible dice rolls for the player_2 (opponent). One can think of it as “what is the best move of player_1 if also every possible dice rolls of player_2 are considered”. Adding more plies and therefore deeper lookahead can result in stronger players but increase the analysis and playing speed of the program. For that reason, more computational power is needed than the conventional laptops that are used to run the training and testing of the thesis.

Moreover, scaling of the system can be achieved by adding extra hidden and output neurons and increasing the amount of trainings. The current system consists of 100 hidden neurons and 1 output neuron. However, more input neurons can make the system to adapt to more complex patterns while with more output neurons the system is able to calculate more outcomes of the game such as draw, handsome game, win by Jan or forced Jan. Moreover, a technique that is followed in the thesis regarding the training iterations is that the α and λ parameters initially take their highest values while they decrease as more training iterations are conducted. However, the more the computational power the more training iterations with same α and λ values can be conducted which results in smoother alteration of α and λ values of the system. Consequently, these changes can lead to further improvement in playing ability of the system.

The input of the ANN can be divided to “raw” and “expert” features. In the system that is presented in the thesis, “raw” features such as positions of checkers on the board, bear off and hit checkers are fed as inputs to the ANN. However, implementing “expert” features can result to more sophisticated game concepts and selection of moves by the SA, something which is difficult to be achieved only from encoding “raw” features. For example, through the “expert” features the SA can decide to move a checker in a strategic spot on the board to win with handsome game instead of just bearing it off. Furthermore, different learning methodology is a field that involves further investigation. One suggestion is that the SA can be trained by playing against human experts. A potential drawback of this methodology is that playing against human experts is much slower than self-playing and a limited number of games can be played due to physical fatigue of the human opponent.
Acknowledgments

The authors thank Anne Håkansson, Johan Montelius and Jan Borgstedt for their guidance and their helpful comments on every stage of the development of the thesis.
References


