T.C.

MARMARA UNIVERSITY

FACULTY of ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

CSE4197 – Project Specification Document

Title of the Project

DeepCheckers (Deep Learning Based Checkers Board Game)

Group Members

150117828 – Cem Güleç

150116027 – Büşra Gökmen

150115069 – Muhammet Şeramet

Supervised by

Dr. Mehmet Kadir Baran

30 November 2020

Table of Contents

1. Problem Statement

Over the last years, deep learning become one of the pioneer methods to be used in many fields. It brought a new aspect to those fields where researchers slightly gave up, having a thought that there will be no hope of having a satisfactory improvement. On the other hand, an ancient game called "Checkers", which over the thousands of years, had different names, different versions, is one of the milestone games to be considered in Turkish culture.

This game is one of those subjects, that has been given up. Even though, having a good amount of improvement in the areas of AI and Machine Learning, due to the game's popularity in worldwide it is unfortunately not able to draw the researchers focus back. Main aim of this project is to solve game of Turkish Checkers by combining those fields in a sophisticated and modern way.

2. Problem Description and Motivation

Thanks to the perfect information, all relevant information is known to all players, board games throughout history served as a test environment for Artificial Intelligence (AI). As board games provide bounded environmental complexity and a clear goal often give a good indication of how well a particular agent is performing and acts as a measure of success. Furthermore, the artificial nature of most games has the benefit of its ease measurement of performance, ease to simulate, and removing the need for hardware agents to interact within the real world. Historically most research within the field of game-related AI has been devoted towards solving classical 2-player board games, such as Chess, Checkers, Go and Backgammon.

A recent milestone for AI in such domains is AlphaZero, which achieves superhuman level strength at Go, Chess, and Shogi by only playing against itself. The predecessor AlphaGo was produced and optimized for the game Go and famously managed to defeat one of the world's best Go players, Lee Sedol. Although AlphaGo, unlike AlphaZero, bootstrapped from human knowledge, AlphaZero decisively outperformed all versions of AlphaGo. Maybe even more impressive one is that AlphaZero also managed to defeat the previously strongest chess AI, Stockfish, after only 4 hours of self-play. Where Stockfish has been developed and hand-tuned by AI researchers and chess grandmasters for several years. [1]

The basic architecture of AlphaZero is as follows. AlphaZero learns by reinforcement learning from simulated games against itself. It uses a deep convolutional neural network that learns to estimate the values of positions and the probabilities of playing possible moves in a position. To select a move to play in the current board position, AlphaZero performs Monte Carlo Tree Search (MCTS). This search consists of simulating random games from the current positions, in which the probabilities of random moves increase with the move probabilities returned by the neural network and decrease with the moves' visit counts. [2]

One of the motives in our project is that the playing time of the Turkish Checkers is decreasing over time. The reason why the numbers decreased over time is that people didn't had a good competitor that will give a chance to improve their playing style, to change the way they look the game, and to get challenged. We hope to meet their expectations by new era technologies.

The other one is to inspire new state of surveys on Turkish Checkers with AlphaZero. To our knowledge there are not any studies that tries to master the game of Turkish Checkers in the similar way. We believe that in case of successful completion of our project, by using the requirements of AlphaZero, the number of literature surveys that focus on Turkish Checkers will accelerate. Based on this we believe this study is important and valuable.

3. Aims of the Project

Aims of the project are listed below:

To create artificial intelligence models (agents) that can learn and play Turkish checkers, which is a strategy game, with deep learning-based techniques.

Preventing Turkish Checkers from losing its value by creating an advanced competitor.

- It can be seen on the researches that has been made, the numbers of people interested to play Turkish Checkers is decreasing by time. The reason is that people did not have a good competitor on the game. One of the minor aims in our project is to increase these numbers.

To create a new survey area that focuses on Turkish Checkers.

- When we were researching related works on Turkish checkers, the numbers of surveys were almost zero. In order to increase the numbers, and find a new way to improve the researches about the game, we expect that our research will help to increase the numbers.

Model (Game Agent) Strategic Learning

- Models (agents) learn board coordinates for Turkish Checkers game and gain the ability to move on the board.
- Models (agents) to learn the strategies and best possible moves of the Turkish Checkers game.

Model Self Game Learning and Optimization

- The models (agents) learn the game by playing with the best model in the previous episode without manually playing with the human. With this approach, it is aimed to learn the game by playing by himself.

Model Predictive Ability

- Models (agents) learn more moves depending on the number of games and decide which of the next moves will be best. In this way, it is aimed to give the models the ability to predict.

Easy to Use by the User

- To make the game and the models (agents) ready for a real user (human) to play the game after the modeling phase is finished and the models are tested.
- To design an interface for the ease of use of the game.

4. Related Work

Under this section, several works described depending on their familiarity with the concepts we are planning to use. Besides studying the game Checkers, we also benefitted from different games such as chess, go, shogi, othello these concepts applied to.

In the work of Murray Campbell, A. Joseph Hoane Jr, Feng-hsiung Hsu [3], authors work on creating a chess playing system with solely purpose to defeat human world champion in regulation match. Deep Blue project is the culmination of about 15-years effort. System was consisting of a single-chip chess search engine, a massively parallel system with multiple levels of parallelism, a strong emphasis on search extensions, a complex evaluation function and usage of a game database that consists of Grandmaster gameplays information.
First version of Deep Blue fails to accomplish this purpose losing against former world champion Garry Kasparov in 1996. After that, with received improvement parameters to be handled they enhance the first version to form Deep Blue II. Followings are the major enhancements done:

* Firstly, a new significantly enhanced, chess chip was designed. The new chess chip had a completely redesigned evaluation function, going from around 6400 features to over 8000. Also, per chip search speed increased to 2-2.5 million positions per second with this system.

* Secondly, more than doubling the number of chess chips in the system, and use the newer generation of SP computer to support the higher processing demands.

* Thirdly, development of a set of software tools to aid in debugging and match preparation for evaluation tuning and visualization tools.

After enhancements done and optimized the system requirements, Deep Blue II wins against world champion Garry Kasparov in a six-game match in 1997.

This is written into history as first computer to win both a chess game and a chess match against a reigning world champion under regular time controls.

In the work of David Silver, Thomas Hubert [4], authors work on general game-playing program that replaces traditional game-playing programs with a deep neural network, reinforcement learning and a game tree search. In the neural network part, they use convolutional neural networks that takes game state and outputs a vector of move probabilities for each action and a value which estimates outcome of the game from that state. Neural networks are trained by reinforcement learning from self-play games. In game search tree, they use a general-purpose Monte Carlo tree search (MCTS) algorithm. As a result, they compared the model with best-known models in Chess, Shogi and Go respectively. In chess, AlphaZero first outperformed Stockfish after just 4 hours (300,000 steps); in shogi, AlphaZero first outperformed Elmo after 2 hours (110,000 steps); and in Go, AlphaZero first outperformed AlphaGo Lee (9) after 30 hours (74,000 steps).

In the work of Jonathan Schaeffer, Neil Burch [5], they describe the background behind the effort to solve checkers. The proof procedure has three algorithm/data components:

1) Endgame Databases (Backward Search). Computations from the end of the game back towards to starting position have resulted in a database where all positions with < 10 pieces on the board.

2) Proof-tree Manager (Forward Search). The manager maintains the master copy of the proof and identifies a prioritized list of positions that need to be examined using the Proof Number search algorithm.

3) Proof Solver (Forward Search). The solvers get a position to evaluate from the manager. The result of a position evaluation can be proven (win, loss, draw), partially proven (at least a draw, at most a draw), or heuristic (an estimate of how good or bad a position is).

The early research was devoted to developing CHINOOK and demonstrating super-human play in checkers, a milestone that predated the DEEP BLUE success. The project has been a marriage of research in artificial intelligence and parallel computing – with contributions made in each of these areas.

In the work of Kumar Chellapilla and David B. [6], authors work on where neural networks compete for survival in an evolving population based on their ability to play checkers. In the neural network was structured as a fully connected feedforward neural network with an input layer, two hidden layers, and an output node. The neural network outputs a scalar value that was interpreted as the worth of that board from the position of the player whose pieces were denoted by positive values. Each game was played using minimax as a search of the associated game tree for each board position. As an outcome, they showed us how the neural network evolved where it had no knowledge about nature of the game; simply a vector rather than board. Yet, even with this handicap, neural network was able to learn how to play competent checkers based essentially from the information contained in just "win, lose, or draw".

In the work of I. C. L. Dubel, Ing. J. Brandsema, L. Lefakis, S. Szóstkiewicz [7], authors work on several methods such as Reinforcement Learning, Monte Carlo method, TD-leaf, and neural networks to make the model learn how to success at game checkers. They apply several policies such as greedy, Ɛ-greedy and softmax action selection policies to study the effect of different ratios of exploration vs. exploitation. After experimenting those different situations with different policies, experiments conducted using reinforcement learning concerning of methods Temporal Difference Learning, Monte-Carlo and TD-leaf.
In order to gather all the information and test their model they use different situations:

* Simple intelligent agent vs. Random player: 25000 episodes used for the learning phase; results are the average over 10 experiments.

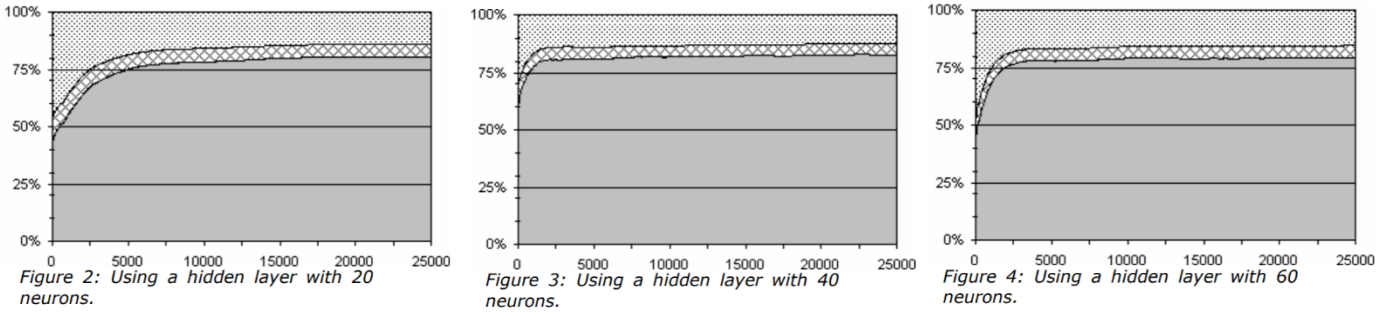

Figure 2: Using a hidden layer with 20 neurons.
Figure 3: Using a hidden layer with 40 neurons.
Figure 4: Using a hidden layer with 60 neurons.

Figure 1: Experiments resulted with different number of hidden layers in the first case.

* Agent using look-ahead vs. Random player: Here, the agent looks ahead to a certain depth in the game tree and chooses an action based on the min-max criterion. By allowing the agent to look-ahead to a depth of three, it is obtained results close to 90% winning accuracy.

* Agent using TD-leaf vs. Random Player: Here, the agent uses look-ahead while learning. At each state presented to the agent, the agent looks ahead to a depth of three and after finding the best possible leaf-state based on the minmax criterion, updates the root-state's value. Outcome of this method is 99% winning accuracy.

Our plan especially includes observing methods done in projects mentioned under this section and other related open-source projects to optimize our project with supporting it with the methods and algorithms we will come up with.

5. Scope of the Project

5.1. Inner Scope

Our primary purpose in this project to propose an intellectual method for using neural networks to extract information in a perfect information environment about how to play expert level checkers without having any dataset to learn about game in terms of features that would be believed to be important. To achieve this purpose, Deep Learning Techniques and MCTS methods will be used. There can be found various literatures providing new solutions on the same problem using Artificial Neural Networks [8] and Feed Forward Neural Networks [9], but the solutions they are offering is outdated comparing to current technologies. That is why we decided to focus on using Convolutional Neural Networks in our project.

5.2. Constraints/Limits

- AlphaZero achieves great results and decisively outperforms compared to previous approaches, but a vast number of computational resources are needed to benefit from this algorithm.
- Training time depends on computer's processing power.
- System should provide an output within a short period of time.
- The output of the system should be reasonable.

5.3. Claims/Assumptions

- It is assumed that the final system will be at least national-master level which will have 2200 rankings at least.
- It is assumed that the end model will be accurate. In a given game state, model will play the most promising move.
- It is assumed that the system will be more efficient compared to given works in Section 4.

6. Success Factors and Benefits

Measurability/Measuring Success:

- The success rate of this project will be measured by being top-rated checkers program according to the results of the games played by the end model. Ranking will be done by standard system.
- Model should achieve a reasonable level of plays without any expert knowledge.
- Model should provide an output within a limited of time.

Benefits/Implications:

This project will present a new approach to virtual games. With these approaches, the games can be automated, and the players can be offered different and many combinations. It can add different playing styles to the game rather than the methods currently available. This kind of project could also be an incentive for the need for research and development on strategy games. For example, when it was seen that the AlphaGo was successfully beating grandmasters, people both became respected and interested. This enabled people to keep an eye out for article production. Apart from these, it can show the applicability of deep learning approaches in studies requiring strategy.

7. Methodology and Technical Approach

Checkers Game Rules:

ITEMS NEEDED:

- Two Players
- One checker 8x8 board
- 16 pieces for each player

On an 8×8 board, 16 *men* are lined up on each side, in two rows. The back rows are vacant. A traditional Turkish draughts *gameboard* is mono-colored. White moves first.

Men move orthogonally forwards or sideways one square, capturing by means of a jump; they cannot move or capture backwards or diagonally. When a man reaches the back row, it promotes to a *king*. Kings can move any number of empty squares orthogonally forwards, backwards, or sideways. A king captures by jumping over a single piece any number of empty squares away, landing on any open square beyond the captured piece along a straight line.

If a jump is available, it must be taken. If there is more than one way to jump, the one capturing the greatest number of pieces must be taken. If there is more than one way to capture the maximum number of pieces, the player may choose. Pieces are removed from the board immediately after being jumped. There is no distinction between king and man during captures; each one counts as a piece.

A player wins if the opponent has no legal move, either because all his pieces are captured, or he is completely blocked. A king versus single man also wins the game. [10]

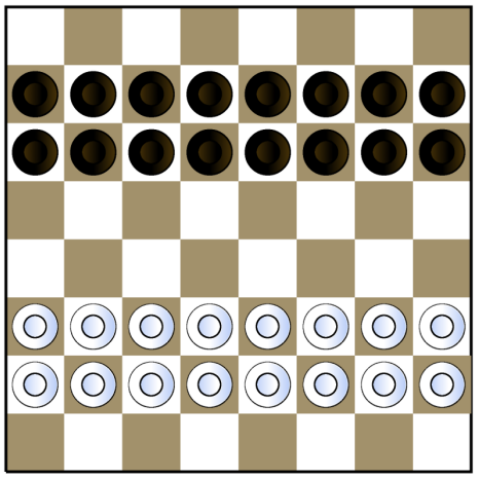

Figure 2: Starting positions of a checkers board.

7.1. Neural Network Approach and CNN

We examine strategic AI-based checkers game implementations through general algorithms for getting good at something quickly without prior knowledge of human expert strategy.

Once it is built up enough game positions to fill its memory the neural network will begin training. Through additional self-play and training, it will gradually get better at predicting the game value and next moves from any position, resulting in better decision making and smarter overall play.

We use code that starts the learning process. It loads the game rules and then iterates through the main loop of the algorithm, which consist of three stages:

1. Self-Play
2. Retraining Neural Network (CNN)
3. Evaluating Neural Network (CNN)

There are two agents involved in this loop, the **best_player** and the **current_player**. [11]

The best_player contains the best performing neural network and is used to generate the self-play memories. The current_player then retrains its neural network on these memories and is then pitched against the best_player. If it wins, the neural network inside the best_player is switched for the neural network inside the current_player, and the loop starts again. Each player is initialized with its own neural network and Monte Carlo Search Tree. The neural network is getting better at predicting the value of each game state and the likely next moves. Models have been learnt by the neural network, without any human input.

CNN Building: A Convolutional neural network (CNN) is a neural network that has one or more convolutional layers and are used mainly for image processing, classification, segmentation and also for other auto correlated data.

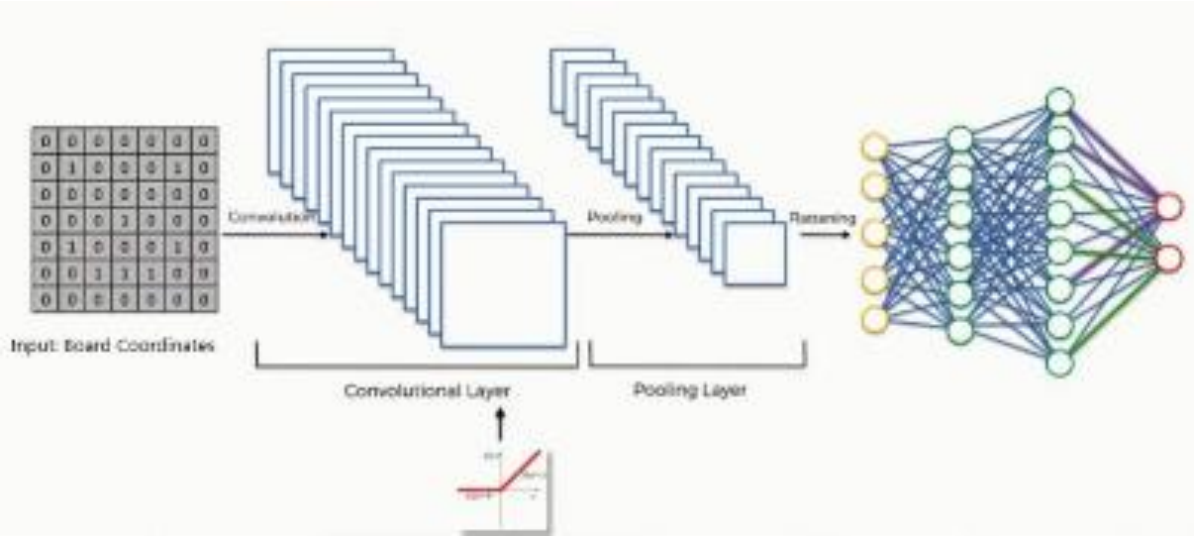

Figure 3: An example usage of CNN in our implementation

- We export the input data in the form of two-dimensional matrixes to the neural network. The symmetry of the filter to be applied to the two-dimensional information (input data) is taken according to the x and y axis. All values are multiplied element by element in the matrix and the sum of all values is recorded as the corresponding element of the output matrix. This is also called a cross correlation relationship.

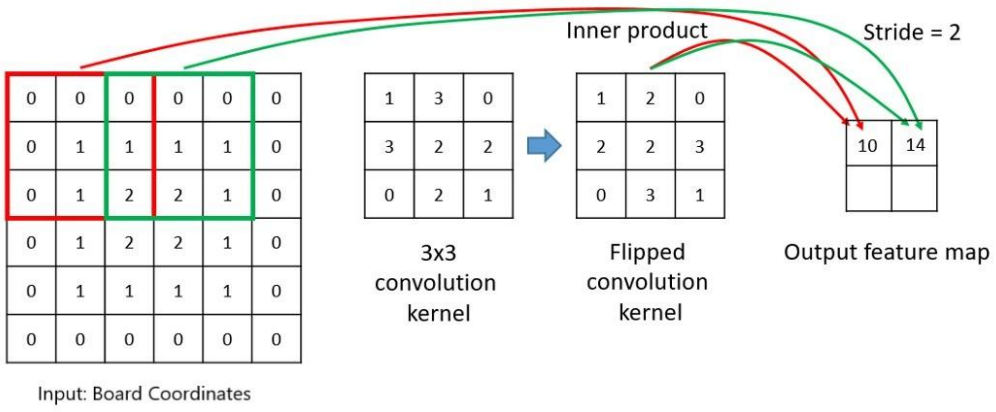

Figure 4: Convolution process with the specified filter. [12]

- Let us imagine this computation process as a layer in a neural network. The input data and filter are also actually the matrix of weights updated by continuous backpropagation. The filter, which is a weight matrix for convolution, is shifted over the image in steps of one pixel or larger steps. A scalar b (bias) value is added last to the output matrix applying the activation function.
- In the pooling layer, the maximum average method is generally used. There are no parameters learned at this layer of the network. It reduces the height and width information by keeping the number of channels of the input matrix constant. It is a step used to reduce computational complexity.
- And then, input goes to fully-connected layer. Fully Connected Layer is simply, *feed forward neural networks*. Fully Connected Layers form the last few layers in the network.
- The **input** to the fully connected layer is the output from the *final* Pooling or Convolutional Layer, which is ***flattened*** and then fed into the fully connected layer. *The output from the final (and any) Pooling and Convolutional Layer is a 3-dimensional matrix, to flatten that is to unroll all its values into a vector.*
- This Flattened vector is then connected to a few fully connected layers which are same as Artificial Neural Networks and perform the same mathematical operations. For each layer of the Artificial Neural Network, the following calculation takes place:

$$g(Wx + b)$$

Figure 5: Activation function.

- **x** — *is the input vector with dimension* $[p_l, 1]$

  W — *Is the weight matrix with dimensions* $[p_l, n_l]$ *where,* p_l *is the number of neurons in the previous layer and* n_l *is the number of neurons in the current layer.*

  b — *Is the bias vector with dimension* $[p_l, 1]$

  g — *Is the activation function, which is usually* ***ReLU****.*

  This calculation is repeated for each layer.

And finally, SoftMax layer is added. SoftMax assigns decimal probabilities to each class in a multi-class problem. Those decimal probabilities must add up to 1.0. This additional constraint helps training converge more quickly than it otherwise would. [13]

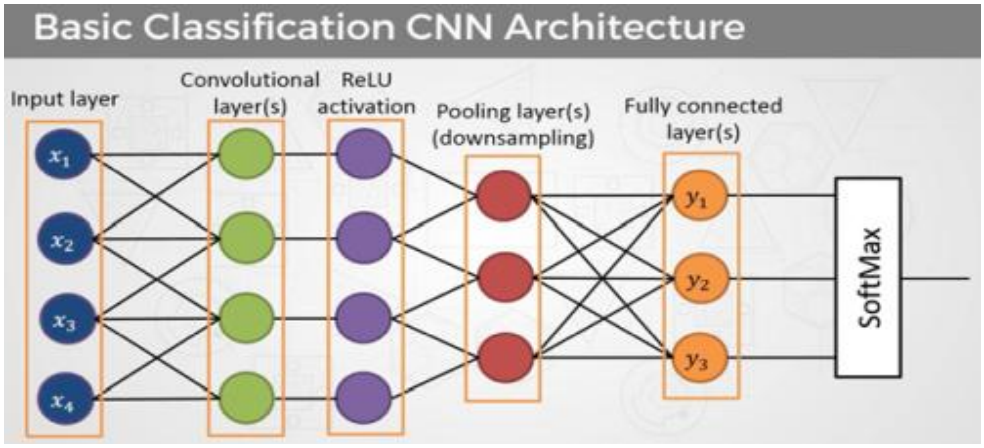

Figure 6: Usage of CNN architecture with different layer types.

7.2. Monte Carlo Tree Search

When implementing AI for computer games, one of the core components is to choose an evaluation function to maximize the quality of game state estimates. For a quite long time, the classic approach is to use heuristic domain knowledge to establish such estimates. Although building a sufficient heuristic evaluation function for a non-terminal game state is time consuming and a complex task.

In the last few years, researchers proposed to use Monte Carlo based techniques as an evaluation function. They have already been applied to games. Yet, this approach remained too slow to achieve a satisfying search depth. Even

more recently, different use cases of Monte Carlo simulations within a tree-search context has emerged. The new technique, which is called Monte Carlo Tree Search (MCTS) was introduced by Rémi Coulom in 2006 [14], is implemented on Go programs. Due to prolific achievements in the challenging game Go, it has gained popularity among studies in AI research. These programs defeated for the first-time professional GO players on the 9×9 board. However, the technique is not specific to Go or class board games but can be generalized easily to modern boardgames or video games.

MCTS is not a classical tree search followed by a Monte Carlo evaluation, but rather a best-first search algorithm, expanding promising regions of the search space much more deeply, that is guided by the outcome of the random game simulations. Furthermore, the evaluation function continues to improve from additional simulations; given infinite memory and computation it will converge on to the optimal game tree. The implementation of the technique is quite straightforward and consist of 4 stages:

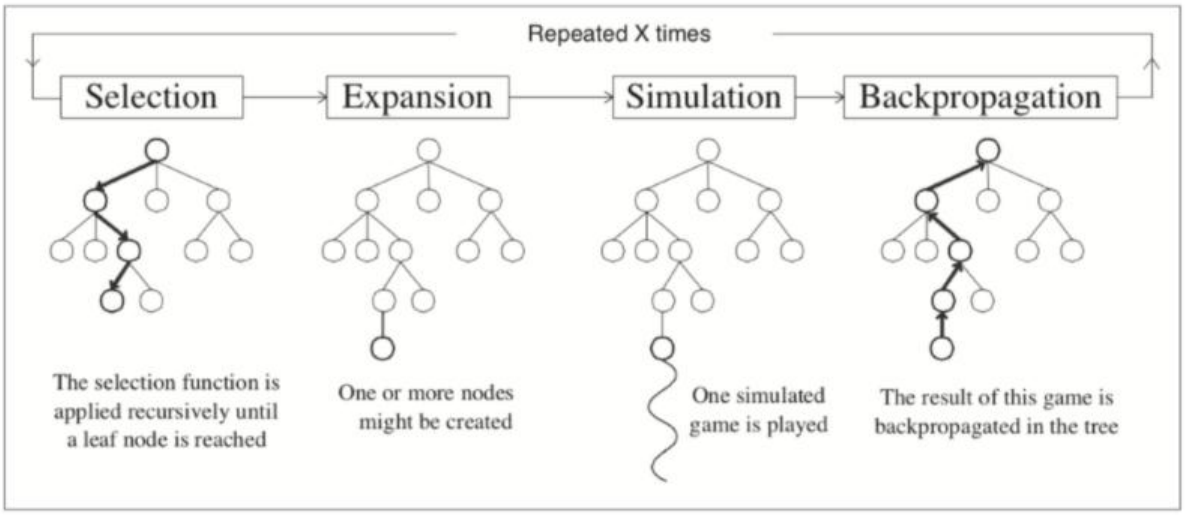

Figure 7: Scheme of a Monte Carlo Tree Search implementation.

1) *Selection:* Starting from the root node, a child selection policy is recursively (or iteratively) applied to traversal down the game tree.

2) *Expansion:* When the game reaches one (or more) state that cannot be found in the tree, then the state is added as a new node. By doing this, the tree is expanded according to the available actions.

3) *Simulation:* A simulation is run from the new node until the end of the game. This task might consist of playing random moves or evaluating the value using neural network.

4) *Backpropagation:* The simulation result, the each tree node that was traversed during simulation will be updated.

7.2.1. *Selection*

Selection is a phase that selects one of the children of a given node. Starting from the root, the tree is traversed until a leaf node encountered. A leaf node is a game state that either contain the end game state or having a visit count of zero (unvisited node). [1]

Every stage of the traversal, the node expanded with the strategy selected for further traversal. The strategy we will use is UCT (Upper Confidence Bound applied to trees). The UCT algorithm, using a carefully chosen default policy, has outperformed previous approaches to search in a variety of challenging games, including Go, General Game Playing, Amazons, Lines of Action, multi-player card games, and real-time strategy games. [15] This strategy is easy to implement and used in many programs. UCT works as follows:

$$\mathrm{UCT}(v_i, v) = Q(v, v_i) + c.P(v, v_i)\sqrt{\frac{N(v)}{1+N(v_i)}}$$

Figure 8: UCT formula

Q (v, $\boldsymbol{v_i}$) corresponds to the mean state value $\boldsymbol{v_i}$ for the node reached by performing action in state v. Which is equal to the Q ($\boldsymbol{v_i}$) / N ($\boldsymbol{v_i}$).

C is a coefficient called exploration constant, which has to be tuned experimentally.

P (v, $\boldsymbol{v_i}$) is prior probability of the move (transition from v to $\boldsymbol{v_i}$), its value comes from the output of neural network.

N (v), N ($\boldsymbol{v_i}$) are total numbers of visits each node v and $\boldsymbol{v_i}$ respectively.

7.2.2. *Expansion*

In expansion phase, for a given leaf node, first encounter of node that cannot be found on the tree, decides whether one or more children of this node will be expanded.

7.2.3. *Simulation*

Simulation (or Playout) is a phase that selects moves in self-play, sequence of moves starts from the current game state until the end of the game where game results can be computed. This task might consist of playing random moves or -better- directly evaluate the current node with a CNN neural network. In our project we will use the help of neural networks to evaluate the position on the game.

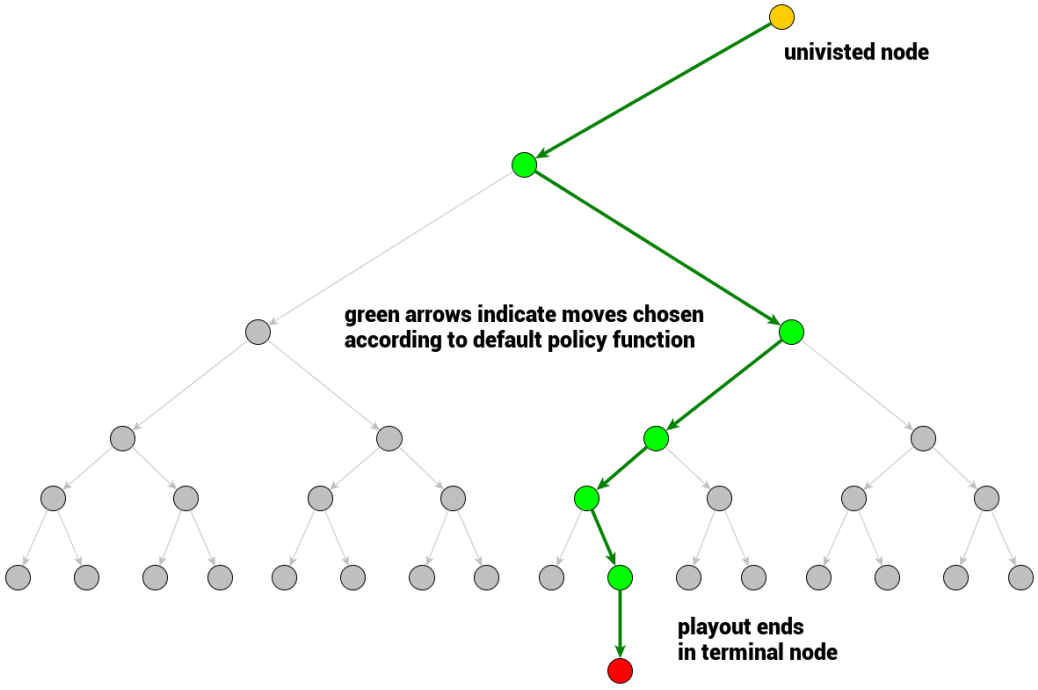

Figure 9: A visualization of playout(simulation) implementation done in MCTS method.

7.2.4. *Backpropagation*

Backpropagation is a phase that propagates the result of the simulation phase backwards from leaf to the nodes that it has traversed during selection phase to reach this leaf node. Simulation result is carried up to the root node and for every node on the backpropagation path certain statistics are computed/updated.

7.3. Self-Play

Learning from zero, a tabula rasa, neural network (Conventional Neural Network) trains with examples obtained from self-play. Input for training of neural network is a list of training examples, where each example is of form (board, pi, v). pi is the MCTS informed policy vector for the given board, and v is its value. The examples have board in its canonical form.

- board: current board in its canonical form
- pi: a policy vector for the current board- a numpy array of length
- v: a float in [-1,1] that gives the value of the current board

After the training, the best current neural network (with its parameters) is saved in folder/filename.

8. Professional Considerations

8.1. Methodological considerations/engineering standards

- We will use Git as a control system during the project. Git is a version control system that provides strong coordination and communication skills for team members.
- The most suitable and widely used language for deep learning applications is Python. We can easily find many source codes to help on platforms like GitHub. Therefore, Python will be used as the programming language in this project.
- Google Colaboratory or AWS (Amazon Web Service) will be used for model development, GPU, and Ram support.
- We can use Jupyter Notebook to run code and develop models locally.
- Google Drive will be used for data transfer and storage.
- Flow chart diagrams will be used to visualize the progress of the project.
- In the modeling process, C++ programming language can be used to get faster results and to increase performance.

8.2. Societal / Ethical Considerations

Some ancestors of this project [13] [16] may be the examples to foresee which ethical impacts this project might have. After DeepBlue and AlphaGo had the impact of winning against a human being as a machine, some ethical debates emerged whether we are truly able to define what machines will be capable of, particularly in the future. An impact resulted from this debate is that a non-profit research institute called "The Future of Life Institute (FLI)" was founded by Max Tegmark, a physics professor at MIT. This institute works to mitigate existential risks facial humanity, particularly from human-level artificial general intelligence (AGI). Not to prevent us from improving our understanding about our brain, self-conscious and intelligence, but to having control over it.

Making a machine succeeding at a game against human may be a small challenge to be put in our way, but eventually it will have a deep impact in our ethical understanding of our lives.

Figure 10: The Beneficial AI 2017 Conference

While doing this project, taking into account social and ethical values, it was taken into consideration not to violate personal data. A social impact, there may be techniques that people will use in strategy games or in business areas where strategy development is important. Economically, this project can reduce game development costs.

The realization of the project does not have a biological effect and an element that may pose a danger to the nature.

8.3. Legal Considerations

Our project does not have a legal consideration. We may need extra information from the user, but no data will be shared with the outside. The researches which we used to deepen our understanding about the concepts described, are free to use licenses. All frameworks - programs that will be used during this project are open source. Therefore, we will not need a license for developing tools.

9. Management Plan

9.1. Description of Task Phases

Phase 1: Literature survey about neural networks, MCTS and AlphaZero. Investigating of methodologies developed so far on related works.

Phase 2: Examining the codes of AlphaZero and AlphaGo techniques written for different strategy games. Observing the results and how the system works.

Phase 3: Deciding the possible outcomes of the final model. Preparing the Project Specification Document.

Phase 4: Implementing the gameplay of the Turkish Checkers game.

Phase 5: Implementation and integration of various CNN models in order to find optimized neural network structure for the game.

Phase 6: Implementing the MCTS structure. Adding features if necessary.

Phase 7: Creating simulations for CNN models to compare the results. Deciding which CNN model is the best for this particular case.

Phase 8: Competing our model against real time opponents. Setting up a connection between model and real time system. Getting feedback by those games against real opponents and ranking our model.

Phase 9: Make model improvements and optimizations.

Phase 10: Development and improvement of the gameplay structure.

Phase 11: Testing whole model.

Phase 12: Presentation of the project.

9.2. Division of responsibilities and duties among team members

Division of labor is shown below at Figure 11.

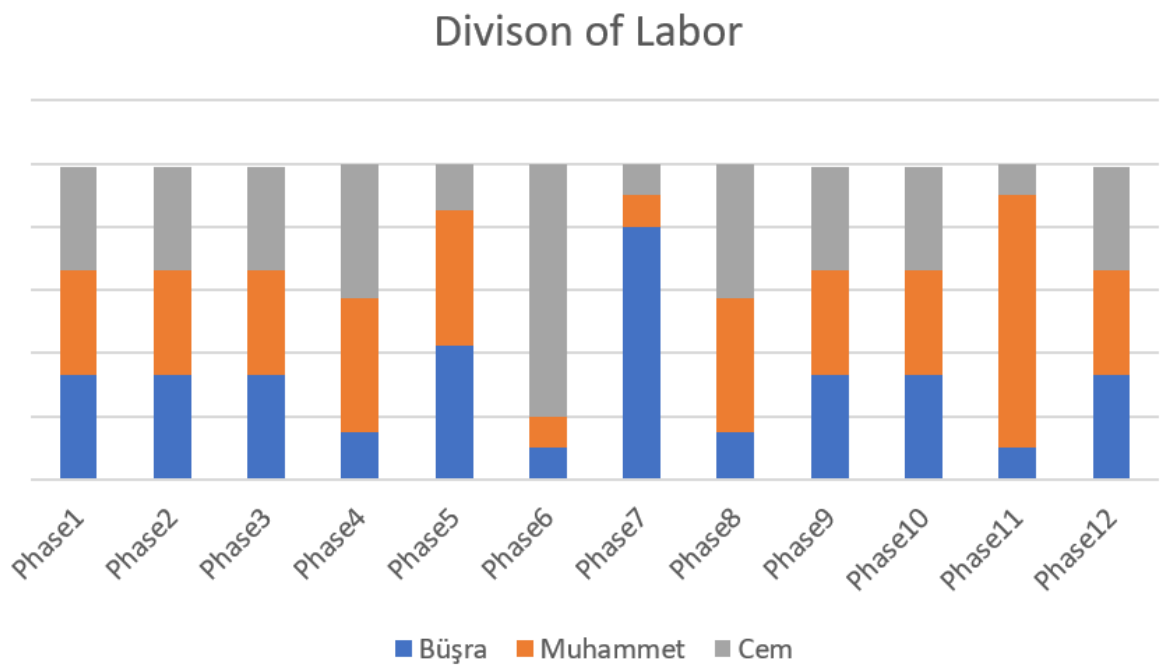

Figure 11: Clustered Stacked Bar Chart for the division of labor

9.3. Timeline

Phase	Start Date	Time Spent (Days)
Phase1	30.08.2020	30
Phase2	30.09.2020	30
Phase3	30.10.2020	30
Phase4	30.11.2020	21
Phase5	21.12.2020	30
Phase6	18.01.2021	21
Phase7	8.02.2021	21
Phase8	1.03.2021	21
Phase9	22.03.2021	30
Phase10	19.04.2021	30
Phase11	17.05.2021	15
Phase12	31.05.2021	21

Figure 12: Detailed information

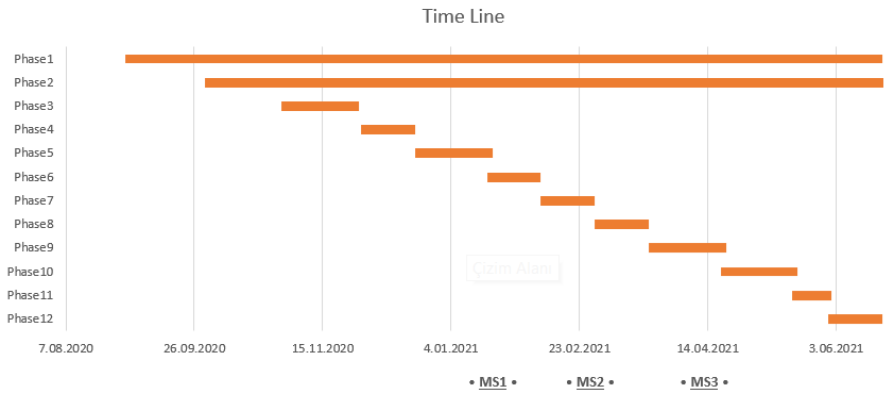

Figure 13: GANTT Chart for describing the timeline

Gantt chart and its detailed information is shown above at Figure 13 & 14.

Milestones are listed below:

- The first milestone (•MS1• on the Figure 13) is right after the phase 5. Until phase 5, we will complete our neural network and game structure code. That will be our initial basis point to our project.
- The second milestone (•MS2• on the Figure 13) is between phase 7 and 8. Around those times we will improve our initial basis with the results obtained from phases.
- The third milestone (•MS3• on the Figure 13) is between phase 9 and 10. At that time, we will get feedbacks from the real world. Then from those feedbacks, we will improve our neural network and gameplay structure.

9.4. Risk management

- The system cannot produce a reasonable move in a given amount of time.
    - *Probability: Low*

  In order to system to work under given conditions, the depth of search tree may be limited. This will affect the playing style in the long run, but we must take the necessary precautions for the system to work in a given time limit.

- The system is not a top-rated checkers program.
    - *Probability: Low*

  The system gets stronger by each play. The more the games it plays, the more intelligence system gets. This is technically somewhat true, but the system level also can be depending on the structure behind the gameplay. To ensure that system works in top-shape, we will use the most beneficial structure that works on checkers.

References

[1] F. Carlsson, J. Öhman, "AlphaZero to Alpha Hero: A pre-study on Additional Tree Sampling within Self-Play Reinforcement Learning", in Digitala Vatenskapliga Arkivet, p. 42, 2019 (http://www.diva-portal.org/smash/get/diva2:1350740/FULLTEXT01.pdf)

[2] I. Bratko, "AlphaZero - What's Missing", in Informatica: International Journal of Computing and Informatics, vol. 42, no. 1, 2018

[3] Murray Campbell, A.Joseph Hoane, Feng-hsiung Hsu, "Deep Blue", Artificial Intelligence, Volume 134, Issues 1–2, 2002.

[4] David Silver, Thomas Hubert, Julian Schrittwieser, "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm", Science

[5] Schaeffer, Jonathan & Björnsson, Yngvi & Kishimoto, Akihiro & Müller, Martin & Lake, Robert & Lu, Paul & Sutphen, Steve. (2007). Checkers Is Solved. Science. 317. 1518-1522. 10.1126/science.1144079.

[6] Kumar Chellapilla and David B. Fogel, Fellow, IEEE Transactions on Neural Networks, "Evolving Neural Networks to Play Checkers Without Relying on Expert Knowledge", VOL. 10, NO. 6, NOVEMBER 1999

[7] Dubel, Amco & Brandsema, Jaap & Lefakis, L.. (2006). Reinforcement learning project: AI Checkers Player.

[8] K. Chellapilla, D. B. Fogel, "Evolving an expert checkers playing program without using human expertise", in IEEE, vol. 5, issue 4, pp. 422-428, Aug 2001

[9] K. Chellapilla, D. B. Fogel, "Anaconda defeats Hoyle 6-0: a case study competing an evolved checkers program against commercially available software", in IEEE, July 2000.

[10] Wikipedia, "Turkish draughts"

Available: (https://en.wikipedia.org/wiki/Turkish_draughts)

[11] David Foster, "How to build your own AlphaZero AI using Python and Keras", January 2018

Available: (https://medium.com/applied-data-science/how-to-build-your-own-alphazero-ai-using-python-and-keras-7f664945c188)

[12] Ayyuce Kizrak, “Evrişimli Sinir Ağları”, May 2018

Available: (https://ayyucekizrak.medium.com/deri%CC%87ne-daha-deri%CC%87ne-evri%C5%9Fimli-sinir-a%C4%9Flar%C4%B1-2813a2c8b2a9)

[13] Google Developers, “Machine Learning Crash Course - Multi-Class Neural Networks: Softmax”

Available: (https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax)

[14] Rémi Coulom. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. 5th International Conference on Computer and Games, May 2006, Turin, Italy.

[15] Sylvain Gelly, David Silver, Monte-Carlo tree search and rapid action value estimation in computer Go, 2011.

[16] Silver, D., Huang, A., Maddison, C. et al. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489 (2016).