T.C.

MARMARA UNIVERSITY

FACULTY of ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

CSE4197 – Analysis and Design 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

15 February 2021

Table of Contents

1. Introduction

Over the last years, deep learning has 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 ML, due to the game's popularity worldwide it is unfortunately not able to draw the researchers' focus back. Main aim of this project is to solve the game of Turkish Checkers by combining those fields in a sophisticated and modern way.

1.1 Problem Description and Motivation

For centuries, playing mind games has gained an important place in human habits. The oldest mind and strategy games date back to 2000 BC.

Features strategy games created an important area of investigation for studies on the approach of artificial intelligence. Chess, Go and Checkers games provided an efficient testing environment for studying the learning, managing and decision-making aspects of AI systems. The fact that these games are popular in the world and that they are played widely has contributed to its selection as a field of study. Another factor is that these games have a set of rules that restrict players' appropriate behavior (i.e. legal moves), thus simplifying the problem in that situation. The third factor is that these strategy games have a goal to win the game for the players to reach. Prizes are awarded to players who demonstrate the best and correct game play strategies under the constraints of limited resources that allow players to achieve the set goal. The last factor is that strategy games have sufficient characteristics to authorize a wide variety of and different complex behaviors represented by the different environments of the players [1].

Great progress has been made with artificial intelligence methods developed using supervised learning systems trained to learn and copy human experts' decisions. However, expert datasets can often be unreliable, expensive, or difficult to access. Even when reliable data sets are available, they can put a constraint on the performance of systems trained in this way. However, RL systems are trained in principle from their own experience, allowing them to exceed human capabilities and operate in areas where human

expertise is inadequate. [2]. 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 processor of AlphaZero is AlphaGo. AlphaGo is produced and optimized for the Go game. This program succeeded in defeating one of the world's most talented Go players, Lee Sedol, in a five game Go game. This was the first time a computer program was able to beat a human expert player in Go. AlphaGo specializes in Go and has been learned from exemplary high quality Go games previously played by human expert players; AlphaZero, on the other hand, is a general strategy game playing program that does not specialize in Go. It learned Go by itself and outperformed all versions of AlphaGo consistently. 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 [3].

The basic structure of AlphaZero is as follows. AlphaZero learns the game from simulated games with RL against its predecessor. Here it uses a deep CNN structure that learns to predict the values of the positions of the game and the probability of players playing the possible moves in a position. AlphaZero chooses a move to play with the MCTS from its current board position. Searching with MCTS involves the simulation of random games from current positions where the random move probabilities increase with the move probabilities rotated by the neural network and the moves decrease with the number of visits [4].

The "social factor" of game playing should not be underestimated. And that’s why one of the motives in our project is to increase playing time of the Turkish Checkers. The reason why the numbers decreased over time is that people didn't have a good competitor that would give a chance to improve their playing style, to change the way they look at the game, and to get challenged. We hope to meet their expectations with new era technologies.

The other one is to inspire a new state of surveys on Turkish Checkers with AlphaZero. To our knowledge there are not any studies that try 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, we hope to attract Turkish researchers attention and thus increase the number of literature surveys that focus on Turkish Checkers.

Based on those motives we believe this study is important and valuable.

1.2 Scope

Our primary purpose in this project is 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 the 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 ANN [5] and FFNN [6], but the solutions they are offering are outdated compared to current technologies. That is why we decided to focus on using CNNs in our project.

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 the computer's processing power.
- System should provide an output within a short period of time.
- The output of the system should be reasonable.

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, the model will play the most promising move.
- It is assumed that the system will be more efficient compared to given works in Section 4.

1.3 Definitions, acronyms, and abbreviations

AGI: Artificial General Intelligence
AI: Artificial Intelligence
ANN: Artificial Neural Network
API: Application Programming Interface
CNN: Convolutional Neural Network
DNN: Deep Neural Network
FFNN: Feed-Forward Neural Network
MCTS: Monte Carlo Tree Search
MVC: Model View Controller. It is an architectural pattern that separates an application into components. Mostly used as a web development framework.
ML: Machine Learning
NN: Neural Network

RL: Reinforcement Learning
TD: Temporal Difference learning
UCT: Upper Confidence Bound applied to trees
WAF: Web Application Firewall

2. Literature Survey

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 benefited 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 [4], authors work on creating a chess playing system with the sole purpose to defeat the human world champion in a regulation match. The Deep Blue project is the culmination of about 15-years of 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 enhanced 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, pet 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 using the newer generation of SP computers 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 were done and optimized the system requirements, Deep Blue II won against world champion Garry Kasparov in a six-game match in 1997.

This is written into history as the 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 [7], authors work on a game-playing program that replaces traditional game-playing programs with a DNN, RL 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 trees, they use a general-purpose 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 after 30 hours (74,000 steps).

In the work of Kumar Chellapilla and David B. [5], authors work on checkers gameplay where NNs used to compete with each other. NN was structured as a fully connected FFNN with an input layer, two hidden layers, and an output node. NN gives a scalar value interpreted as the value of that board from the player's position, whose pieces are represented 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 NN evolved where it had no information about the nature of the game; simply a vector rather than a board. Despite this obstacle, NN was able to learn to play checkers competently, mainly based on the information it contained from just "win, lose, or draw".

In the work of I. C. L. Dubel, Ing. J. Brandsema, L. Lefakis, S. Szóstkiewicz [8], authors work on several methods such as RL, Monte Carlo method, NN, and Temporal Difference-leaf to make the model learn how to succeed at game checkers. They apply several policies such as greedy, Ɛ-greedy and softmax action selection policies to study the effect of different rates of exploration vs. exploitation. After experimenting those different situations with different policies, experiments conducted using reinforcement learning concerning 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.

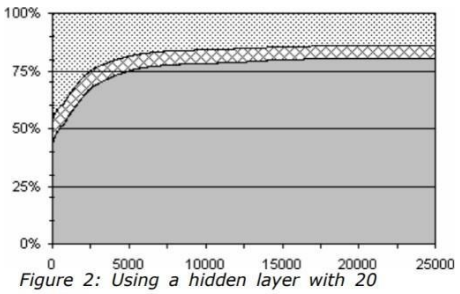

Figure 2: Using a hidden layer with 20 neurons.

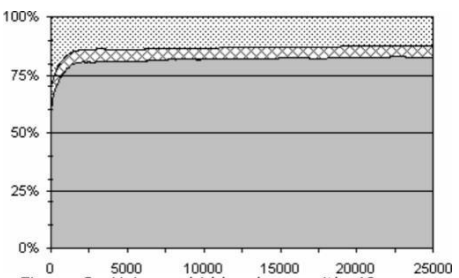

Figure 3: Using a hidden layer with 40 neurons.

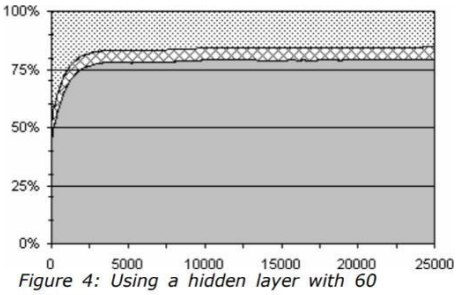

Figure 4: Using a hidden layer with 60 neurons.

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

* Agent using look-ahead vs. Random player: Here, the agent is able to look ahead of the current game state to a specific depth in the game tree and chooses an action based on the min-max standards. By having the ability to look-ahead of the game the agent 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 the agent looks ahead to a depth of three moves to find the best possible leaf-state value. After finding the best leaf-state based on the minimax standards, it 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.

3. Project Requirements

3.1. Functional Requirements

i. It should have an easy-to-use game interface for users.
- Users should be able to see the opposing player's move.
- Users should be able to see who won the game and the score.

ii. In order to test the game on different sites, the game program must be suitable for API use.

3.2. Nonfunctional Requirements

3.2.1. Security Requirements

From the beginning to the end of the API design and development phase, the principle of seamless integration with the WAF, API management solution, API gateway and other tools that will be required to keep the API secure should be worked on.

3.2.2. Performance Requirements

i. There shouldn't be much delay when a move is made. The opposite

player should be able to see the move as fast as possible.

ii. The data received using the API must reach quickly.

3.2.3. Usability Requirements

The game interface must be available.

3.2.4. Maintainability Requirements

If the player makes an illegal move, the message should be returned to the user that it was a wrong move. Possible errors that will occur during the game should also be checked in the source code.

4. System Design

4.1 UML Use Case Diagram

Following diagram shows us that inside of our program there will be two main actors: Player and the AI system. Player represents the actual human player that will compete against our model. Beside from simple game playing actions (starting the game, making a move, etc.) human players should be able to decide the opponent models level. The diagram is given in Figure 2.

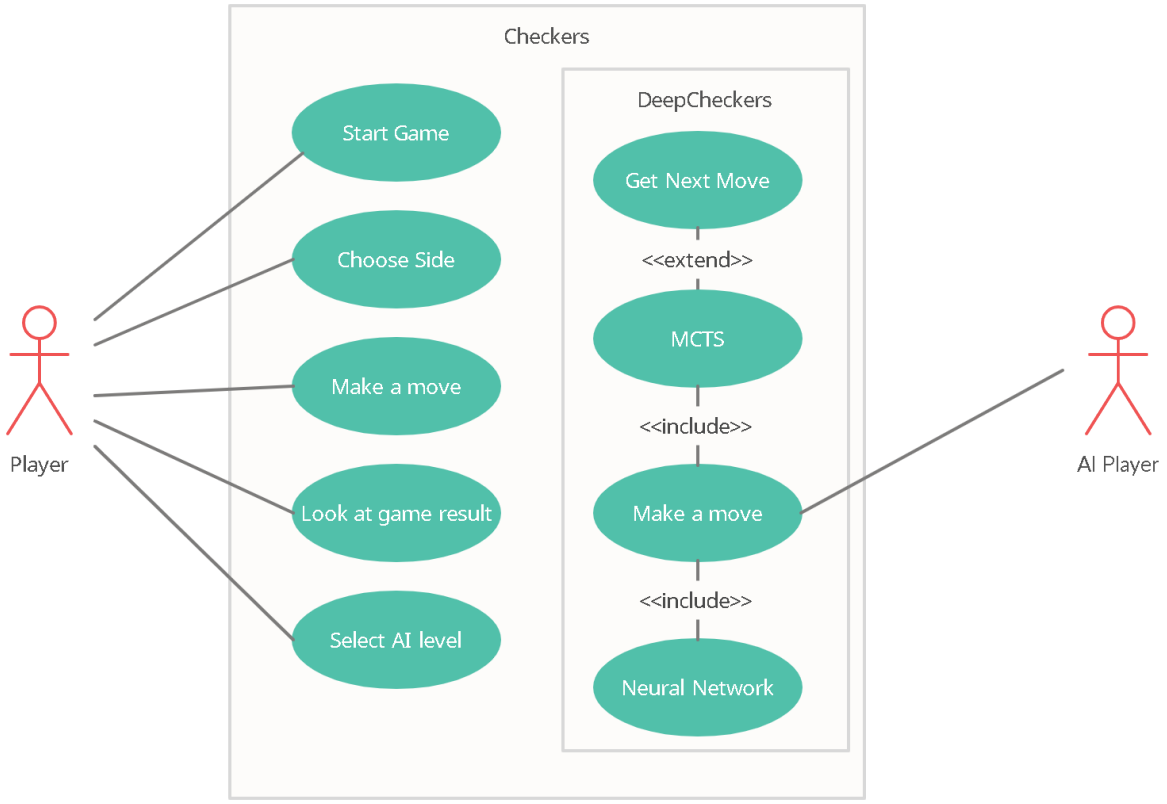

Figure 2: UML Use Case Diagram

4.2 Database ER diagrams

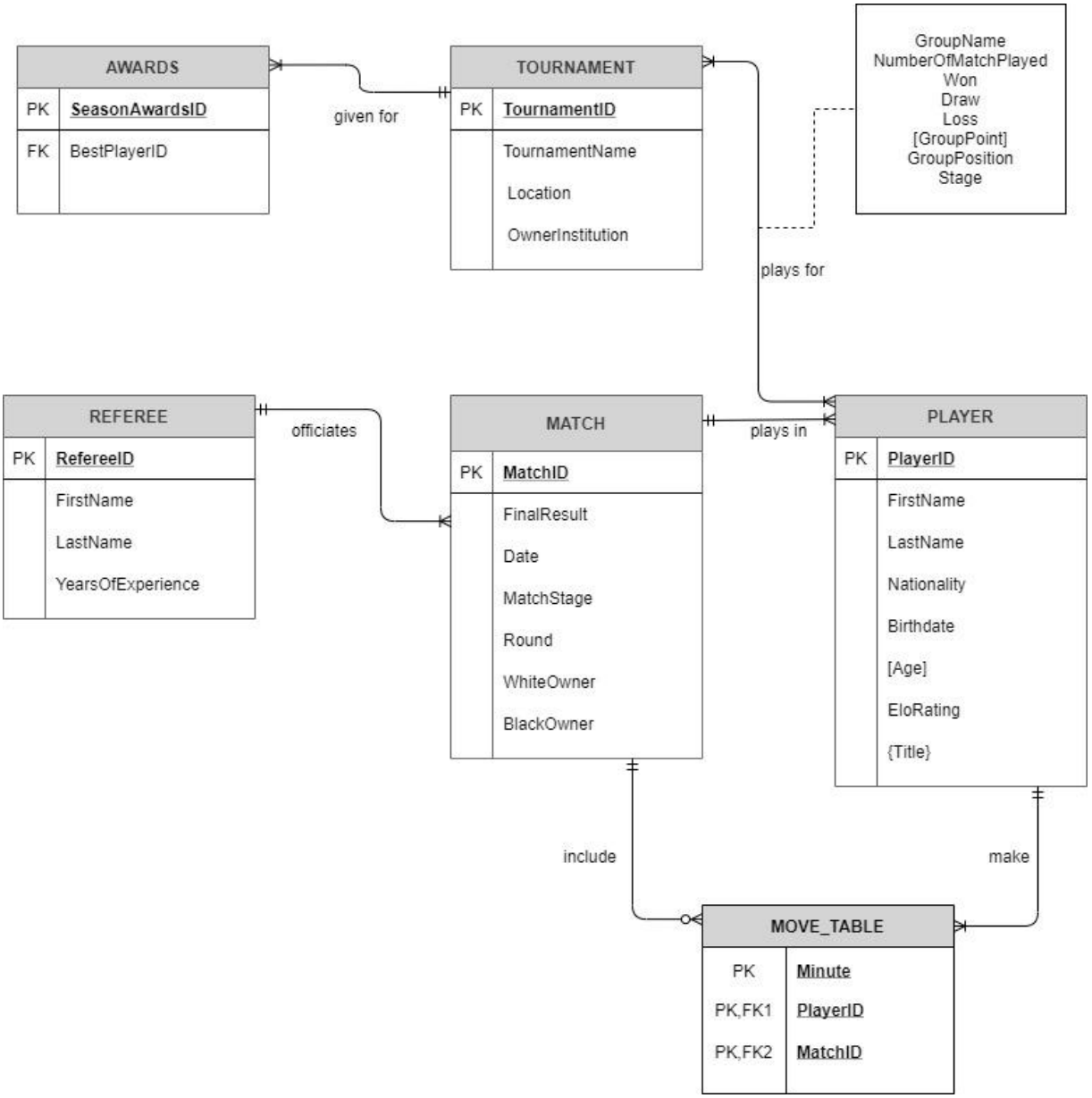

Figure 3: Database ER Diagram

4.3 User Interface

In our project the platform will be built based on MVC architecture. The architecture is given in figure 4.

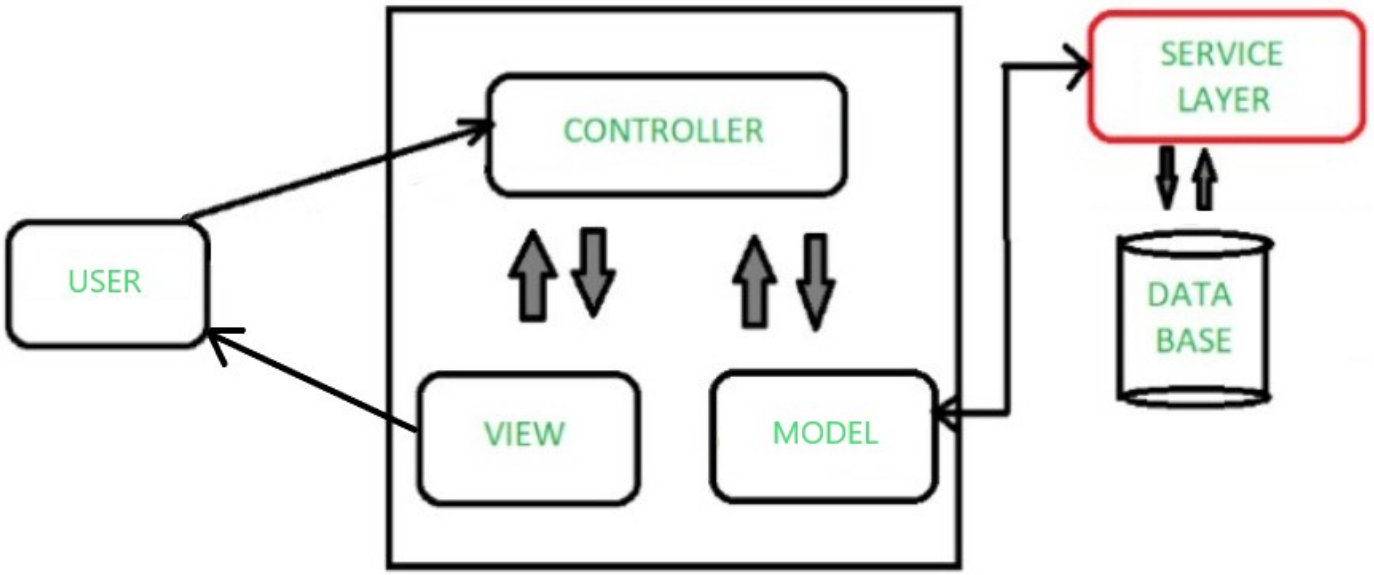

Figure 4: MVC Architecture

MVC will help us to separate our project into three main parts: the model, the view, and the controller.

- Model Component: This component corresponds to all data-related logic that the user works with. In our project this part will be the decisions that the user will choose on the view component.
- View Component: This component is used for all display operations. For example in our project users will be able to choose the AI player level, game time and game screen.
- Controller Component: This component acts as an interface between model and view components. In our project main task for this component will be to handle the user's actions.

Our project will consist of two pages. The main page and the game page.
In the main page, user will be able to select the time limit of the game, and the difficulty level of the AI player. Example of this page is given in Figure 5[9].

Figure 5: Example of Settings Page

The game page will be the page that gameplay will occur. In this page user will be able to make a move and see the current state of the game, and the result of the game. Example of this page is given in Figure 6[10].

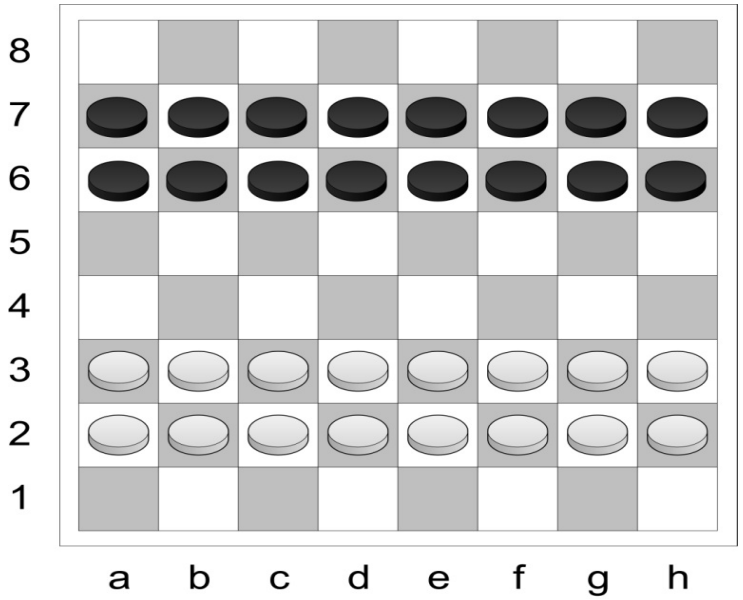

Figure 6: Example of Game Page

4.4 Test Plan

In this section, we consider methods and approaches to testing our project. The purpose of this test plan is to play the game according to its rules and to ensure its usability by the player. We have developed some test scenarios to achieve our goal.

Test #1

TEST CASE:	Simultaneous play
Requirement:	The moves made on the game interface should be displayed instantly
Procedure:	The coordinates for the move are taken as input.
Expected Result :	The status of the checkers according to the coordinates is displayed in the interface.

Test #2

TEST CASE:	Proper Move
Requirement:	It must be checked whether the player's move is legal.
Procedure:	Player move is taken as input
Expected Result :	If the move is suitable for the game, it is the AI to move. If the move is not suitable for the game, a warning message will be shown.

Test #3

TEST CASE:	API Connection
Requirement:	Using the data obtained from the game site instantly with the help of API and making the AI move
Procedure:	The move from the human player on the game site is taken as input with the API
Expected Result :	The model moves according to the coordinates coming with the API and the coordinates of this move are transmitted to the interface.

Test #4

TEST CASE:	Starting a New Game
Requirement:	When the previous game is finished, the new game should be automatically switched to, and the API connection at the back and the game interface should be renewed.
Procedure:	The last move for the first game is taken as input.
Expected Result :	A new game screen is shown

5. Software Architecture

Data Flow:
In our project users are able to login/register an account. Also users can select the game options as they desire such as the difficulty level of the AI player. The complete data flow diagram can be seen below in figure 7.

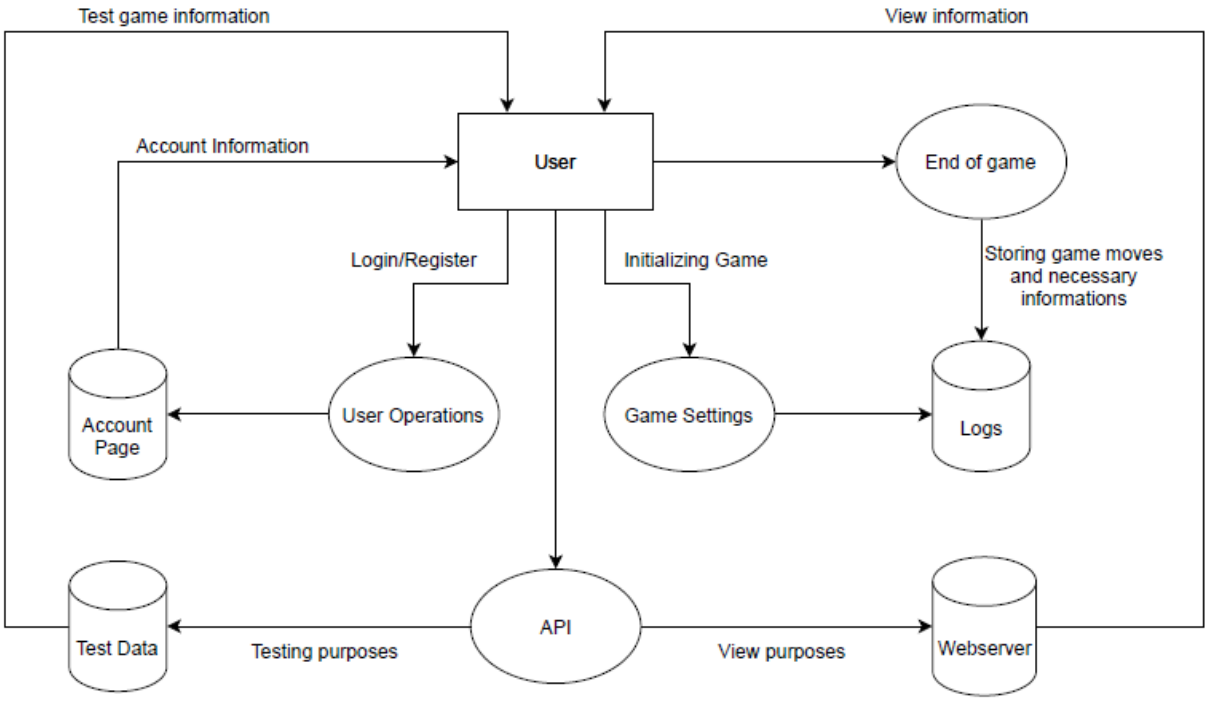

Figure 7: Data Flow Diagram

Control Flow:

There are several control mechanisms that need to be explained further, in order to understand how the system control flow really works.
Firstly, until the user presses the start button, which starts the game, the menu waits for further commands.
For the process of starting the game, selected settings at the menu (which are difficulty, game type, move time etc.) are configured with the corresponding model (which is based on the difficulty level).
After having the settings configured and an appropriate model is prepared for the match, the system waits for white side, whether it is the ai model or the user, to make a move.
If the move made is legal, under the rules of Turkish checkers, it is accepted to be updated on the board accordingly.
This whole process repeats itself until someone defeats some other player.
Victory of a player is decided whether the opponent ran out of time or defeated by the game rules.

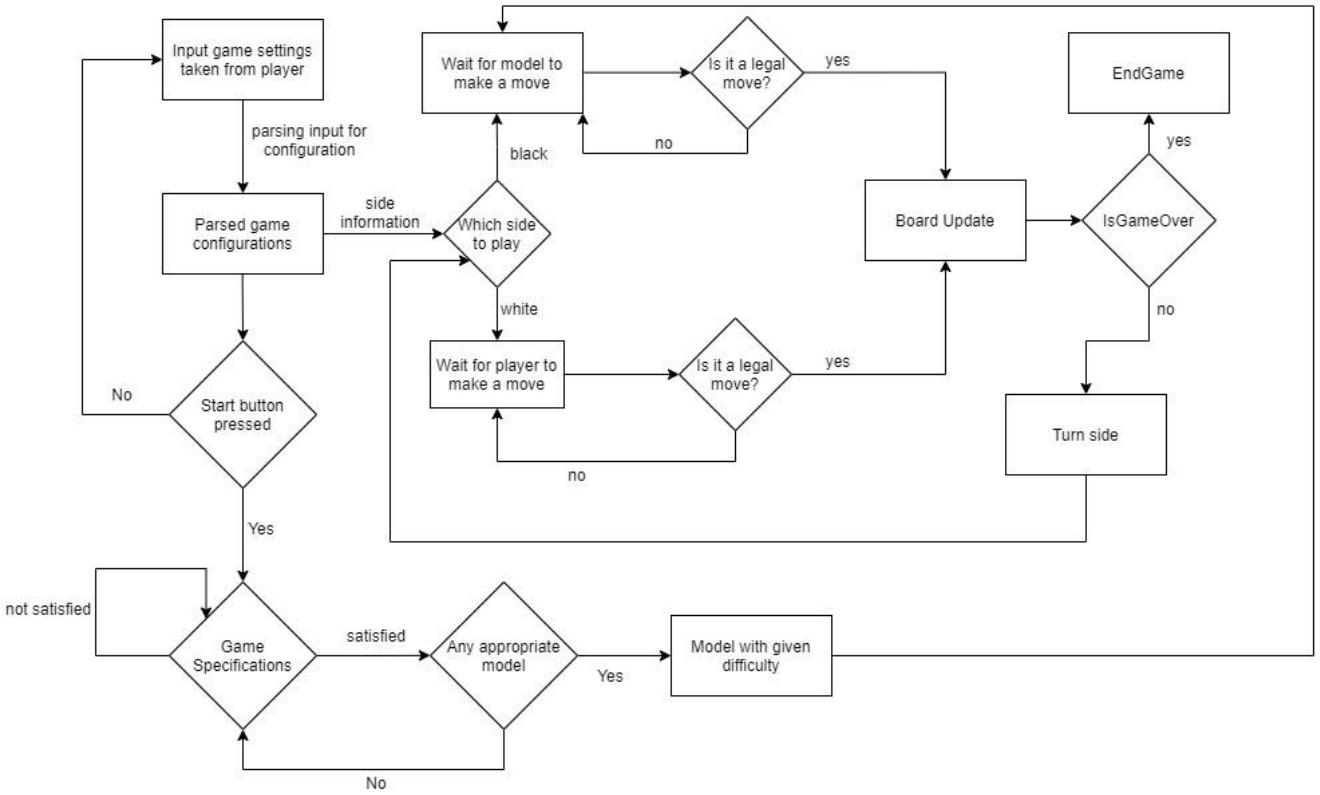

Figure 8: Control Flow Diagram

Modular Design:

The system is consist of three parts :

- **User Interface/Front-End:** Users will interact with our platform via UI layer. Each UI object has a seperate controller.
- **Backend System:** This is the part of our model, API, a test database which contains some games that have already been played, and a system which our model will be working on (Google Colab).
- **Web Browser:** This part contains a web application that contains our platform and provides UI for users. In this part user interactions will be held.

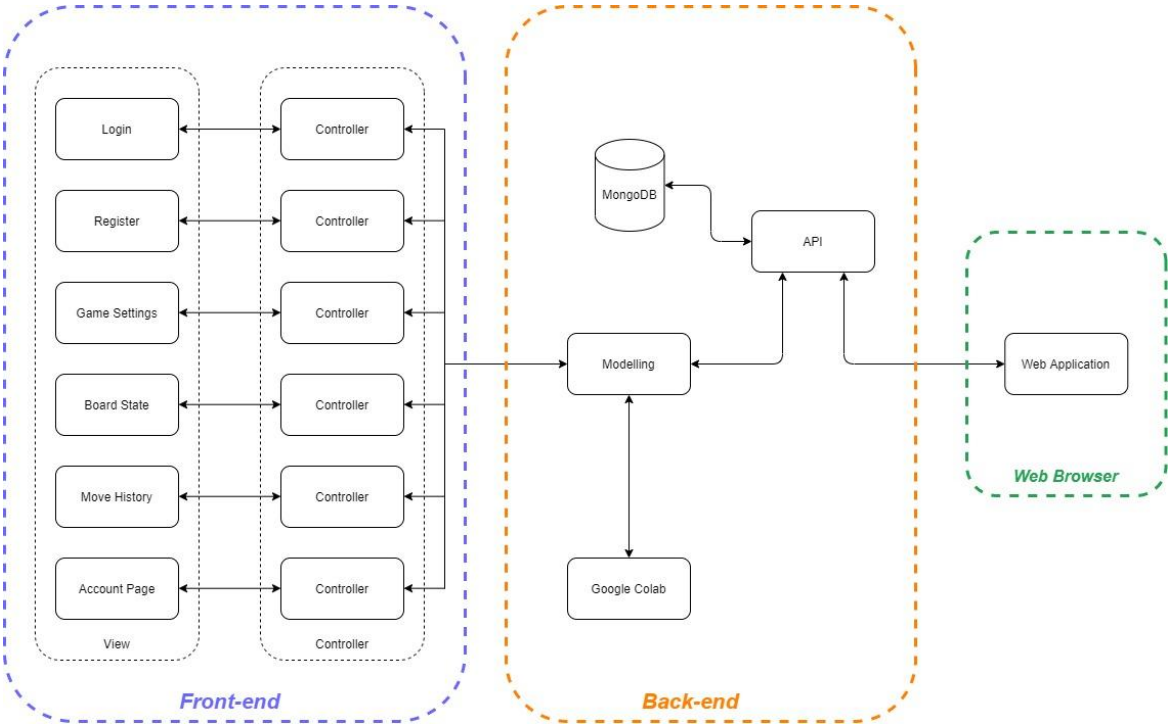

Figure 9: Modular Design

6. Task Accomplished

6.1 Current state of the project

Our completed tasks are as follows:

- We finished our research on convolutional neural networks and MCTS models and developed the structure of our system.
- We did research on how to prevent errors when they occur.
- We created the basic structure of our Deep Learning model. We will make improvements on it.
- We created the MCTS structure.
- We did research on how to use API connection.

6.2 Task Log

During this semester, we had our meetings with our advisor every two to three weeks. In these meetings we reported our progress through weeks and discussed the topics we are having problems with. Some important notes from these meetings are given below:

- **First Month**

We started this project in order to find a good solution to our problem. To do that, we needed to research these topics and examine them. This topic was a new era for us, and some researches were outdated, so we had lots of problems with the learning process.

- **Second Month**

After such a rough learning process, we started with the system design and discussed which algorithms or architectures we are going to use. We decided to use MCTS as a game tree search algorithm, and a NN (starting with CNN) to improve our decision making in the game. Then we started to examine similar projects using these methods and discussed what our difference will be. Also we made some plans on how to improve our project.

- **Third Month**

This month we had examined how MCTS works and looked at some sample project codes. We have decided the programming languages and frameworks for the project. We implemented the game structure, and we started to implement the MCTS algorithm.

- **Fourth Month**

We continued to implement our project. We tried to connect the game and MCTS structures. We designed a simple CNN that we plan to use in the future.

6.3 Task Plan with Milestones

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 the whole model.

Phase 12: Preparation of the final poster and finalizing project.

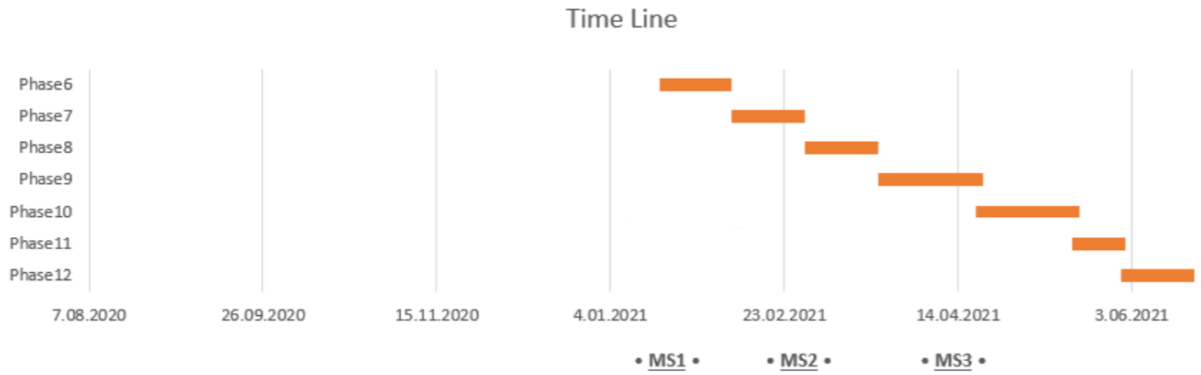

Figure 10: Timelines and milestones for the second semester

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

7. References

[1] S. Y. Chong, M. K. Tan and J. D. White, "Observing the evolution of neural networks learning to play the game of Othello," in IEEE Transactions on Evolutionary Computation, vol. 9, no. 3, pp. 240-251, June 2005, doi: 10.1109/TEVC.2005.843750.

[2] Silver, D., Schrittwieser, J., Simonyan, K. *et al.* Mastering the game of Go without human knowledge. *Nature* 550, 354–359 (2017). https://doi.org/10.1038/nature24270

[3] 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)

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

[5] 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

[6] 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.

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

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

[9]https://apkpure.com/tr/chess-free-learn%E2%99%9E-strategy-board-game/com.mggames.chess

[10]Wikipedia, "Turkish draughts"

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