NBA Franchise Manager - how to build a championship team

Gabriel Bueno de Moraes Fior
Kellogg College

Supervised by
Dr. Max Van Kleek

University of Oxford
A dissertation submitted for the MSc in Software Engineering

Michaelmas 2022

Abstract

The game of basketball has been recently revolutionized by analytics, with innovation mainly focusing on predictions of game outcomes as well as player- and team performance. Little attention has been devoted to the performance optimization of the decisions made by NBA front office personnel concerning team management as a whole, which is a very complex task and involves several interconnected actions, such as player development, optimal draft strategy and finally trade execution.

This dissertation proposes a decision support system for NBA front offices, which will assist such professionals in making well-informed decisions having access to recommendations based on statistical data. These recommendations arise from the results of a Monte Carlo simulation, where 5 NBA seasons will be simulated many times and the best scenarios for a given franchise will be highlighted, in order to understand the actions that led to the best results.

In this work, several components mimicking real NBA processes, such as trades involving players, evolution of draft picks and outcomes of games, will be implemented and discussed. We will also make the case that, even though simplifications are in place, the results are very much aligned with expert expectations and provide reasonable input for NBA decision makers.

Finally, we will also reflect on the implementation carried out for fulfilling the requirements and also suggest possible improvements as future work.

Acknowledgements

I would like to thank my supervisor Prof. Max van Kleek for his continued support during my thesis, including the initial exchange of ideas after my first course at Oxford and the very helpful comments throughout the writing phase.

I also would like to thank all my lecturers from the Software Engineering Program. They were all very engaged in teaching their subjects and allowed me to deeply understand several areas of the study program.

I also would like to thank d-fine GmbH for the financial support and for making the necessary arrangements for me to take part in this study program.

I also would like to thank my many basketball coaches, specially Jece Leite, for teaching me everything about basketball.

Finally, I thank Aline for her endless patience and support through this incredible journey. I also thank my parents for always making me believe I could reach all my dreams, no matter how far-fetched they seemed to be.

Declaration

Except as acknowledged, attributed, and referenced, I declare that this dissertation is my own unaided work.

Signature	*Date*

Table of Contents

List of Figures

Chapter 1

Introduction and Motivation

There is no single recipe for building championship teams in the NBA. Everyone has their way: homegrown talent (draft), signing external talent (trades, free agents), or a combination of those. This thesis tries to shed some light on a few key items that contribute to championship teams.

This chapter introduces the problem we are trying to solve, why it's important, and the methods we will employ to conduct our investigation.

1.1 Problem domain

The goal of this thesis is to advise on optimal ways in which an NBA roster can be assembled to win a championship. To achieve this, we built a decision support system that allows NBA general managers to simulate common decisions for a given season (e.g., draft picks, trades) and observe the most likely outputs for each team, described by the final standings. Thus we produce a tool for NBA decision-makers to "practice" strategies.

We acknowledge it is impossible to replicate a general manager's daily decisions exactly. Let's consider a player traded from team A to team B. The performance of the said player depends on many factors that cannot be easily quantified, such as team culture and team-player relationship, among others. However, by building a decision support system like the one described in this work, we were able to draw promising results (as seen in chapter 5) that can be used as part of the decision-making process of NBA front offices, which can leverage decision-making reproducible analysis.

1.2 Motivation

The motivation for this work was a perceived lack of studies concerning the role of the NBA front office personnel when discussing each team's chances of winning a championship. The media and the general public focus a lot of time and resources on dissecting all the possible factors affecting players and rosters, especially in the off-season. Therefore, we wanted to investigate if a decision-support system for this target group could be built and, if indeed possible, what parameters, requirements, and especially limitations would need to be considered for it to be helpful.

Another source of motivation is the generality of such a system. Other American major leagues, such as the Major League Baseball (MLB) or the National Football League (NFL), could use the tool as another input for their decision-making process. This is possible given that the general concepts of the leagues are similar. Background information on the NBA will be given in chapter 2, but as a general overview, players are assembled in rosters, which compete for a championship on the court, and players can be signed, traded or drafted from college or overseas. These simplified considerations also apply to other leagues, so that if we build the system in a modular and extensible way, we don't see a reason why the system could not be adapted for other sports.

1.3 Organization of the thesis

The organization of the remaining chapters is as follows:

Chapter 2 provides a thorough explanation of NBA basketball, how the league organizes its schedule and teams, and especially how rosters are assembled, i.e., how teams can sign, trade, and draft players and how much each team is allowed to spend on its roster.

Chapter 3 discusses the methodology employed in this thesis for building a system that is able to simulate NBA seasons in the most realistic way. It also determines what success looks like, how we will benchmark our results against actual results, and, most importantly, what design we will strive for when building such a system.

Chapter 4 explains in detail what design we decided to implement, its advantages and disadvantages, and the challenges we ran into while implementing the proposed design. It starts with a high-level explanation of the system and then continues with a detailed description of each component of the system.

Chapter 5 presents the results obtained after performing a Monte-Carlo simulation consisting of 1000 scenarios, whereas each scenario consists of 5 consecutive NBA seasons. We detail our findings regarding roster construction, focusing on a single team. We also evaluate how the implementation of the system satisfied the requirements proposed when discussing the methodology.

Chapter 6 introduces social and ethical considerations regarding this work. It discusses potential effects on fans and ultimately on NBA games in general, and also elaborates on the social effects of Analytics on the game of basketball.

Chapter 7 summarizes the conclusions of this thesis and enumerates interesting avenues to be explored in future work.

Chapter 2

Background

This chapter introduces practical basketball concepts, specifically a description of the National Basketball Association (NBA) season and the associated rules. Please note that the rules of basketball, such as the number of players, dimensions of the court, etc. are out of the scope of this thesis.

We explain below concepts that will be important later as we describe how our simulation tool works.

2.1 League structure

The NBA is the most prestigious basketball league in the world, putting together the best players and millions of fans worldwide. The league currently has 30 teams, divided into two conferences (East and West). Each conference is divided into three divisions of 5 teams apiece.

Each team can have between 11 and 13 players in its game roster (also called active list), whereas additional players can be included in the inactive list, thus not able to play in that particular game.

More detailed information regarding the league structure can be found on [10].

2.2 Schedule

The NBA schedule is divided into a regular season (comprised of 82 games) and the playoffs, where the top 8 teams of each conference play best-of-seven series until the champion has been decided, meaning that the champion must win 16 playoff games in order to claim the championship.

Each team plays 41 home and 41 away games during the regular season. In total, each team plays each other team from its division 4 times, from other divisions in its conference either three or four times and teams from the other conference two times.

The scheduling process is rather complicated, as multiple factors need to be observed in order to produce the final dates:

- The league aims to reduce the distance traveled by each team to a minimum. For that reason, it schedules multiple games hosted in one specific region within a few days of each other. For example, if the Miami Heat is scheduled to play the Los Angeles Lakers on the 1st of January, it would be optimal if they could also play the Los Angeles Clippers, the Sacramento Kings and the Golden State Warriors during the same trip, as all those teams play in California.
- The league aims to reduce the number of games played on consecutive nights ("back-to-backs") to avoid injuries.
- Normally, NBA teams share arenas with other sports and events, thus, are unable to play at their arenas on event dates. For example, the San Antonio Spurs have an annual road trip that coincides with the annual San Antonio Rodeo, forcing them out of town for roughly 20 days.

Between the regular season and the playoffs, starting from the 2019-20 season there has been a play-in for defining the final two teams that participate in the playoffs. For each conference, the teams that finished in places 7, 8, 9 and 10 (denoted as teams 7,8,9 and 10) compete as follows for the last two spots:

1. Teams 7 and 8 play a game in the home court of team 7. The winning team advances as the 7th-ranked team into the playoffs.
2. Teams 9 and 10 play an elimination game in the home court of team 9. The losing team is eliminated.
3. The loser of the game between teams 7 and 8 and the winner of the game between teams 9 and 10 play an elimination game on the home court of the best-ranked team. The winner advances to the playoffs as the 8th-ranked team.

Finally, each team must win four games to advance during the playoffs. The distribution of games in each court is 2-2-1-1-1, where the best-ranked team hosts up to 4 games on its court. For example, if the best-ranked team (team 1) is playing against the eighth-ranked team (team 8), the first two games are hosted by team 1, the next two games are hosted by team 8, and so forth.

2.3 Financial considerations

In order to incentivize competition and fair conditions for each participant in the league, a salary cap regulates each team's spending, thereby stopping teams from simply paying more money to assemble the best possible team.

The salary cap was negotiated as part of the Collective Bargaining Agreement (CBA) [6]. Other financial considerations, such as contract conditions (e.g., there is a concept called the Designated Veteran Player Extension, in which a single player can earn up to 35% of the salary cap of a team), exceptions allowing a team to sign players under different conditions, luxury tax penalties, among many other rules, are also present in the CBA agreement.

For the purposes of this thesis, we highlight that the league wants to incentivize a leveled playing field for all teams, and it tries to achieve that by limiting the financial spending of teams. However, it is worth mentioning that teams in larger markets, e.g., New York and Los Angeles, are still able to financially attract better players due to other reasons, such as exposure to marketing opportunities and because those teams are more often nationally televised, thus increasing the recognition of the players in those teams.

2.4 Draft

The draft poses an extraordinary opportunity for franchises to improve their rosters. First-year players (also known as "rookies") enter the league with the lowest salaries and remain under contract with the franchise that drafted them for 3-4 years, depending on the position that the player was drafted. The pool of players eligible to be selected is large (between 150-200 players), and, most importantly, a franchise can draft a player that might become one of the best players in the league, who in the future could demand a large salary and therefore is more challenging to add to a roster, given financial limitations described in section 2.3.

The NBA draft occurs every year shortly after the season's final games have been played. Sixty players are chosen (thirty in the first round and thirty in the second round). There is no maximum age limit for drafted players. However, they must be 19 years old to enter the draft. This fact, defined in the latest CBA agreement [10], is also known as the "one-and-done" rule since players graduating high school frequently go to college for one year until they are old enough to enter the league.

The NBA draft lottery determines the order in which each team selects players. Initially, this involved a lottery drawing machine with several balls from each team, where the number of balls was proportional to each team's record in the

previous season. The process has been entirely digitized now and has incorporated a change for the worst-performing teams, in that the worst three teams in the entire NBA have the same probability of landing the top pick in the upcoming draft. This change was implemented to discourage teams from deliberately losing to secure the highest draft pick.

The concrete way of ordering draft picks is highlighted below (please refer to [11] for an in-depth explanation):

1. After a given season is concluded, the 14 teams that did not make the playoffs are eligible for the Draft Lottery, and assigned probabilities for being selected decrease linearly with the previous year's record. For example, the worst three teams have a probability of 14%, the fourth-worst team has a probability of 12.5%, and so forth.
2. For the first four picks, a result is drawn from the machine respecting the assigned probabilities. A team cannot be picked twice.
3. The picks 5-14 are then determined by reverse order of regular season record for the remaining teams in the lottery.
4. The picks 15-60 also follow the same logic but now involve the teams that made the playoffs in the previous season.

It is worth noting that draft picks can also be traded, and such trades can involve protections. For example, team A can trade a first-round draft pick to team B, but if the pick lands in the top 5, then the pick is not conveyed in that year but only in the following year. There is an interesting study [5] associating a financial value to each draft pick protection.

Another interesting aspect to be explored is the salaries of draft picks. Players selected with picks 1-30 in the first round are automatically signed to 4-year contracts with values determined by the league (total compensation is highest for the first pick and lowest for the last pick). Those salaries are also fully guaranteed in the first two years, while the drafting team has a player option in years 3 and 4.

Second-round picks (31-60) are considered regular free agents and can sign any contract they can negotiate, respecting the CBA rules.

2.5 Trades

Transactions between teams often happen in the NBA, involving players under contract, draft picks, rights to drafted players, and cash. Trade regulations are also described in [6] and [10]. We highlight the following interesting points:

1. Trades can happen from the start of the regular season until the NBA's trade deadline (generally at the beginning of February). There is also a moratorium period (typically at the beginning of June), where no new deals can be made (apart from minor exceptions like exercising contract options).

2. Sign-and-trades are interesting cases where a team signs an existing player to a deal and trades him to a new team immediately after that. This is interesting because a player's current team can offer better contractual terms to its players (to disincentivize players from leaving small market teams for larger market teams), including exceeding the salary cap for signing existing players. In this sense, in a situation where a player wants to switch teams, but his current team does not want to simply lose the player after his contract is finished, the team can sign him to a new contract (with improved financial terms) and then trade him to another team of his preference, while gathering new assets in the process (like draft picks).

3. Teams can also include cash in trades. However, there is a limit on the total amount that can be negotiated within a year (in 2019-20, this limit was roughly 5 million US dollars).

4. It is possible, under certain conditions, to add trade prohibitions to a contract, limiting a team's ability to trade a player. Those are not very common currently, much more common are "trade kickers", where a player's compensation sees a significant increase if he is traded.

As described above, trades are a highly complex part of a General Manager's job, given the many restrictions in place.

An interesting topic would be to model the trades as financial contracts (e.g., a swap) in that two counterparties agree to buy/sell an asset at a specific time. One could then assign a fair price to the involved players/draft picks/cash (treating them as assets) and quantitatively determine if the trade makes sense for both parties. We will discuss this topic in some detail in chapter 7.

2.6 Standings

We also briefly explain how the final standings of a given season are calculated. This has important implications, as it determines playoff match-ups and lottery positioning.

The defining quantity for determining final standings is the winning percentage (the number of wins divided by the number of losses). Teams are ordered following the winning percentage (in each conference). In the simplest case, no ties

occur and the positioning is finished.

For settling ties, different criteria are utilized depending if the tie is a two-way tie or a multi-way tie, as described in more detail in [12].

For a two-way tie, the following criteria are observed (in order) for settling the tie. If a criterion is already sufficient, then the tie is settled. Otherwise, the next criterion is examined.

1. Winning percentage on games between the teams
2. Division leader wins tie vs. non-division leader
3. Winning percentage for games played against teams from the same division
4. Winning percentage for games played against teams from the same conference
5. Winning percentage in games played against playoff teams from the same conference
6. Winning percentage in games played against playoff teams from the other conference
7. Net points scored during the entire regular season

For a multi-way tie, the following criteria are observed:

1. Division leader wins tie vs. non-division leader
2. Winning percentage on games between all teams
3. Winning percentage for games played against teams from the same division
4. Winning percentage for games played against teams from the same conference
5. Winning percentage in games played against playoff teams from the same conference
6. Winning percentage in games played against playoff teams from the other conference
7. Net points scored during the entire regular season

2.7 Free Agency

Free agency is the period when players and teams negotiate the player's upcoming contract, as his current contract is about to expire. Again, the terms of those negotiations are extensively characterized in the CBA agreement [6]. However, there is one central point we would like to highlight concerning the two types of free agency: unrestricted and restricted.

The unrestricted free agent can sign with any team he chooses. As mentioned before, usually his current team (or the team that holds his rights, also known as "Bird" rights) can offer a more lucrative contract, but that is not always sufficient to make sure a player re-signs with his current team, as other factors also play a role in the decision, such as team culture, personal connections with a different city or franchise, among others.

The restricted free agent is subject to the "Right of First Refusal", in that his current team has the right to match any offers other teams have made to employ the player. This usually is the case for first-round draft picks, and the reason for this restrictive approach is to support teams in keeping their draft picks instead of allowing players to leave in an unrestricted way after only four years. This approach has been proven very effective, as top-level draft picks usually sign a new contract after their original draft contract with the team.

Chapter 3

Methodology

In this chapter, we describe the Methodology we followed for building a decision support system for NBA front office personnel and the requirements we foresee for such a system.

3.1 Methodology and evaluation

The Methodology, on a high level, was based on the following considerations:

- The main operational aspects of the NBA had to be understood, such as the number of games, schedules, player movements (trades, signings, draft), etc. This comprises the "business logic" of our simulation tool.
- Based on the theoretical aspects of how NBA seasons are organized, we suggested a few requirements that the system should implement to be as close to reality as possible. For example, the system should be able to model games since the number of wins defines the teams that advanced to the playoffs and which team won the championship.
- Based on the requirements analysis, we started with the actual implementation of the system. Following previous experience and based on the design principles (discussed in section 3.2), we built a working prototype that could produce results.
- We then moved to the evaluation phase. This evaluation consisted of two parts: first, did the system implement the requirements, and if yes, how reasonable were the assumptions/approximations involved in each component? Second, were the results produced by the tool compatible with reality?

- Finally, depending on the results of the system, we iterated and improved any parts of the system that were not satisfying the two previous parts.

It is also worth noting that, given a set of approximations for simulating a very complex model, which is an NBA season, we conduct Monte Carlo simulations of a large number of random scenarios, where there are three main factors of randomness that affect the results: (a) the game outcomes, which is probabilistic, (b) the roster, since player movement and drafting are also probabilistic, and c) the settlement of ties when determining playoff seeding. In this sense, we are interested in "what worked well" for a given team so that the system user can learn from those "best case scenarios" and apply those optimal conditions to its team for increased chances of success.

After building a system to simulate NBA seasons, a necessary step is to provide assurances that the system can deliver reliable results. We divide the simulations into two categories:

1. Historical seasons: we simulate five past NBA seasons (2017-18 until 2021-22) and compare the actual results with the actual outcomes. We expect the same teams that either won championships or had great odds of winning to appear with higher frequency in the simulated results.

2. Future seasons: we simulate five future NBA seasons (2021-22 until 2025-26) and analyze the outcomes for a given team (we pick the New York Knicks as an example, but the analysis could have been carried out for any other team). We determine interesting KPIs (e.g., points scored by the top 5 players) that might be relevant for NBA front-office professionals when trying to build a roster.

3.2 System requirements

As motivated in section 1.2, our framework will be a decision support system that offers recommendations to NBA front office personnel based on data. We translate this motivation into the following goals:

Goal 1 Our main goal is to provide informative reports for interested parties (e.g., NBA general manager) containing descriptive information regarding a given number of NBA seasons. The report can contain information such as leading scorers, leading rebounders, evolution of draft picks per year, among other indicators.

Goal 2 A decision support system for the NBA can always be further optimized. Hence, we state as an additional goal that the framework should have the

ability to be easily extended and refactored by other people to plug-in new models or components.

In order to achieve these goals, we observe the following requirements:

Requirement 1 The system should be able to simulate one NBA season and track the final positions of each team, as well as their main statistical indicators.

Requirement 2 Within an NBA season, the system should be able to simulate regular season games and playoff games and calculate the final position of each team.

Requirement 3 Within an NBA season, the system should be able to simulate the NBA draft, where a team can draft amateur players. This simulation will then assign stats to each player (e.g., points per game) according to their draft position, and each player will be added to a team's roster.

Requirement 4 Within an NBA season, the system should be able to simulate trades involving players, observing a set of rules that guide each trade.

Requirement 5 Within an NBA season, the system should be able to simulate free agent (professional players without a currently valid contract) signings during the offseason.

Requirement 6 The system should be able to simulate a full scenario comprised of 5 NBA seasons. Each season involves an offseason (where amateur players are drafted and free agents can be signed) and a regular season (where players can be traded and games are played).

Requirement 7 The system should be able to conduct a Monte Carlo simulation of those scenarios in order to define the most-likely scenarios for each team and the optimal decisions that were taken by the front office that led to success in that specific scenario.

Requirement 8 The system should be able to produce a report after the simulation has been finished, displaying the main results of a given team and season, e.g., player X has exceeded expectations, a player drafted that year was a great addition to the team, etc.

Requirement 9 The system should be able to run a single scenario (encompassing five years) within 30 seconds or less.

We also highlight the following properties that we expect from the system:

Property 1 Modularity: the framework should be built in a way that allows for its components to be individually extended, so that new and possibly better and more complex models can be easily integrated into the tool.

Property 2 Reproducibility: given a set of initial conditions (e.g., team rosters, game schedule, etc.), we expect the system to yield reproducible results. This has the advantage of checking for errors more efficiently within a given simulation execution and simplifying maintenance.

Property 3 Performance: the system should be able to run and produce results within a reasonable time frame. Although the system's performance is closely coupled with the number of scenarios it is asked to run, we define a simulation involving 1,000 scenarios as the benchmark, which should take no more than 1 hour to complete.

Chapter 4

System Design

In this chapter, we present the different components built to satisfy the requirements previously stated in chapter 3.2. We discuss the design decisions' trade-offs and present the main challenges we faced when implementing the system.

A high level overview can be found in figure 4.1 and the different SQL tables with the corresponding relationships can be found in figure 4.2.

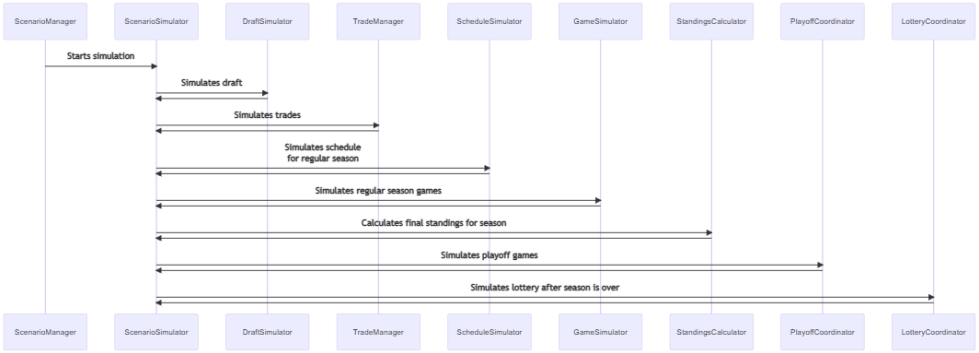

Figure 4.1: Overview of main components of the simulation tool, including the sequence of calls between components.

We decided to implement the simulation tool in Python, a prevalent language that

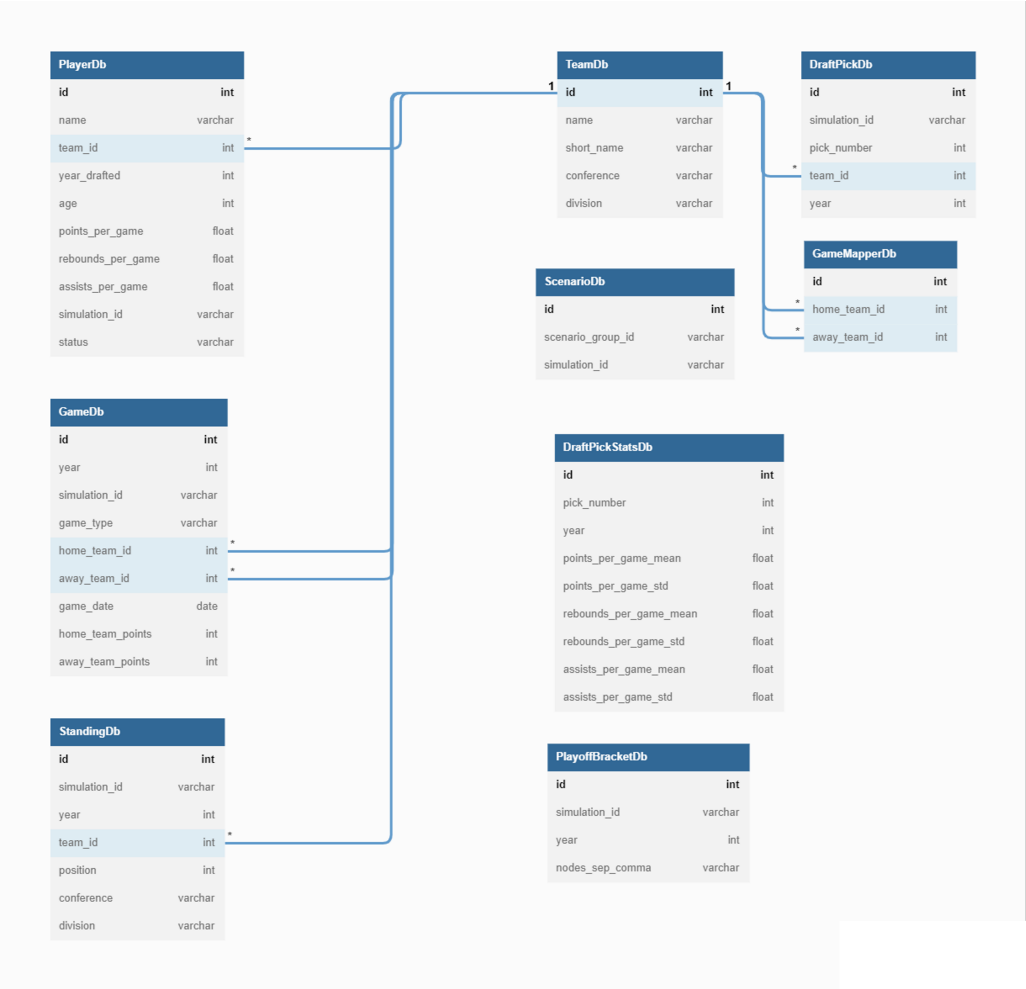

Figure 4.2: Overview of SQL tables and their corresponding relationships.

is easy to use for people wishing to continue expanding this tool. We also used PostgreSQL as a database engine for storing the simulation data.

In the following, we describe each component in detail.

4.1 Data fetching

We fetched data regarding players' statistics and game outcomes from the public NBA.com data portal using Python adapters [13]. We also retrieved sports betting odds from [14] for benchmarking our results for historical seasons with expectations from experts.

4.2 Draft Simulator

The *DraftSimulator* class is responsible for simulating the draft picks for a given season and comprises two parts, described in the subsections below.

4.2.1 Draft ordering

The draft positioning was already publicly available for the initial NBA season (2021-22). For subsequent seasons, in reality, there is a lottery system responsible for allocating teams to draft positions [10] so that the worst teams in the previous season have the highest chance of "winning" the lottery and being rewarded with the highest pick, in order to make the league increasingly more competitive.

In this thesis, we simplified this concept and, instead of implementing a lottery, we ordered the teams by their record in the previous season in reverse order, i.e., the worst performing team (independent of the conference) was rewarded with the first pick, the second worst with the second pick, etc. This does not 100% correspond to reality, as the lottery is deterministic rather than random in this case. Another critical point is that the tool does not allow for trades of draft picks, which is typically the case when players are traded.

4.2.2 Draft pick player averages

When each pick is made (ranging from picks 1-60), a new imaginary player is minted and added to the team's roster. Upon creation, this new player receives a value for his scoring, rebounding and assists averages, which are randomly drawn from a normal distribution with mean σ and standard deviation μ.

We estimated values for the mean and standard deviation by collecting data from all active players in the NBA (roughly 600 players as of the time of writing) and aggregated the data by draft pick position in order to generate a tabular structure as seen in the table 4.1 (please note that this corresponds to the structure of table *DraftPickStatsDb* as shown in figure 4.2). Finally, every time a brand new draft pick should be generated in a simulation, we drew a sample from a normal distribution centered around the mean and with the standard deviation corresponding to the draft pick number.

This process of drawing from a normal distribution contributed to the randomness of our simulations, as there is a chance that the draft pick becomes a superstar and affects the team in a very positive way. However, it becomes increasingly less likely as the draft pick number increases. That is also why there is a jargon in the NBA called "draft gems", referring to excellent players found in the later stages

			Points per game		Rebounds per game		Assists per game	
id	pick #	year	Mean	Std	Mean	Std	Mean	Std
1	1	1	14.03	5.32	3.30	2.47	5.77	3.42
2	1	2	15.73	7.43	3.24	2.41	5.77	3.74
3	1	3	14.56	9.32	3.14	2.43	5.249	4.12
4	1	4	13.46	9.37	2.74	2.55	4.97	4.42
5	1	5	10.14	10.08	2.18	2.79	3.58	4.27
6	2	1	10.69	4.83	2.38	1.89	3.61	1.06
7	2	2	10.39	7.44	2.23	1.80	3.21	2.24
8	2	3	11.90	8.63	2.30	1.77	3.73	2.44
9	2	4	10.29	9.37	1.95	2.14	3.08	2.57
10	2	5	9.82	9.23	1.67	1.49	2.88	2.81
...	...	...	...	...	...	...	...	...

Table 4.1: Mean and standard deviation values for points, rebounds and assists for each draft pick number based on data from NBA current active players.

of the NBA draft lottery. We will see the impact of drafted players in more detail in section 5.3.

4.3 Trade Manager

As discussed in more detail in section 2.5, it is challenging to simulate trades given all the rules a team has to follow, mainly regarding salary cap constraints and the fact that draft picks can and typically are involved.

In the component *TradeManager*, we simplified the trades by considering "talent index" as the only parameter that validates a trade, so that trades must be comprised of similar talent index values on both sides to be valid. In practice, this talent index was defined as the sum of points per game in a given season of all players from a given team involved in the trade. We also allowed a small tolerance (configurable on the component but initially set to 3 points per game) for allowing a large universe of possible trades.

The steps involved in completing a trade are defined as follows:

1. The 30 teams are organized in two sets, A and B.
2. Each pair (teamA and teamB from each set) is matched.
3. A random number r ($0 <= r <= 1$) is drawn to represent the probability

of a trade being executed. We chose that number to be 0.5 to allow for a reasonable number of trades within a season. If $r > 0.5$, the trade between teamA and teamB is neglected and a new pair is processed.

4. Assuming the trade for teamA and teamB continues, we select all possible trade combinations involving all players from teams A and B. An example is given below:

   (a) Team A has three players: player A1 with 10 points per game, A2 with 5 points per game and A3 with 5 points per game.

   (b) Team B has players with same averages: $B1 == A1$, $B2 == A2$, $B3 == A3$.

   (c) Now, we produce the following combinations, assuming a tolerance of 0 points per game:

$$\{(A1), (B1)\}, \{(A2, A3), (B1)\}, \{(A1), (B2, B3)\}, \{(A2, A3), (B2, B3)\}$$

5. We shuffle the trade combinations and pick one trade to be executed, as we limit the number of trades to 1 per team pair. In this sense, no more than one trade is executed by each team per season. Another simplification we do is only to allow trades between 2 teams and to have the validity of the trade only for one season for simplicity reasons.

With these simplifications, we add another degree of randomness to the simulations, allowing players to be traded. We also discuss the influence of traded players on section 5.3.

4.4 Schedule Simulator

The *ScheduleSimulator* component is simply responsible for allocating games between teams on a randomly selected date on the NBA calendar.

As discussed in more detail in section 2.2, there are plenty of rules to be followed when determining the schedule of an NBA season. We simplified these rules so that we did not consider "back-to-backs" (games played on consecutive nights) nor "road trips" (where specific teams travel to distant states and play a series of games against teams located within a short distance of each other, like the teams based in California). We simply allocated games randomly respecting the NBA calendar.

4.5 Game simulator

We considered two approaches for simulating games, described in subsections 4.5.1 and 4.5.2. We ultimately decided to employ the team-focused approach, which increased accuracy when predicting game outcomes.

4.5.1 Player-focused simulation

Our first attempt to simulate a game was to loop through the players of each team and simulate how many points each player scored. By adding up the points scored by each player, we would have a final score (in the case of a tie, we started the process again). This "tiebreaking" mechanism has been proven stable throughout our studies.

To simulate how many points each player scored in a given game, we had to draw a random number from a distribution that resembles the scoring output of the said player. As detailed in the appendix, a player's points scored per game can be approximated by a Gaussian distribution, where the mean is the average of points scored per game in a given season and the standard deviation is a linear function of the mean (the coefficient and the intercept are determined via a linear regression fit).

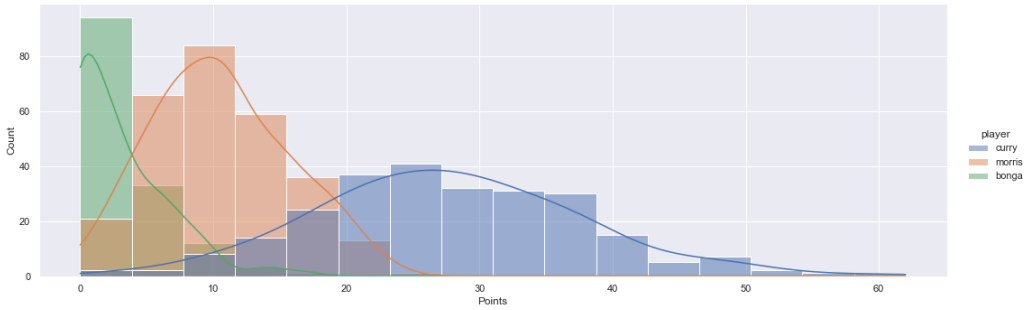

Figure 4.3: Stephen Curry's, Monte Morris's' and Isaac Bonga's scoring totals per game in the 2021-22 NBA season.

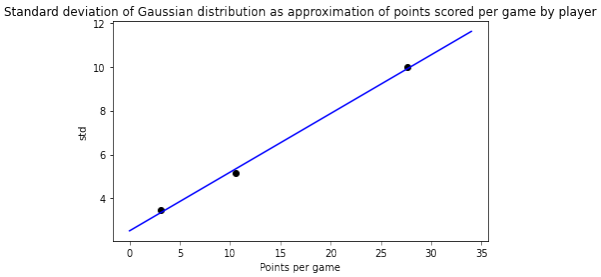

Figure 4.4: Linear regression as approximation of standard deviation in player's scoring output.

In figure 4.3, we present a histogram of the scoring outputs of Stephen Curry, one of the highest-scoring players in the NBA, as well as two more players (Monte Morris, Isaac Bonga) for visually demonstrating that the "points per game" variable can be approximated by a Gaussian function (further proof can be found in the appendix). In figure 4.4, we display the linear regression function used for determining the standard deviation parameter as a function of the scoring average.

Therefore, by using each player's mean and standard deviation, we can draw a random number for each player, following an individual distribution, and use that as points scored in a given game. Finally, we arrive at the game's final score by adding those scoring outputs for each team.

This worked fine in practice and allowed for a fast prediction of the game outcome, however during our benchmark tests, we realized that the predictions did not correspond well with the expected outputs, so we decided to abandon this simplified approach and adopt a more team-centered approach for game simulation, as detailed in the following subsection.

4.5.2 Team-focused simulation

As seen in subsection 4.5.1, we required a better way of predicting the outcome of games than simply summing up the number of points each player scored.

The task at hand was a prevalent one, i.e., given a set of features describing an object, assign a single category to said object. The classes were 0 (the away team won) or 1 (the home team won). While there are many possible algorithms for performing a classification task (as described in more detail in [15]), given our previous experience, we decided to compare two candidates: Machine Learning classifiers (linear classifier, decision tree classifier and random forests classifier) and Deep Learning classifier [7].

The basic idea of all classifiers was the same. Given the set of features, we want a model that can assign a category to a game (either 0 or 1) with a given probability. We select the class whose probability is highest and assign the game outcome to that category. Finally, we pick the model with the highest accuracy, defined as the count of games it classified correctly divided by the total number of games examined.

Although classification is a widespread and well-understood problem, game outcome prediction remains a very complex task, as it depends on many factors: location (home or away), players scoring-, rebounding- and assists statistics, injuries, timeouts, and many more. Our goal was to create a model accurate enough to predict games using the features we already had available after fetching public data from the NBA API [13].

We decided to employ a model that used 20 features:

1. Points per game averages for the current season of the top 5 scorers of the home team ($home\_scorer1$,$home\_scorer2$, ...)
2. Points per game averages for the current season of the top 5 scorers of away team ($away\_scorer1$,$away\_scorer2$, ...)
3. Rebounds per game averages for the current season of the top 5 rebounders of the home team ($home\_rebounder1$,$home\_rebounder2$, ...)
4. Rebounds per game averages for the current season of the top 5 rebounders of the away team ($away\_rebounder1$,$away\_rebounder2$, ...)

Having defined these factors, we assembled a dataset (summarized in table 4.2) that was used for predicting game outcomes based on said factors. We used the same dataset for all training routines, including an 80-20 train-test split using a fixed seed for reproducibility.

home_won	home_scorer_N	away_scorer_N	home_rebounder_N	away_rebounder_N
0	24.9512	19.878	7.42683	7.26829

Table 4.2: Dataset used for training the candidates for game prediction.

It is out of the scope of this thesis to discuss in detail the inner workings of the trained models, and we refer the reader to [7] and [15] for more detailed information.

In conclusion, we incorporated the model described above in the component *GameSimulator* so that each time a game between teams *Home* and *Away* was played, we

fetched the top 5 scorers and rebounders of each team and fed that to the algorithm, which returned the probabilities of each team winning the game. A random number was drawn based on those probabilities to determine which team won. We did that to increase randomness in our simulation since using the predicted class by the network (i.e., "home won" or "away won") would have yielded the same deterministic result every time the same two teams met. We wanted the outcome to be probabilistic, as it is in reality, i.e., a much better team has a higher probability of winning, but the worse team still has a tiny chance of succeeding.

In the upcoming subsections, we describe our reasoning for determining the engine powering the *Game Simulator* class. We explore the complexity of different options, the corresponding accuracy and the final chosen algorithm.

4.5.2.1 Machine Learning approach

We describe in this subsection our procedure for selecting the best Machine Learning classifier for our task. Detailed descriptions of how each classifier works can be found elsewhere [15], hence we focus on the optimization mechanism and the results we obtained.

We used the package Optuna [2] to find the best model and, within the best model, the optimal hyperparameters. Figures 4.5 and 4.6 illustrate how the optimization took place and how random forests yielded the highest accuracy in the test dataset.

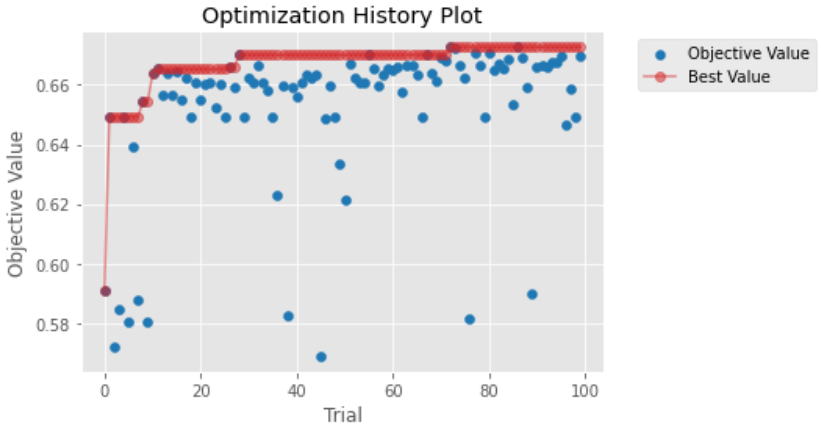

Figure 4.5: Optimization history across one study from Optuna.

As we can see from figure 4.6, a random forest classifier yielded the top accuracy ($\approx$ 67%) in the test dataset. We also extracted the feature importance of this classifier, displayed in figure 4.7. We learn from this figure that the five most

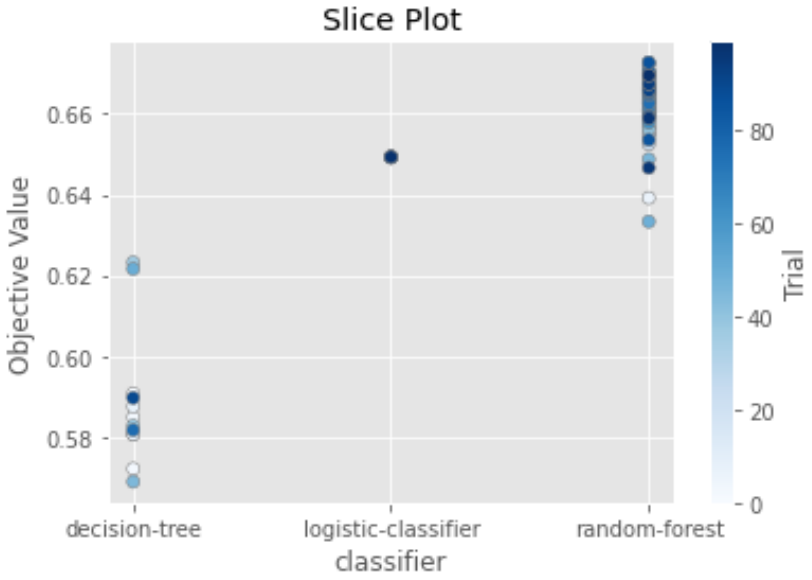

Figure 4.6: Comparison between accuracy on test datasets yielded by different Machine Learning models.

important features of the model are all scoring outputs, both from the home and the visiting team.

4.5.2.2 Deep Learning approach

In this subsection, we detail our Deep Learning approach and the accuracy obtained in the test dataset.

For this classification task, we used the fastai [7] standard Tabular model, which contained a linear model of roughly 24,000 parameters composed of normalizations, dropouts and filter components. In listing 4.1 we present a summary of the tabular model.

Code Listing 4.1: Tabular model used for predicting games

```
TabularModel (Input shape: 64 x 0)
==========================================================================
Layer (type)              Output Shape              Param #    Trainable
==========================================================================
                          64 x 20
BatchNorm1d                                         40         True
--------------------------------------------------------------------------
                          64 x 200
```

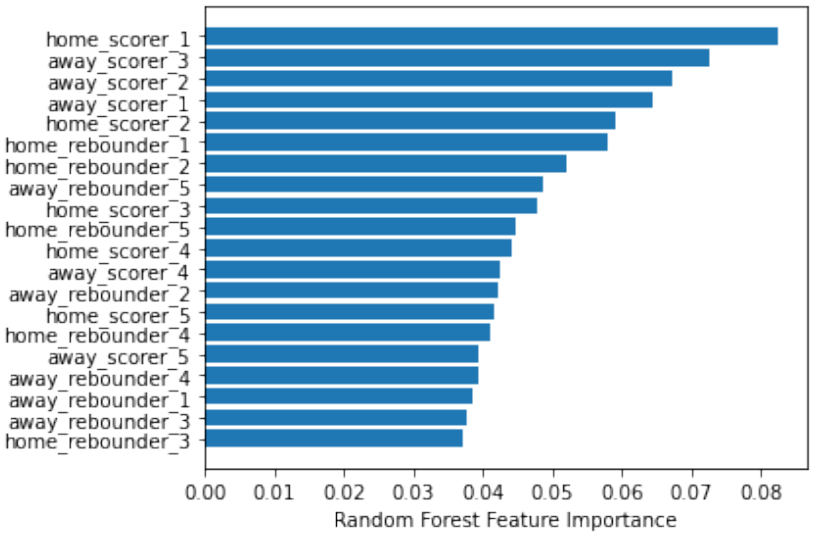

Figure 4.7: Feature importance for the best performing Random Forests classifier after hyperparameter tuning.

```
Linear                                          4000          True
ReLU
BatchNorm1d                                     400           True
----------------------------------------------------------------
                        64 x 100
Linear                                          20000         True
ReLU
BatchNorm1d                                     200           True
----------------------------------------------------------------
                        64 x 2
Linear                                          202           True
----------------------------------------------------------------

Total params: 24,842
Total trainable params: 24,842
Total non-trainable params: 0

Optimizer used: <function Adam at 0x00000236FB880310>
Loss function: FlattenedLoss of CrossEntropyLoss()
```

Using the standard model without further optimizations, we obtain an accuracy of $\approx 64\%$ after 5 epochs, as shown in table 4.3.

epoch	train loss	valid loss	accuracy
0	0.688989	0.653900	0.603035
1	0.643882	0.651571	0.633387
2	0.630948	0.644491	0.619808
3	0.610332	0.640782	0.630192
4	0.596930	0.639657	0.637380

Table 4.3: Information regarding model training without hyperparameter optimization.

Again using Optuna, we performed a hyperparameter optimization (varying several parameters, such as the number of layers, dropout ratios, etc.) and arrived at the best accuracy of $\approx 65.5\%$.

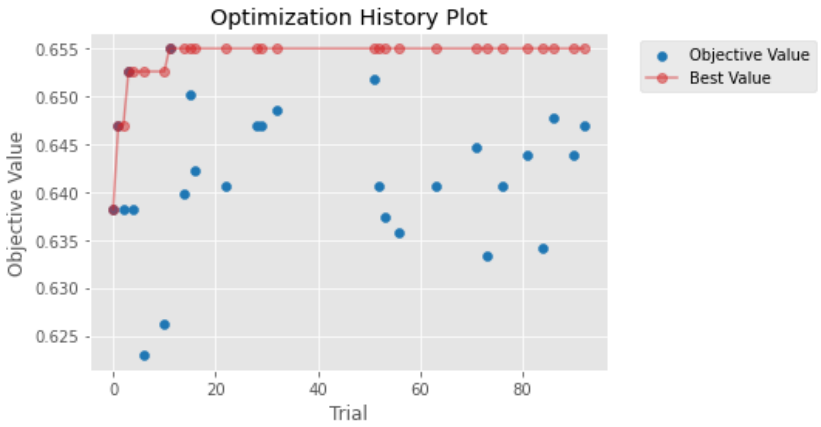

Figure 4.8: Optimization history across one study from Optuna.

4.5.3 Discussion

This subsection explains our thought process when ultimately deciding which algorithm we should choose as the classifier.

Comparing the best models we obtained for each variant (Machine Learning vs. Deep Learning), we concluded that the Random Forests classifier (accuracy $\approx 67\%$) outperformed the Deep Learning classifier (accuracy $\approx 65\%$). We argue that, although the Deep Learning classifier was more powerful and had a greater capacity to "learn" from the data (given its almost 24,000 parameters), our dataset had only 20 features to learn from, thus hindering the learning capacity of the neural network and therefore favoring a simpler model, like a Random Forests classifier.

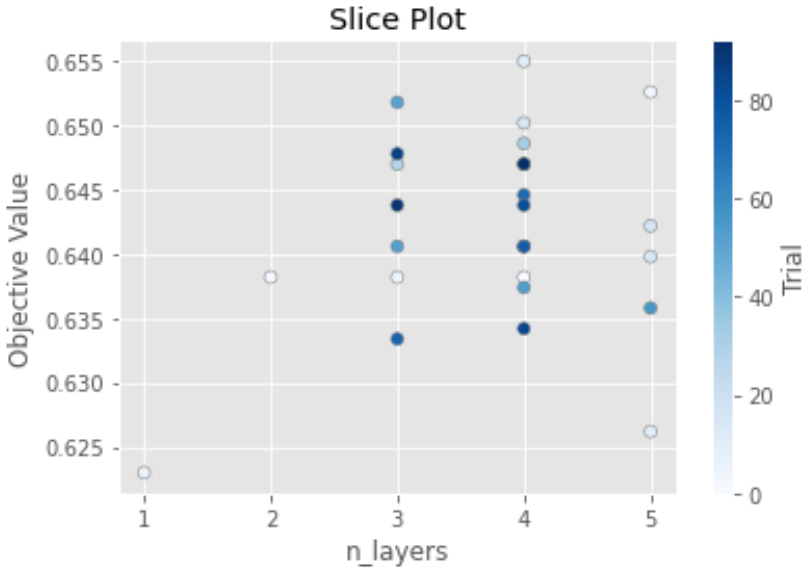

Figure 4.9: Comparison between accuracy on test datasets yielded by a neural network having different number of layers.

Another point that served as a basis for our decision was the complexity factor. Random forests are less complex and more straightforward than deep neural networks, where the weights and patterns learned can sometimes be tough to explain. Therefore we leaned towards the simpler model.

In conclusion, we used a Random Forests classifier for the game outcome prediction.

4.6 Standings Calculator

The component *StandingsCalculator* determines the final positions of each team after all regular season games have been played.

As described in more detail in 2.6, the process for determining the final standings is rather complex. It involves not only the record of each team but also the division (for the division champions) and the direct match-up outcomes between tied teams, among other factors.

We decided to proceed as follows when determining the final standings of a given season:

1. For each conference, we rank all teams according to the number of wins.

2. If two teams have the same number of wins, we order the tied teams randomly.

Although this ordering process is not ideal, it allows the simulation toolbox to solve ties in a simple and fast way.

4.7 Playoff Coordinator

The component *PlayoffCoordinator* is responsible for simulating the entire playoffs. Thus it contains an instance of the *GameSimulator*.

The *PlayoffCoordinator* itself is purely deterministic, as it simply reads the standings created by the *StandingsCalculator* and creates the corresponding match-ups between teams (e.g., in the first round, there are eight teams in each conference that play a best-of-seven series, as described in section 2.2).

After creating the match-ups, the *PlayoffCoordinator* calls the game simulator, which simulates each game of the playoffs, equivalently to how it simulates games during the regular season. Finally, after simulating the finals, the season's final standings are written in the database, following a binary tree format.

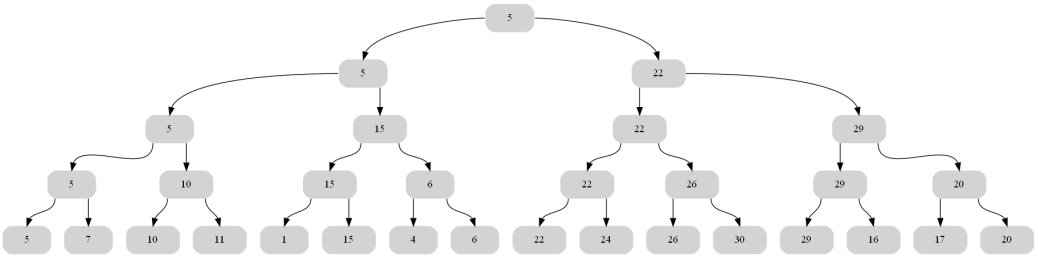

Figure 4.10: Example of how final standings are stored in the database.

This format allows for more straightforward future analysis, as the match-ups are clearly defined and the winner of each match-up keeps climbing the tree.

4.8 Lottery Coordinator

The component *LotteryCoordinator* determines the draft ordering of the upcoming draft, as explained in section 2.4.

In theory, the main goal of the NBA draft is to keep the league as competitive as possible so that the worst teams in a given season receive the best incoming

players in the subsequent season. This is however not always the case, mainly because of poor decisions from NBA front office personnel (not selecting the best player for a given team). Moreover, since the draft is probabilistic, good teams also have a chance at obtaining a high draft pick, which could hurt the overall talent balance across all teams.

We kept the complexity at a minimum when determining the order of draft picks. The component reads the standings of the recently finished season and defines the draft order by reversing the order of each team according to the standings from the previous season. For example: if team A has finished the previous season in the last position (i.e., it was the worst team in the league), then it would have the first pick of the first- and second rounds. We also assume that each team has two picks in each year's draft, which is not always the case since teams can negotiate draft picks.

Chapter 5

Evaluation of system and reflection

In this chapter, we will assess the veracity of the scenarios produced by simulating five historical NBA seasons (2017-18 to 2021-22) as well as five present- and future NBA seasons (2021-22 to 2025-26). It is worth noting that the season 2021-22 serves as a division for our simulations, as it is included both in the historical and future sets. Finally, we also reflect upon the work conducted in this thesis and how it has fulfilled the previously stated requirements.

5.1 Test of game simulations

We start by highlighting the test for the *GameSimulator* class. An excerpt of the test can be found in listing 5.1.

Code Listing 5.1: Excerpt from *game_simulator_test*, where different match-ups are simulated many times and the results are manually inspected.

```
@pytest.mark.parametrize("team_home_short_name,
                                team_away_short_name", [("BOS",
                                "OKC"), ("OKC", "BOS"),
("DET", "PHX"), ("NYK", "GSW")])
def test_game_combinations(team_home_short_name,
                                team_away_short_name,
                                game_simulator,
                                mock_simulation_id):
    """
    Boston is much better than OKC, so we write a test to
                                     determine how often BOS
                                     beats OKC.
    We expect 75 x 25 odds roughly.
    """
```

```
    run_game(team_home_short_name, team_away_short_name,
                                       game_simulator,
                                       mock_simulation_id)

def run_game(team_home_short_name, team_away_short_name,
                                   game_simulator,
                                   mock_simulation_id):
    players: dict[int, PlayerDb] = {p.id: p for p in
                                       game_simulator.db_handler.
                                       get_players_for_season(
                                       mock_simulation_id, 2021)}
    teams_dict = game_simulator.db_handler.
                                       get_teams_by_team_short_name
                                       ()
    team_home = teams_dict[team_home_short_name]
    team_away = teams_dict[team_away_short_name]
    home_wins = 0
    away_wins = 0
    n_simulations = 100

    for simulation_idx in range(n_simulations):
        mock_game_db = GameFactory() # omitted for brevity
        game_simulator.simulate_using_dl([mock_game_db], list(
                                           players.values()))
        home_win = 0 if mock_game_db.away_team_points >
                                           mock_game_db.
                                           home_team_points else 1
        away_win = 1 if mock_game_db.away_team_points >
                                           mock_game_db.
                                           home_team_points else 0
        home_wins += home_win
        away_wins += away_win

        # We expect Boston to win most games
    print(f'{team_home_short_name} won {home_wins} - {home_wins
                                       / n_simulations} times,'
          f' {team_away_short_name} won {away_wins} {away_wins /
                                           n_simulations} times'
                                         )
```

In the test, we run 100 simulations for each match-up combination and manually examine the results. Each match-up involves teams and rosters from the 2021-22 season and always involves an excellent team (Boston, Phoenix or Golden State) against one of the worst teams (New York, Detroit, Oklahoma City). By examining the results of the test (displayed in table 5.1 and comparing them with another game outcome predictor [17], we can validate that the game simulator can predict outcomes with reasonable accuracy. This is an excellent indicator of the

validity of the observations presented later in this chapter.

Home team	Away team	Win percentages (Test)	Win percentages (Benchmark)
Boston	Oklahoma City	81% - 19%	92% - 8%
Oklahoma City	Boston	32% - 68%	11% - 89%
Detroit	Phoenix	29% - 71%	6% - 94%
New York	Golden State	32% - 68%	28% - 72%

Table 5.1: Summary of game simulator predictions during tests vs. benchmark values for given match-up.

5.2 Historical simulations

For our assessment, we compare our simulation results against real sports betting data sets [14], which contain the odds when betting that a given team wins the championship in a given season. For example, odds equal to $+200$ mean that when someone wagers 100 dollars, a payout of 200 dollars is achieved if the bet is successful. Equivalently, odds equal to -200 mean that a wager of 200 dollars yields only 100 dollars in the case of a positive outcome.

Furthermore, American odds can be seen as probabilities, i.e., the smaller the payout, the higher the probability of the outcome. We follow the mechanism described in [8] for calculating the associated probabilities.

Team	Year	Odds	probability
Brooklyn Nets	2021	230	0.303
Los Angeles Lakers	2021	425	0.190
Milwaukee Bucks	2021	850	0.105
Golden State Warriors	2021	1100	0.083
Utah Jazz	2021	1400	0.066

Table 5.2: Excerpt of sports betting dataset with betting odds and corresponding probabilities for each team and year.

The first step is to produce benchmark results. With the probabilities displayed in table 5.2, we run 100 simulations of 5 NBA seasons and, during each iteration, we sample the champion for each season using the associated probabilities. This gives us a rough understanding of which teams we expect to win the most championships in the five historical seasons.

The second step involves producing results with our simulation tool. We simulate

5 NBA seasons (2017-18 to 2021-22) and count the number of championships each team obtains.

The final step involves comparing the actual results from the first step with the simulated results from the second step.

We perform a qualitative comparison of the number of championships collected by each team. We see in figure 5.1 and on tables 5.2 and 5.4 that, although the probabilities of a given team winning the championship vary between the two approaches for simulating the results (sports betting data vs. simulation tool), when we compare the eight teams that won the most championships in the given period (as seen in tables 5.2 and 5.4), we realize that 4 out of 8 teams are present in both tables. We conclude that, even though there is a discrepancy regarding the probabilities of specific teams (especially the ones with the highest probability), the overall trend, i.e., the list of teams with the highest chances of winning the championship in any given season, is very similar in both approaches.

We also calculate an "error" between the two approaches. By dividing the number of championships accumulated by each team over a simulation by the total number of simulations, we determine the probability of a team winning a single championship. We then define our error as the absolute difference between the probability associated with the betting data and the probability associated with the simulated data. We arrive at the value of 4%.

In this sense, we argue that the model can produce valid results for historical seasons, given our qualitative arguments presented above and the fact that sports betting odds are generally regarded as good indicators of outcome prediction.

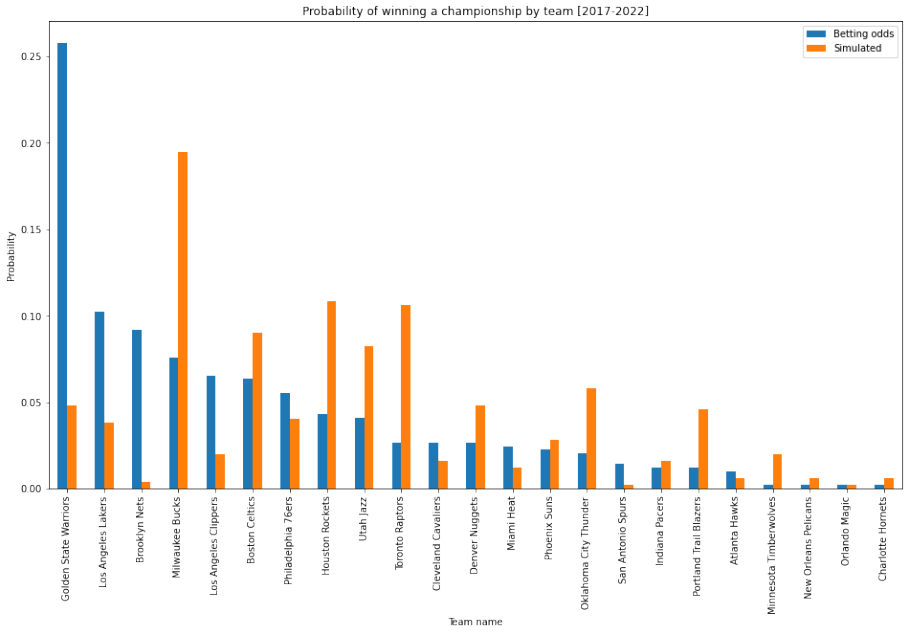

Figure 5.1: Comparison between the probability of a team winning a championship between 2017 and 2022 using betting data vs. simulated data.

0	Golden State Warriors	126
1	Los Angeles Lakers	50
2	Brooklyn Nets	45
3	Milwaukee Bucks	37
4	Los Angeles Clippers	32
5	Boston Celtics	31
6	Philadelphia 76ers	27
7	Houston Rockets	21

Table 5.3: Sports betting data

0	Milwaukee Bucks	97
1	Houston Rockets	54
2	Toronto Raptors	53
3	Boston Celtics	45
4	Utah Jazz	41
5	Oklahoma City Thunder	29
6	Golden State Warriors	24
7	Denver Nuggets	24

Table 5.4: Simulation data

5.3 Future simulations

In this section, we do a Monte Carlo simulation of 1000 scenarios, each comprised of 5 NBA seasons, and select one particular team (New York Knicks) for an in-depth analysis, to pinpoint exciting findings made possible by the tool that could be interesting to NBA general managers to research further.

An overview of the champions produced by the simulation can be seen in table 5.5.

Champion	# championships
Phoenix Suns	1316
Memphis Grizzlies	690
Boston Celtics	480
Golden State Warriors	421
Miami Heat	317
...	...
New York Knicks	10

Table 5.5: Summary of championships won by each NBA team during the scenario simulation. We highlight the New York Knicks, which won only ten times and will be the subject of further analysis.

We highlight three interesting results from analyzing data from that particular team.

The first result concerns the scoring average of the top 5 scoring players, as shown in figure 5.2. We reiterate that the scoring average is a significant factor for determining championship winning odds, as championship seasons typically have players with higher scoring averages than non-championship seasons.

From figure 5.2, it is possible to conclude that increases in scoring output play a significant role in the championship occurrences. In other words, during championship seasons, we observe that the top scorers produce more points than for non-championship seasons. This is true not only for the highest-scoring player but also for all other high-scoring players (scorers 2, 3, 4 and 5).

It is also worth mentioning that (as discussed in more detail in subsection 4.5.3, where we explain feature importance) the influence of the scoring output in general decreases with player index, whereas the home team's chances are most greatly defined by its top-2 scorers and the away team's chances are most strongly affected by its top-3 scorers.

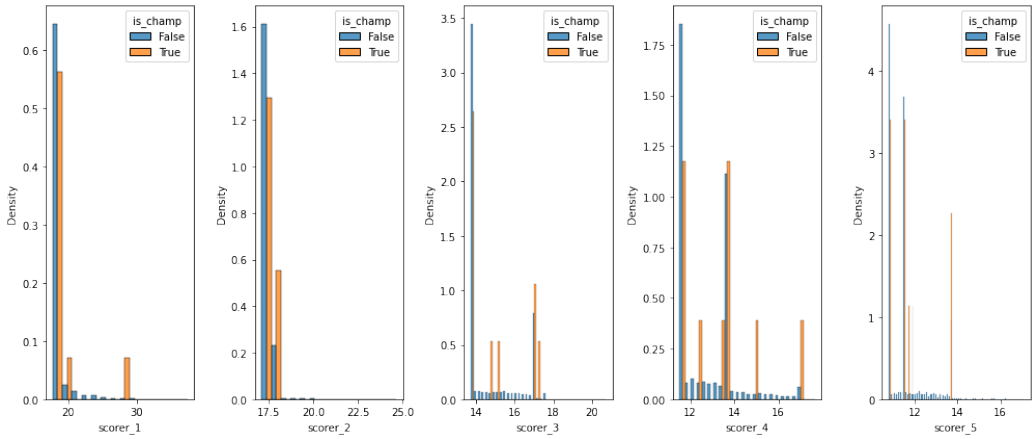

Figure 5.2: Distribution of points per game outputs of top 5 scoring players of championship- vs. non-championship teams.

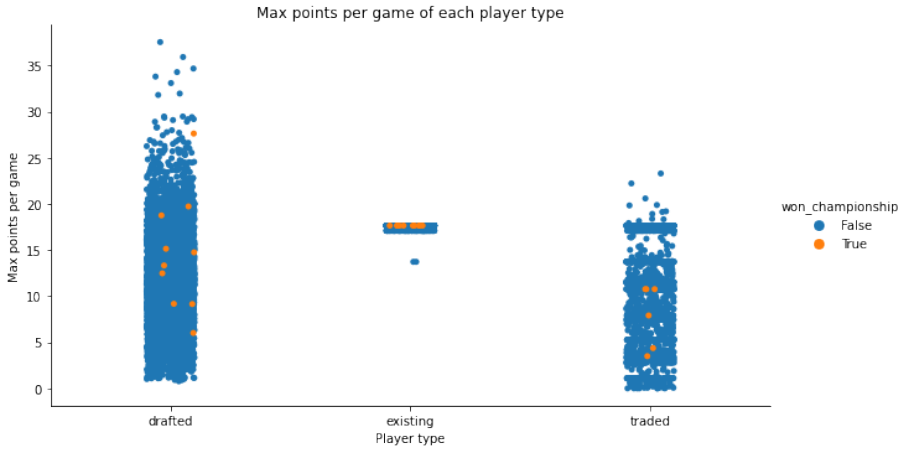

Figure 5.3: Points per game outputs by each player type (drafted by the team, traded or existent previously to the start of the simulation). Blue denotes non-championship seasons whereas orange denotes championship seasons.

The second result concerns each player type's maximum points per game, as shown in figure 5.3. We summarize our findings below:

1. Again, we observe a strong influence of points per game over the likelihood of a team winning the championship. For all player types, championship seasons appear more likely when max points per game are higher.

2. When draft picks perform particularly well scoring-wise, it seems that championship seasons are more likely.

3. Scoring outputs by traded players do not seem too predictive for championship odds.

The third result concerns each player type's maximum rebounds per game, as shown in figure 5.4. We summarize our findings below:

1. We observe a lower correlation of maximum rebounds per game than the maximum points per game over the likelihood of championship, as the championship data points seem more distributed across the y-axis for both drafted- and traded players.

2. However, for existing players, we see that all championship seasons involved high maximum rebounding numbers from existing players. That suggests that a team should always have strong rebounders, most often from existing players on the roster.

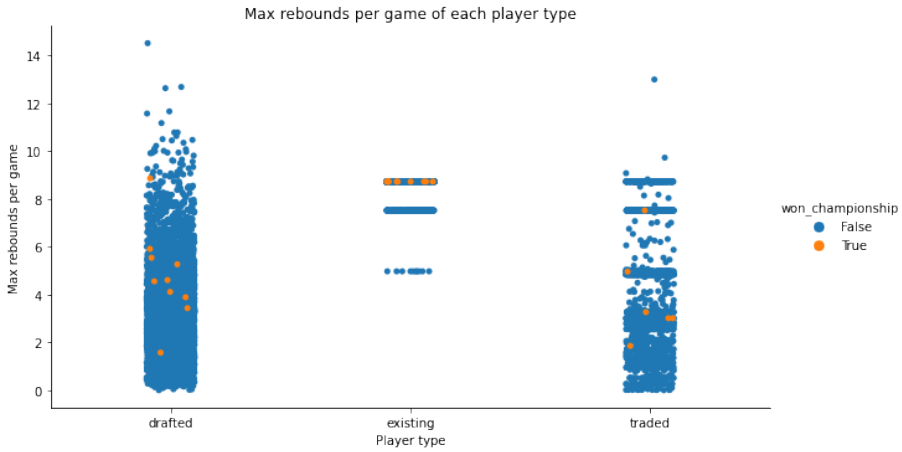

Figure 5.4: Rebounds per game outputs by each player type (drafted by the team, traded or existent previously to the start of the simulation). Blue denotes non-championship seasons, whereas orange denotes championship seasons.

5.4 Reflections

In this section, we revisit the properties and requirements stated in section 3.2 and discuss our reasoning when devising a solution.

We start with the properties initially devised for the system:

Modularity - We used Object-Oriented programming to organize our code in an easily understandable manner and to ensure that each class or object was responsible for one task as part of the simulation tool. For example, one of the most

important classes is the *GameSimulator*. An excerpt of its implementation is displayed in listing 5.2.

Code Listing 5.2: Excerpt from implementation of GameSimulator.

```
class GameSimulator:
    db_handler: DBHandler
    simulation_id: str
    model_name: str = "model_gs.pt"
    logger = Logger()

    def __post_init__(self):
        self._load_learner()

    def simulate_game_type(self, simulation_id, year, game_type)
                                        :
        games = self.db_handler.get_games_by_game_type(self.
                                            simulation_id, game_type
                                            , year)
        players = self.db_handler.get_players_for_season(
                                            simulation_id, year)
        self.simulate_using_dl(games, players)
        self.db_handler.write_entities(games)
        self.logger.logger.debug("Finished simulate game type")
```

Modularity is a major concern, as we want to keep each component as decoupled as possible. In this example, the *GameSimulator* is only coupled with the *DBHandler*, responsible for database interactions. Apart from that, it simply calculates the outcomes of games and writes the updated result in the *games* table.

We wanted our components written in a modular way also for extensibility reasons. It would be presumptuous to assume that this simulation tool can simulate games better than every other statistical tool on the internet. Instead, we focus on writing a tool that serves as an infrastructure layer, where other interested people can extend and easily replace components with improved versions and examine how the results change. Again staying in the *GameSimulator* example, it suffices to reimplement the function *simulate_game_type* (for example, using a different algorithm that yields better accuracy when predicting games) in order to have all games simulated using an improved logic. This is the benefit of modularity and extensibility.

Reproducibility is also required for comparing simulations using different implementations. If we consider a set of initial conditions (players of each team, location of the game, etc.), we expect the system to yield the same reproducible results. The primary step to achieve this was to initialize a seed for all the random numbers drawn from the simulation and ensure that the seed was kept every time.

Following the standards for our implementation [9], we decided to implement a random number generator class as a Singleton, as seen in the listing 5.3.

Code Listing 5.3: Implementation of the RandomNumberGenerator following the Singleton design pattern.

```
class SingletonMeta(type):
    _instances = {}

    def __call__(cls, *args, **kwargs):
        if cls not in cls._instances:
            instance = super().__call__(*args, **kwargs)
            cls._instances[cls] = instance
        return cls._instances[cls]

class RandomNumberGenerator(metaclass=SingletonMeta):
    def __init__(self):
        self.generator: np.random.Generator = np.random.
                                              default_rng(12345)
```

The Singleton pattern guaranteed that only our custom random number generator was used across the entire application, thus assuring the reproducibility of results.

Performance was a concern given the large number of planned Monte Carlo simulations. We wanted to avoid the complexity of having a long-running program that needs to run over many hours. Table 5.6 summarizes the performance of the simulation tool for multiple configurations.

# simulations	Years simulated	Time elapsed (s)
1	1	16.84
1	5	108.5
5	1	82.68
5	5	453.4

Table 5.6: Comparison of time elapsed for different simulation parameters.

As seen in table 5.6, it takes 16s for each season to be simulated on average. In our largest case (of 1,000 scenarios and five years), it took us approximately 20h to finish the simulation. Although not optimal, this was still manageable and much better than in the beginning, as the time per season simulated was 5x longer.

We made three major design decisions to improve the time elapsed per season:

1. **Database** - We started using SQLite [18] as a relational database. Our application is read- and write-heavy, as the entities being retrieved (e.g., games) are updated, apart from new standings being written. Therefore we changed

implementations and started using PostgreSQL [16] to improve read- and write times. Although this move has improved the time elapsed per season by a tiny amount, it was not a decisive factor. It is also worth noting that all the interactions with the database are located inside the *DBHandler* class, where a single line defines the connection with the database. Therefore, it would be straightforward to switch to a different database engine.

2. **Paralellization** Another design decision we investigated was whether it was helpful to parallelize the application. It makes sense in theory to start one thread per simulation (i.e., if we were running a Monte Carlo simulation comprised of 1000 scenarios and five seasons, we would start 1000 threads). However, while testing this approach, we realized that the overhead of having one database connection per thread (and the increased complexity of pooling connections) was too heavy and decided to abandon these efforts, as the total simulation time increased when threads were part of the implementation.

3. **Predictions of algorithm** The difference maker for our simulations involved the way we generated the predictions using the model as part of the *GameSimulator*. In the beginning, we were running a loop of all games and predicting the outcome of each game separately, calculated when we passed in features in a tabular format. Much better performance could be achieved when we generated predictions for all games at once using a vectorized approach, which allowed us to improve simulation times significantly.

Furthermore, we briefly discuss the requirements stated in section 3.2 and how we fulfilled them.

(a) Requirement 1 stated that final positions and statistical indicators should be tracked. This was achieved by integrating a PostgreSQL database into the tool so that all relevant metrics (players, games, standings) could be tracked. It is worth mentioning that a unique *simulation_id* was assigned to each simulation (comprised of a number N of years, $1 <= N <= 5$, so that data from each simulation could be easily filtered.

(b) Requirement 2 states that the tool should be able to simulate games. This was discussed before when explaining the implementation of the *GameSimulator*, and the main idea was that this component should be able to predict game outcomes reasonably accurately. It is also worth noting that a test (described in chapter 5.1) was written precisely for this component, highlighting that this was one of the significant parts of the software. Although the predictions can constantly be improved, we believe that approximately 67% of accuracy was satisfactory, and future implementations could improve the

prediction logic and be integrated seamlessly.

(c) Requirement 3 concerns the draft mechanism. This functionality is present on the *DraftSimulator*, where we implemented statistical modeling of the relevant stats (points-, rebounds- and assists per game) for the N years of the simulation, and in the *LotteryCoordinator*, where we determine at which position each team will be drafting, according to the standings of the previous season (or following a hardcoded sequence if it was the first year of the simulation).

(d) Requirement 4 concerns the simulation of trades. This functionality is present on the *TradeSimulator*, where we allowed players who accounted for similar totals of scoring average to be traded between 2 teams within one particular season.

(e) Requirement 5 concerns free-agent signings. As discussed in section 2.7, players without a contract must abide by the guidelines for signing a new contract, either with the previous team or with a new team. The main factor for this simulation would be the contract of each player (duration, amount, guaranteed payments, etc.). We did not implement this functionality due to time constraints, but a practical implementation would involve the following steps:

  (i) The contract data from all players is fetched [1] and stored in the database in a new table (*Contracts*). Contracts for incoming draft picks are defined in [10] and should also be stored in the table as templates.

  (ii) The table *Contracts* has, among other columns, a *player_id* field, which acts as a foreign key to the table *Players*.

  (iii) The rules for free agent signings, defined earlier in section 2.7, would have to be translated into code and added to the simulation routine.

(f) Requirement 6 concerns the simulation routine and states that the sequence of execution should involve N seasons, where $1 <= N <= 5$ and an offseason (including the Draft and Free Agency) between two consecutive seasons. This requirement was implemented in the *ScenarioSimulator*, as can be seen in listing 5.4.

Code Listing 5.4: Excerpt from ScenarioSimulator showing the sequence of events occurring during a simulation comprised of N years.

```
class ScenarioSimulator:
    db_handler: DBHandler
    start_year: int
    is_benchmark: bool = False
```

```
    logger = Logger()

    def __post_init__(self):
        self.simulation_id = str(uuid.uuid4())

    def simulate_scenario(self, n_years):
        pm = PlayerManager(self.db_handler, self.
                                        simulation_id)
        pm.duplicate_entities_without_sim_id()
        dpm = DraftPickManager(self.db_handler, self.
                                        simulation_id)
        dpm.duplicate_entities_without_sim_id()
        if not self.is_benchmark:
            pe = PlayerExpander(self.db_handler, self.
                                            simulation_id,
                                            self.start_year
                                            , n_years)
            pe.expand_entities()

        for year in range(n_years):
            year_to_simulate = self.start_year + year
            self.simulate_year(year_to_simulate)

    def simulate_year(self, year: int):
        if not self.is_benchmark:
            # Start offseason
            draft_simulator = DraftSimulator(self.db_handler
                                            , year, self.
                                            simulation_id)
            draft_simulator.simulate_draft()
            TradeManager(self.db_handler, self.simulation_id
                                            , year).
                                            execute_trades
                                            ()

        # Start regular season
        ScheduleSimulator(self.db_handler, year,
                                        simulation_id=self.
                                        simulation_id).
                                        generate_schedule()
        game_simulator = GameSimulator(self.db_handler, self
                                        .simulation_id)
        game_simulator.simulate_game_type(self.simulation_id
                                        , year, GameTypes.
                                        REGULAR_SEASON)

        # Start playoffs
        sc = StandingsCalculator(self.db_handler, self.
                                        simulation_id, year
```

```
                                        )
        sc.calculate_standings()
        pc = PlayoffCoordinator(self.db_handler, self.
                                        simulation_id, year
                                        , game_simulator)
        pc.simulate_playoffs()

        # Start offseason
        if not self.is_benchmark:
            LotteryCoordinator(self.db_handler, self.
                                            simulation_id,
                                            year).
                                            generate_lottery
                                            ()
```

(g) Requirement 7 states that it should be possible to complete a Monte Carlo simulation of a large number of scenarios to have a distribution of outcomes of a given team and understand the optimal factors that led to a successful outcome. We implemented that in the *ScenarioManager*, where the total number S of scenarios could be defined. We also needed a way to group a single Monte Carlo experiment since each simulation was assigned a unique *simulation_id*. This was accomplished by creating a new table (*Scenarios*), where we had a 1:n relationship between columns *scenario_id* and *simulation_id*, respectively.

(h) Requirement 8 states that it should be possible to create a visualization of the results from the simulation results so that the interpretation of said results is easier for the system user. This was implemented as a Jupyter notebook (available on [4]), where the results from the chapter 5 are also available. Furthermore, if the user wants a different visualization (e.g., in PDF format), such an output can be quickly produced from the notebooks.

Finally, we end our reflections by briefly discussing our key learnings after writing the thesis and how our view regarding NBA analytics has evolved. We list our learnings below:

1. Reliably predicting game outcomes is challenging. We used relatively straightforward models (as described in chapter 5) and were able to draw interesting conclusions, but nevertheless our accuracy was very distant from the accuracy seen in other classification problems (such as image classification). We attribute this difficulty to the large number of factors affecting game outcomes, such as players scoring-, rebounding- and assists totals, but also players' mental- and physical health, as well as injury probabilities, matchups, among many others. Since modeling and simulating all those factors

comprises a very complex task, predicting game outcomes is also complex.

2. A similar argument can be made to simulating an NBA season as a whole. As discussed in this thesis, many factors play a role when deciding the likelihood of each team winning the championship, and modeling those is very complex.

3. We are skeptical that a complete simulation model taking into consideration all the relevant parameters for the league activities (e.g., draft, trades, free agency, etc.) can ever be developed, however we see value in partial solutions (like this thesis) which attempt to simulate parts of the dynamic and provide valuable insights.

NBA analytics, as briefly mentioned in chapter 6, is changing how the NBA game is played. For example, the most valuable players in 1990 were tall, strong players (like Shaquille O'Neal) that could dominate within a short distance from the basket. Currently, teams often play without a true big man (see the latest Golden State Warriors' title runs).

To summarize: our view of analytics is that it is a handy tool for helping with the team's decision-making and roster construction. However, it is not the only tool: each team will still be constructed differently due to its unique circumstances, such as market size, players selected in previous drafts, fan base, owner preferences, among many others. Hence we argue that analytics is only one of the significant factors affecting the likelihood that a team wins the NBA championship.

Chapter 6

Social and Ethical considerations

This chapter briefly discusses the social and ethical considerations that involve this thesis.

6.1 Data collection

We note that we used open-source code made available under permissive licenses, so that we are allowed to integrate it with our codebase, provided we include the necessary citations, as we did in this thesis. Moreover, public data freely available online concerning statistical performance of professional basketball players was used so that no ethical considerations in this regard need to be made.

6.2 Potential social effects

For this section, we asked ourselves the question: *What can go wrong?*

We anticipate two main discussion points that might arise from the results of this thesis, discussed in the following subsections.

6.2.1 Betting issues

We produced this framework to be used by NBA front office personnel, who could use the tool for professional purposes, observing the limitations and assumptions involved with the tool. However, regular fans could, in theory, also use the tool as a predictor for future events so that different activities could be based on the tool's outputs, such as bets, which could potentially lead to money losses.

To prevent such negative scenarios from happening, we again explicitly mention here that the results of this thesis are experimental software that has not been vetted by NBA personnel and does not guarantee any predictive power that can be used for game prediction.

6.2.2 Influence of analytics in basketball

Another adverse scenario that could follow from this thesis is the further dominance of analytics in basketball. As can be seen in figure 6.1, the 3-point rate percentage (ratio of 3-point shot attempts versus total shot attempts by a team) has been steadily increasing in the last 20 years, since the average value of the 3-point shot is 1.2 whereas the average value of a midrange shot is 1 (assuming league average accuracy). As analytics become increasingly omnipresent when playing the game, specialists and fans, in general, have sometimes opposed such changes and argued that the old style of play, more physical and centered in the paint, was more "fun".

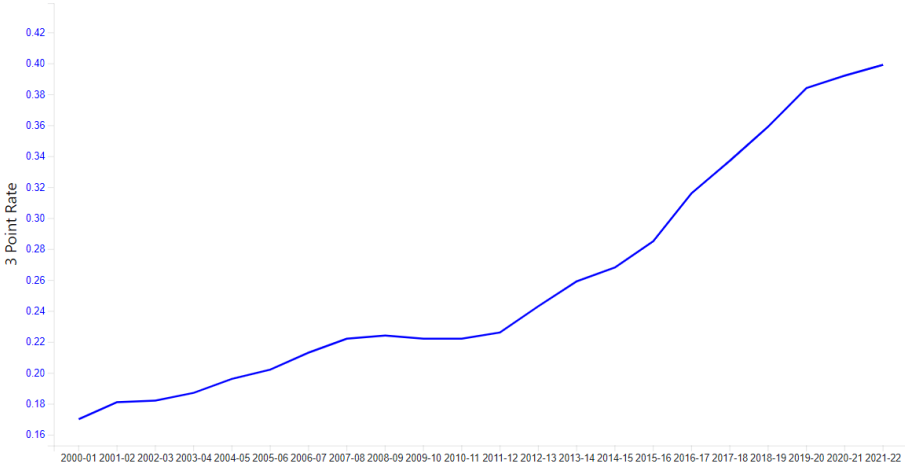

Figure 6.1: Time evolution of 3-point attempts. From [3].

In this sense, this thesis aims to provide another use-case of statistical analysis and analytical tools to the NBA front office. This could, in theory, further contribute to the analytical revolution we are already seeing, which could be detrimental to some fans that do not appreciate the direction in which the game is evolving.

Finally, another negative effect of this thesis could be the complete adoption of the tool by all 30 NBA teams, which is improbable. However, in this imaginary scenario, we would expect all teams to try to build their rosters in similar ways (i.e., prioritizing specific roster configurations), which would probably make the league as a whole less appealing to the public (given the lack of originality within team

construction). On the other hand, we would expect a more homogeneous level of competition, as all teams would have an equal capacity to build a championship team.

Chapter 7

Conclusions and future work

In this chapter, we draw conclusions after implementing the system and briefly digress on which extensions of the system could be achieved with future work.

The main contribution of this project has been to create a simulation tool that allows NBA General Managers to use Monte Carlo simulations to identify the most critical factors contributing to their team winning a championship in the upcoming 5 NBA seasons.

The primary motivation for this work was the non-reproducibility of the decisions of the General Manager, in contrast with the player actions during a game, which can be perfected during practice. This tool allows the professional to have additional data points to justify a particular decision.

Multiple requirements were stated to produce a tool able to help the stakeholders mentioned above, mainly revolving around the fact that it should be as close to reality as possible, including NBA rules, evolution of young players selected in the draft and trades between teams. Although a few simplifications and approximations have been made, most requirements were fulfilled.

In order to complete those requirements, a solution design was proposed and implemented where the major drivers for the design were modularity, reproducibility and performance. This was fully achieved and involved a few trade-offs, especially concerning possible database choices. It is also worth noting that we wanted to make future contributions as seamless as possible to integrate, hence thrived for making new and existing modules modular and "plug-and-play" so that no unforeseen dependencies might obstruct the implementation of new ideas.

Interesting results were also presented, divided into two major sections. Historical results were given to compare actual results in the NBA with expected results to

demonstrate that our tool produces trustworthy simulations. Results concerning future simulations were also shown, highlighting the importance of the scoring output of the leading players on the roster and discussing how important is the evolution of drafted players for an NBA franchise.

Finally, we highlight below some potential ideas for future work.

Free agent signings - The only way we considered players switching teams during a simulation is when they were traded. In reality, after a contract expires, a player can join another team for a variety of reasons (and also in many ways, such as sign-and-trade, after being waived, etc.). Adding considerations regarding contracts and the attractiveness of certain cities to incoming free agents, such as Los Angeles or Miami, would be valuable when predicting more realistic scenarios. CBA rules [10] should also be part of the simulation, otherwise simulated scenarios might not have been eligible to occur in reality.

Extend trade mechanism - We considered a very simplistic scenario for trades: only two teams and the sum of the points per game metric of all players should match on both sides (respecting a given tolerance). In reality, trades are much more complex and can involve cash, draft picks (with or without protection) and multiple players, hence the talent level is often unbalanced given the uncertainty of some parts of the trade, especially draft picks. In that sense, expanding our trade component to account for those extended scenarios would also allow the tool to generate results encompassing more realistic trade results.

Simulation of games - We trained an algorithm for predicting game outcomes that examined only 20 features (points scored per game of the top 5 scorers of each team as well as rebounds per game of the top 5 rebounders on each team). An improved model could consider additional features, such as including the assists per game, or even more complex ones, for instance, how many games were played by a team on the previous five nights. Another option would be plugging in pre-trained models instead of training our own models.

Injuries - A significant factor that dictates the success or failure of a team in a given season is injuries. We did not consider injuries as part of the simulation, and an additional module assigning a likelihood of injury to all players (and also the severity) would be beneficial.

Further analysis of results - Further exploration of simulated data could yield additional results. An example could be examining the influence of categories of draft picks (e.g., picks 1-5 are worth twice as much as picks 5-20 and three times as much as picks 21-60). Another interesting example could be breaking down the simulated scenarios into final positions in the standing instead of simply differentiating between championship or non-championship seasons.

Automatic generation of reports in PDF - A straightforward improvement on the tool would also be the automatic generation of a PDF report, allowing non-technical users to profit from the simulation results immediately.

All the work produced in this thesis is open-source and can be accessed online [4].

Bibliography

[1] *2022-23 NBA Player Contracts*. https://www.basketball-reference.com/contracts/players.html. Accessed: 2022-08-06.

[2] Takuya Akiba et al. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[3] Darryl Blackport. *PBP Stats - Stats from play by play data*. https://www.pbpstats.com/. Accessed: 2022-09-02.

[4] Gabriel Bueno de Moraes Fior. *NBA Franchise Manager - how to build a championship team*. Version 1.0.0. Oct. 2022. URL: https://github.com/gabrielfior/nba-franchise-manager.

[5] Benjamin T Foster and Michael D Binns. "Analytics for the Front Office: Valuing Protections on NBA Draft Picks". In: MIT Sloan Sport. Anal. Conf. 2019.

[6] David Freedman. *CBA Breakdown - Understand the NBA's collective bargaining agreement*. https://cbabreakdown.com/salary-cap-overview. Accessed: 2022-09-02. 2019.

[7] J. Howard and S. Gugger. *Deep Learning for Coders with Fastai and Pytorch: AI Applications Without a PhD*. O'Reilly Media, Incorporated, 2020. ISBN: 9781492045526. URL: https://books.google.no/books?id=xd6LxgEACAAJ.

[8] *Implied Probability Calculator*. https://www.gamingtoday.com/tools/implied-probability/. Accessed: 2022-07-29.

[9] Robert Kern. *NEP 19 Random number generator policy*. https://numpy.org/neps/nep-0019-rng-policy.html. Accessed: 2022-09-02. 2019.

[10] National Basketball Association. *National Basketball Association Constitution and by-laws*. https://ak-static.cms.nba.com/wp-content/uploads/sites/4/2019/09/NBA-Constitution-By-Laws-September-2019-1.pdf. 2019.

[11] National Basketball Association. *NBA Draft Lottery: Odds, history and how it works*. https://www.nba.com/news/nba-draft-lottery-explainer. Accessed: 2022-09-03. 2022.
[12] NBA. *2021-2022 Season League Standings*. http://global.nba.com/standings/. July 2022.
[13] *Nba Api - An API Client package to access the APIs for NBA.com*. https://github.com/swar/nba_api. Accessed: 2022-09-01.
[14] *NBA Preseason Odds*. https://www.basketball-reference.com/leagues/NBA_2018_preseason_odds.html. Accessed: 2022-07-29.
[15] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
[16] *PostgreSQL documentation*. https://www.postgresql.org/docs/current/. Accessed: 2022-09-02. 2022.
[17] N. Silver et al. *2021-22 NBA Predictions*. https://projects.fivethirtyeight.com/2022-nba-predictions/games/. Accessed: 2022-08-07.
[18] *SQLite documentation*. https://www.sqlite.org/. Accessed: 2022-09-02. 2022.
[19] Paras Varshney. *Q-Q Plots Explained*. https://towardsdatascience.com/q-q-plots-explained-5aa8495426c0. Accessed: 2022-09-03.

Appendix A

Further considerations on the validity of Gaussian distribution for player scoring

In this appendix, we justify our arguments that a Gaussian distribution can approximate the player scoring outputs. We refer the reader to figure 4.3 for a visual inspection of the distribution of 3 player scoring averages, which we summarize in table A.1.

Player name	Mean (μ)	Standard deviation (σ)
Stephen Curry	27.68	9.99
Monte Morris	10.52	5.16
Isaac Bonga	3.07	3.47

Table A.1: Summary of statistics of points scored in each game by the three benchmark players.

As shown in figure A.1, we use QQ-plots for visually determining if a Gaussian distribution is a valid approximation for each player's single game scoring values. The basic idea [19] is that the distribution (in our case, an array of points scored in N games) is ordered and compared with another ordered distribution that we believe can serve as a good approximation for the data (in our case, a normal distribution). Although not a rigorous mathematical proof, QQ-plots are an excellent visual indicator of the feasibility of the Gaussian approximation.

If our dataset were to follow a normal distribution, we would expect the ordered terms from both distributions to be in the same quantiles. That is why the x-axis

and y-axis are labeled theoretical and data quantiles, respectively.

We conclude that all three scoring outputs can be modeled as gaussian distributions. Furthermore, we extrapolate this interpretation to the scoring outputs of all players, given that players of different scoring levels (high, medium, low) were considered in our analysis.

We also point out that the scoring output of the higher-scoring player (Curry) follows more closely the target distribution than the lower-scoring player (Bonga). In other words, we would expect a higher error between actual- and simulated scoring totals for lower-scoring players than for higher-scoring players.

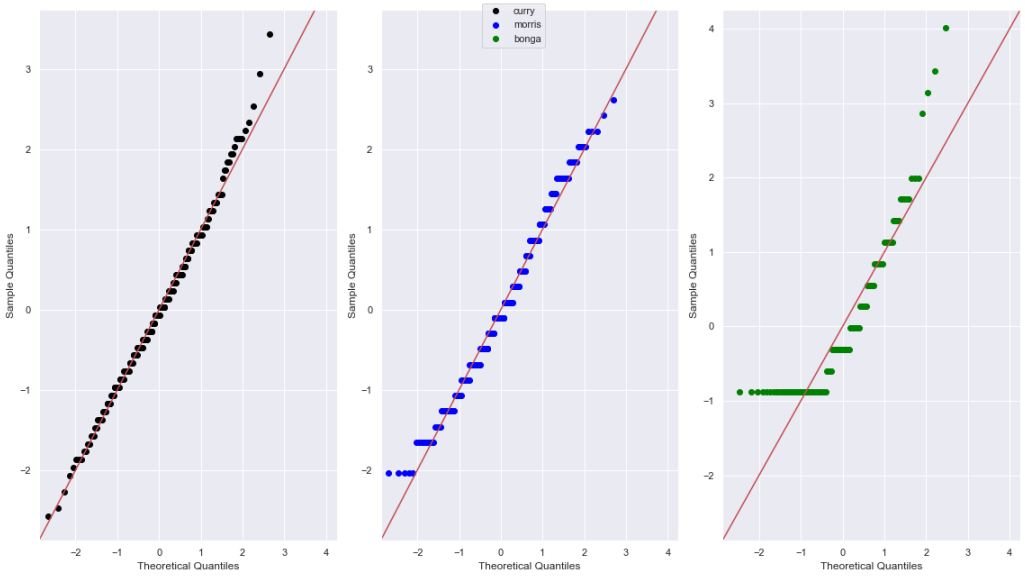

Figure A.1: QQ-plots for the three analyzed players.