 Recommendation Systems for Influencing Player Competence and Engagement in Match-Based Games

Author: Zhengxing Chen

Committee: Magy Seif El-Nasr (Advisor)
 Christopher Amato
 Seth Cooper
 Yizhou Sun

A dissertation submitted in fulfillment of the requirements for the degree of Doctor of Philosophy
in the
College of Information and Computer Science
Northeastern University

December 13, 2018
Declaration of Authorship

I, Zhengxing Chen, declare that this dissertation titled, “Recommendation Systems for Influencing Player Competence and Engagement in Match-Based Games” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.

- Where any part of this dissertation has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.

- Where I have consulted the published work of others, this is always clearly attributed.

- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this dissertation is entirely my own work.

- I have acknowledged all main sources of help.

- Where the dissertation is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: __________________________________________

Date: __________________________________________
Abstract

For years, video games have garnered the attention of people worldwide, providing a unique environment for entertainment, education, scientific problem solving, promotion of health awareness, and socialization. With the increase of video games’ popularity, a critical question arises: how do we keep a player engaged?

In this dissertation, I focused on increasing player engagement, where I define engagement as the continued desire to play a game repeatedly during one session or over a longer period of time. I proposed three recommendation systems targeting different types of in-game elements. I focused on online match-based games, where players competitively play against others. Further, I focused on recommendations in the pre-match stage. My hypothesis is that the recommendation systems I developed will increase player engagement by giving players the ability to make informed choices concerning in-game elements, such as which characters to play as, which items to equip, or which opponents to play against. My theoretical foundation for this work is based on two psychological theories that have shaped understanding of player sustained engagement in games: Self Determination Theory and Flow Theory. In such theories, scholars have shown that competence, or the ability of a player to feel competent given the tasks they are doing within the game, has a direct relation to sustained engagement, or the desire to play the game repeatedly. Additionally, Flow theory describes a state of ‘flow’ where players are sufficiently challenged, where the task they are doing is neither too easy or too hard. Based on these theories, the recommendation systems I proposed recommend in-game elements to players that influence their in-game choices, and subsequently will have an impact on the way they play, competence they perceive, and eventually their engagement.

To develop these recommendation systems, I explored two related research questions. The first is concerned with how to develop effective and efficient recommendation systems that can recommend in-game elements that will have a winning potential for players, in both one-vs-one and team-vs-team settings. Targeting this question allows us to focus on developing experiences that are not frustratingly hard
for players, and thus help increase their winnings. It should be noted that these systems can also work in reverse, meaning they can also suggest items that make players lose. The second research question is concerned with studying how to balance win/loss ratio of matches through designing in-game element recommendation systems which directly improve player engagement in relation to win/loss ratio and its impact on the churn rate.

For the first research question I proposed two systems: (a) **Q-DeckRec** a recommendation system for one-vs-one Collectible Card Games (CCGs). In these types of games, the player often starts with a deck of cards of their choice. Q-DeckRec recommends winning-effective decks. This system presents a novel approach to search a large space of possible card decks to recommend, using minimal computational resources after a training phase. In addition to cards as starting items to recommend, many games, such as MOBA (Multi-player Online Battle Arena) games, require players to select a set of characters to play with against another team. For these types of games, I proposed (b) **DraftArtist**, a recommendation system that recommends winning-effective characters. This proposed system presents a novel contribution that efficiently searches for the best possible characters to recommend given a large number of possibilities, uncertainty caused by not knowing what the other team will select, and the desirability to select characters that synergize with teammates and counter the opponents. Using a match outcome prediction model trained on real data, we find that teams following our recommendation algorithm have higher predicted win rates against teams constructed by other character selection strategies. Our algorithm maintains sufficient efficiency to be deployed in real-world scenarios.

For the second research question, I proposed an opponent recommendation system, called **Engagement-Optimized Matchmaking** (EOMM). EOMM is the first system that formally treats matchmaking as an optimization problem to maximize player engagement quantitatively. Our system shows significant improvement in enhancing player engagement across all players as compared to other matchmaking methods.
The contribution of my dissertation is in developing three novel recommendation systems that target improving player engagement through increasing competence by giving players in-game element recommendations that increase possibility of a players feeling competent or entering a state of Flow by either increasing the probability of them winning or balancing the win/loss ratio to decrease the probability that they leave the game. There are much work left for future research, such as studying systems tailored for more specific types of in-game elements, as well as those which can make central decisions on different types of recommendations for any particular player.
I would like to thank my advisor Prof. Magy Seif El-Nasr for her patience, motivation, and knowledge. I could not forget our first meeting where she shed lights on game analytics and encouraged me to take on this direction. Without her support over the last five years, I could never have been so fortunate to carve a niche in the interdisciplinary research area of video games and artificial intelligence.

My sincere thanks also go to my dissertation committee members: Prof. Christopher Amato, Prof. Yizhou Sun, and Prof. Seth Cooper. They provided me with enormous helps and inspired my research in diverse perspectives.

I am grateful to all collaborators I worked with during my research as well as colleagues I met during my internships at Electronic Arts, eBay, Google, and Facebook. My special gratefulness goes to Prof. Truong-Huy Nguyen for suggesting me PhD and life advice, and Dr. Su Xue and Dr. John Kolen for fostering my incredible internship experience at Electronic Arts.

I thank Prof. Karl Lieberherr and Prof. Stephen Intille for their kind helps in the first year of my PhD. I also thank all my friends and lab mates for their support, understanding, and encouragement.

I cannot thank more than enough to my parents and grandparents, who have taught me to become kind, strong, and wise throughout my whole life.

Last but by no means least, I would like to express my thankfulness to my wife, Shiyun, for accompanying me through the so far most challenging yet most memorable time in my life. We will have much more adventure to experience in the future.
Contents

Declaration of Authorship iii

Abstract v

Acknowledgements ix

1 Introduction 1
   1.1 Motivation and Research Question 1
   1.2 Dissertation Overview 7
      1.2.1 Starting Item Recommendation in One-vs-One 7
      1.2.2 Character Recommendation in Team-vs-Team 8
      1.2.3 Matchmaking for Optimizing Overall Engagement 10
      1.2.4 Dissertation Outline 11

2 Related Work and Background 13
   2.1 Game Background 13
      2.1.1 Collectible Card Games (CCGs) 13
      2.1.2 Multiplayer Online Battle Arena (MOBA) 15
   2.2 Theories of Player Engagement and Competence 18
   2.3 Data-Driven Models for Player Engagement 20
   2.4 Techniques towards Improving Player Engagement 22
      2.4.1 Dynamic Difficulty Adjustment 22
      2.4.2 Procedural Content Generation 24
      2.4.3 Recommendation Systems 25
         Starting Item Recommendation 27
         Character Recommendation 28
         Matchmaking (Opponent Recommendation) 29
5.4 Theoretical Findings .............................................. 79
5.5 Case Study ......................................................... 80
   5.5.1 Data Collection .............................................. 80
   5.5.2 Preparation .................................................. 80
   5.5.3 Simulation Procedure ....................................... 83
   5.5.4 Results and Discussion ..................................... 84
5.6 Summary .......................................................... 86

6  Applicability, Application and Ethical Issues 87
   6.1 Applicability and Application ................................. 87
      6.1.1 Q-DeckRec ................................................. 87
      6.1.2 DraftArtist ................................................ 89
      6.1.3 EOMM ..................................................... 90
   6.2 Ethical Issues .................................................. 92

7  Limitation and Future Work ........................................ 95

8  Conclusions ........................................................ 99

Bibliography .......................................................... 101
List of Figures

2.1 The game map of DOTA 2. The bases of the two teams are on the ends of the diagonal. Player-controlled characters can move through lanes (Top, Bottom, and Middle) as well as partial areas of jungles. Most MOBA games have similar game maps like this one. Figure adapted from the Internet: https://bit.ly/2LbD3dv ........................ 16

2.2 Drafting interface in DOTA 2. Figure downloaded from the Internet: https://bit.ly/2uBY8ni ................................. 17

4.1 Diagram representing the four steps of UCT. Diagram adapted from Browne et al. (2012). ................................................. 54

4.2 Drafting order in match mode "Captain Mode" in DOTA 2. The green and purple cells represent the turns of two different teams (i.e., Radiant and Dire), assuming the green team starts first. Diagram is adapted from Gamepedia (2018) and best viewed in color. ........ 64

5.1 Model matchmaking on a complete graph. Each node represents a player, and every edge is associated with the sum engagement metric of two players if paired. EOMM amounts to finding an optimal pair assignment on $G$. ................................. 76

5.2 Predicted win probability vs. real win probability. Real win probability is the ratio of matches, with similar predicted winning probabilities, whose outcomes are real “Wins”. ................................. 81

5.3 Predicted churn risk vs. real churn risk. Real churn risk is the ratio of matches, with similar predicted churn risks, which are indeed the last match before churn. ................................. 83
# List of Tables

3.1 Results of Genetic Algorithm Approach ........................................ 46  
3.2 Results of Q-DeckRec ................................................................. 46  
3.3 Performances of MC simulation approach ...................................... 46  

4.1 Performance of Team Win Rate Predictors .................................... 58  
4.2 Mean predicted win rate of $UCT_{n,1}$ (row) vs. $UCT_{n,c}$ (column). Numbers are from the view of the row strategy. Bold cells indicate the best value of $c$ for $UCT_{n,c}$ for each $n$, as they force $UCT_{n,1}$ have the lowest win rate. .................................................. 60  
4.3 Mean predicted win rate of row strategy vs. column strategy. Simulations are based on the drafting rules of match mode “All Ranked”. Numbers are from the view of the row strategy. The strategies are sorted in ascending order by their strengths, from top to bottom, and from left to right. Win rates symmetric to diagonal and always sum to one, thus half of them are omitted. ........................................ 62  
4.4 Average wall time per pick of different strategies (unit: millisecond, match mode: All Ranked). ...................................................... 62  
4.5 Mean predicted win rate of row strategy vs. column strategy. Simulations are based on the drafting rules of match mode “Captain Mode”. The table can be viewed in a similar way as Table 4.3. ......................... 63
5.1 An example of the impact of player states on their engagement. Data is from a popular PvP game made by EA. Average churn risks vary drastically upon players’ recent three match outcomes (Win, Lose or Draw). Churn risk is measured by the ratio of the players who stop playing within a period time (7 days in this table) after a match. The churn risk of some states with repeated losses (5.1%) is almost twice as much as those of other “safer” states (2.6%-2.7%).

5.2 Average number of retained players per round of matchmaking simulation. 10,000 rounds of matchmaking were simulated.
Chapter 1

Introduction

1.1 Motivation and Research Question

The video game industry has produced many games that engage a large and diverse audience of committed players. The *Super Mario Bros.* (Nintendo Co. Ltd) franchise, for example, sold over 160 million copies since the 1980s (GameCubicle, 2017). In the summer of 2016, *Pokémon Go* (Niantic, Inc.) swept the globe with more than 500 million downloads (Polygon, 2017), triggering crowds to walk on the street as part of the game. Recently, there has been a rise in competitive team-based player-vs-player games, which engage millions of players. As an example, *League of Legends*, a 5-vs-5 online competitive game, engaged 90 million registered users, 27 million unique daily players and 7.5 million concurrent users at peak (Minotti, 2015; Tassi, 2016). The game is characterized by mechanics that engage players in both competition and collaboration as well as problem solving and strategy formation and execution.

At the same time, the viewership of video games has also grown rapidly. E-Sports, a subset of highly competitive video games, was estimated to have hundreds of millions of viewers in 2017. This number is projected to grow at a rate of 12% each year (SuperData, 2017a). There are now internet-based channels like *Twitch.tv*, which was developed to specifically show videos of gameplay with commentators discussing strategies, just like other sports events (SuperData, 2017b).

Besides entertainment, video games have also been utilized for various purposes, such as promoting aesthetic appreciation (Järvinen, 2008), health awareness (Shiyko et al., 2016), scientific problem solving (Cooper et al., 2010), education (Gee, 2003), and socialization (Ferguson and Olson, 2013). A salient example is *Foldit* (Cooper
et al., 2010), a multiplayer online game that engages non-scientists in solving the protein structure prediction problem, which remains largely unsolved by computational approaches. An increasing number of empirical studies also support the role of video games in fostering mental health (Jones et al., 2014). For instance, some studies show that moderate video game playing helps reduce emotional disturbances in children (Hull, 2009) and relieve stress among adolescents (Colwell, 2007). More interestingly, researchers have also argued that video games can encourage players to adopt healthier physical life (Shiyko et al., 2016). Additionally, physiological experiments around visual attention and abilities have shown that expert video game players tend to have higher visual abilities than others (Green and Bavelier, 2003; Li et al., 2009). This suggests that video game playing may even become a helpful complement to traditional eye-correction techniques that are used in the clinic to improve eyesight (Li et al., 2009).

As video games continue to be used for many societal benefits, one question arises naturally: how can we keep players engaged to reap the benefits of such experiences? Here, I adopt the most straightforward definition of player engagement as the "continued desire" to play a game repeatedly during one session or over a longer period of time (Schoenau-Fog, 2011). We want to note here that this goal also has a disadvantage and ethical implication. For example, we do not want to encourage excess addiction to video games, thus the application of the systems developed here need to be done with care and thought. We will discuss this further within Section 6.2.

While there exist many kinds of video games, in this dissertation, I studied player engagement in match-based video games (Guo et al., 2012), an interesting test bed with rich multi-player strategies and interactions. In such games, multiple players, often random online players, compete for a defined victory goal in small sessions called matches, each spanning a few minutes to hours. Matches are independent from each other, in the sense that in-game elements (Ralph and Monu, 2015; Fullerton, 2008), which can range from game artifacts (e.g., levels, maps, and weapons) to players or opponents, get refreshed at the beginning of each match so that every participant starts from a relatively similar state. A simple example is online Chess matches; at the beginning of each match the board gets refreshed and new opponents might be paired. Match-based video games also imply there are multiple players involved,
1.1. Motivation and Research Question

though some could be computer-controlled. Match-based video games are represented in a wide range of genres and titles, ranging from one-vs-one online board games like Chess to team-based games like League of Legends (Riot Games) and DOTA 2 (Valve Corporation), where players need to collaborate. Therefore, match-based video games arguably have more variety in player interactions and strategies than single-player games such as Tetris.

My dissertation is grounded on an assumption that controlling the in-game elements presented to players could influence the way they play, their subjective experience of playing the game, and eventually their engagement. To this end, I propose to use recommendation systems (Medler, 2011) for improving player engagement. These recommendation systems act as information filtering systems for suggesting in-game elements to players among a variety of choices often offered in modern video games, such as which characters to play as, which items to equip, or which opponents to play against. Following my assumption, recommendation systems could increase the likelihood of players favoring the recommended in-game elements, which are predicted to affect player engagement positively thus players can enjoy better engagement.

While there are other equally valid methods for influencing player engagement, such as manual game design and Procedural Content Generation (PCG) (Yannakakis and Togelius, 2011; Togelius et al., 2011), which use algorithms to generate game content with limited or indirect user input (discussed in Section 2.4.2), I concentrated on in-game element recommendations for several reasons. First, recommendation systems as an algorithmic approach can be deployed in a scalable, just-in-time, and on-demand fashion, alleviating the enormous cost and effort associated with manual design. Second, studying other methods may require additional data and software tools which I do not have access to; for example, studying PCG may require permission to modify game content in existing games that are not available to me. Third, research work on in-game element recommendations, which I will overview in Section 2.4.3, is still scarce, which presents an opportunity to explore the topic further.
Chapter 1. Introduction

Recommendation systems can be applied at many different points in a match-based video game, such as the pre-match stage and mid-match stage. In this dissertation, I focused on recommendation systems applied in the pre-match stage, which is defined as the time window from the moment a player requests to start a match until the match officially begins. Depending on the nature of the game, there are various in-game elements to be determined before the match can officially start, such as opponents, characters to play as (these characters are important as they have specific characteristics and power that can give one advantage over other teams if selected properly), and starting items to bring into the match. The in-match stage period is when the real match takes place and players engage in competition. I chose to focus on the pre-match stage, because it is the initial, yet fundamental, gateway into the main experience of the match. Bad determination of in-game elements in the pre-match stage may jeopardize player engagement from the beginning of the match and could lead to toxic behaviors such as quitting the match early (Shores et al., 2014) and cyberbullying (Kwak, Blackburn, and Han, 2015).

Given this scope, I define the following fundamental research question for my dissertation:

How can we design in-game element recommendation systems working in the pre-match stage, which can improve player engagement in match-based video games?

Under the umbrella of this fundamental research question, I further proposed two research sub-questions. The first research sub-question takes an indirect approach to improve player engagement by recommending winning-effective in-game elements to players, where I define winning-effective in-game elements as those that could boost players’ winning chances if selected.

Recommending winning-effective in-game elements for improving player engagement is mostly inspired by two theoretical foundations. The first theory is Self Determination Theory (SDT) (Ryan, Rigby, and Przybylski, 2006; Przybylski, Rigby, and Ryan, 2010; Yee, 2006a; Wu, Wang, and Tsai, 2010; Sherry et al., 2006; Lazzaro, 2004; Schoenau-Fog, 2011). The reason I adopted this theory is due to the extension of this work by Przybylski, Rigby, and Ryan (2010) and Ryan, Rigby, and Przybylski
1.1. Motivation and Research Question

(2006), showing the constructs that explain sustained engagement, defined as the tendency to be in the game longer or to return to the game when offered the choice. SDT is concerned with intrinsic and extrinsic motivational factors. The early focus of SDT is on intrinsic motivation, which is the innate desire to satisfy psychological needs beyond external rewards or goals, such as money. The theory defines three psychological motivational factors: competence - the need for challenge and the feeling of being capable and effective, autonomy - the sense of volition or willingness when doing a task, and relatedness - the feel of being connected with others.

Another important theoretical foundation that influenced my first research sub-question is the study of Flow (Csikszentmihalyi, 1990), a theory concerned with optimal experience. Flow is a state that refers to an optimal mental state of engagement where people get fully immersed, and where they engage and get so consumed by the activity that they lose track of time (Csikszentmihalyi, 1990). Csikszentmihalyi studied people as they get into this state since mid-1970s. He was interested in discovering factors that explain why people immerse in their tasks. Sweetser and Wyeth (2005) adapted this theory to video games and identified several factors affecting players getting into this state, specifically a balance between skill and challenge, which aligns with the construct of competence from SDT (Ryan and Deci, 2000).

Both Flow and SDT show that engagement, as I define it, can be achieved through increasing competence. In reality, not all players are competent enough to maintain optimal player engagement. As modern video games with rich design often offer several choices of in-game elements which may overwhelm players, one cause of incompetence shown by several studies is the challenge of making winning-effective choices of in-game elements, such as what character/items to select that best fits a match (Looi et al., 2018; Summerville, Cook, and Steenhuisen, 2016; Hanke and Chaimowicz, 2017). Our rationale is to use recommendation systems to filter winning-effective in-game elements efficiently from a large number of candidates and then present the recommendations to players. We assume if incompetent players adopt our recommendations, their chance of victory will be improved, which will consequently lead to better game outcomes and improved competence, thus finally better engagement as theories suggest.
It is important to note that, in this dissertation, I focused on the recommendation techniques for searching winning-effective in-game elements, but I did not specifically evaluate the resulting change in player competence or engagement due to the use of the recommendation. Neither did I investigate how such recommendation should also avoid under-challenging players. These will be explored further in future research.

Furthermore, considering that there are one-vs-one and team-vs-team games, I would like to study the two settings separately because the latter setting has more complexity in the sense that the strategies by multiple players might simultaneously influence the strategy to identify winning-effective in-game elements for the player themselves.

In summary, I formalize my first research sub-question as:

How can we design systems working in the pre-match stage that can recommend winning-effective in-game elements, within the settings of one-vs-one and team-vs-team games?

(R.S.Q. 1)

While recommendation systems focusing on improving winning-effectiveness are grounded by theories like SDT and Flow Theory, they rely on the intermediate links between winning-effectiveness and player competence and between player competence and engagement. Bypassing these intermediate links, in-game element recommendations for directly improving player engagement determine an engagement metric and efficiently search for the in-game element that achieves the optimal value of that engagement metric. Specifically, I choose to use data-driven approaches, which determine recommendations purely based on analysis and interpretation from past player data without the need to rely on psychological and sociological theories as in R.S.Q. 1. Data-driven approaches often use machine learning models to establish numerical mappings from players’ historical data and profiles to engagement metrics (Yannakakis et al., 2013). Engagement metrics can be seen as proxies for engagement which could be quantitatively calculated from past data without the need to conduct particular experiments to measure cognitive states or subjects’ interest, thoughts or emotions. Empirical studies have examined proxies
1.2. Dissertation Overview

for engagement such as: purchases in the game (Xie et al., 2015; Sifa et al., 2015), the number of matches played within a time window (Xue et al., 2017; Weber et al., 2011), or churn risk (Hadiji et al., 2014; Harrison and Roberts, 2012), where churn risk is a conventional name defined as the likelihood of a player to discontinue play or quit the game.

In summary, my second research sub-question is formalized as:

How can we design systems working in the pre-match stage which recommend in-game elements for improving player engagement through data-driven approaches?

(R.S.Q. 2)

In seeking the answers for R.S.Q. 1 and R.S.Q. 2, my dissertation outlines three recommendation systems, with each system dealing with a specific type of in-game element, namely starting items, characters, and opponents. The overall contribution of my dissertation is the enrichment of arsenal of recommendation techniques that recommend pre-match in-game elements which are either winning-effective or positively influential in player engagement measured quantitatively.

In the following section, I will briefly overview each recommendation system. Although this dissertation covers my own work, I could not have completed much of the research without collaborations with other researchers appearing in Chen et al., 2018a; Chen et al., 2018b; Chen et al., 2017. I will therefore shift the narrative point-of-view from “I” to the academic form of “we” in the rest of this dissertation.

1.2.1 Starting Item Recommendation in One-vs-One

We first attempt to answer R.S.Q. 1 in the setting of one-vs-one match-based video games. We chose to study winning-effective starting item recommendation in one-vs-one Collectible Card Games (CCGs). CCG is a popular genre within match-based video games; digital CCGs like Hearthstone reached a record of 40 million registered accounts in 2016 (Lazarides, 2017). In CCGs, the pre-match in-game items required to be selected by players is a deck, a fixed number of cards (usually just a few dozens)
that every player is asked to select from a pool of cards (usually a few hundreds) before a match starts. Decks have an important impact on players’ competence because in-match capabilities and strategies could largely depend on the choice of the starting items. Ideally, a winning-effective deck should contain cards which have effective synergy with each other and opposition against the cards in the opponent’s deck.

CCGs feature the challenge that the possible deck choices are exponentially large which make efficiently searching for winning-effective decks a difficult problem. For example, in the particular game we study, which we will detail in Chapter 3, the number of all possible decks is around $1.4 \times 10^{25}$.

To recommend winning-effective decks, we formalized the problem of searching winning-optimal decks as a combinatorial optimization problem (COP) and proposed a deck recommendation system that could approximately, but efficiently, solve the COP. The deck recommendation system proposed an efficient solution to this problem (Chen et al., 2018a). It first extracts a deck building policy from simulation-based deck evaluations in a training phase. The policy encodes synergistic and oppositional relationships among cards into a parameterized machine learning model, which is then used to construct a winning-effective deck based on any given opponent in the future. The algorithm developed is cheap enough for the proposed system to be deployed for large-scale or real-time application, e.g., an online CCG’s backend for recommending winning-effective decks to a population of online players. We conducted experiments to show that after the training phase, the proposed deck recommendation system is able to build highly winning-effective decks with efficient use of CPU time, which is not achievable by existing solutions.

1.2.2 Character Recommendation in Team-vs-Team

To further the work discussed above towards addressing the R.S.Q. 1 mentioned above, we chose to study winning-effective character recommendation in a different setting, specifically the team-based game genre, Multi-Player Online Battle Arena (MOBA). MOBA is one of the most popular type of e-sports games (SuperData,
In a MOBA match, two teams, composed of 5 players each, combat in a virtual environment; each player controls a character to cooperate with other teammates in attacking opponents’ characters, armies, and structures, while defending their own in-game properties. In most popular MOBA games like *League of Legends* and *DOTA 2*, there are possibly more than 100 characters that can be picked by a player at the pre-match selection phase. As estimated by Hanke and Chaimowicz (2017), the number of total possible team compositions (10 characters in a match) is approximately $1.56 \times 10^{16}$. What increases the difficulty of winning-effective character recommendation is the order of character selection: the two teams alternate to pick characters in a certain order (varied upon specific match modes) until every player gets one character selected. In order to pick a winning-effective character, players not only need to know complicated synergistic and oppositional relationships among characters, but should also be able to predict what characters will be selected by other players and how they will affect their choices.

Like the previous approach described above, we first formally defined the problem of identifying the winning-optimal character and then proposed a system that can search for winning-effective (approximately winning-optimal) characters. To this end, we viewed the character selection phase between two teams as a combinatorial game (Browne et al., 2012). Under this problem formulation, the winning-optimal character for a player is the one that leads to a character line-up with the largest predicted win rate on its team, assuming that both teams behave optimally in the character selection phase. We proposed a character recommendation system that efficiently identifies the approximate winning-optimal character for team victory while considering the character selection strategies adopted by other players (Chen et al., 2018b). The proposed system utilizes a type of efficient search algorithm called Monte Carlo Tree Search (MCTS) (Coulom, 2006), which could estimate the long-term value of each character candidate through efficiently simulating possible following character selections. We conducted empirical simulation experiments, demonstrating that the proposed system is superior in the problem of recommending winning-effective characters over other baseline and state-of-the-arts strategies.
1.2.3 Matchmaking for Optimizing Overall Engagement

To answer R.S.Q. 2, we chose to study the opponent recommendation problem, better-known as matchmaking (Medler, 2011), in which the goal is to connect and divide a population of online players into individual matches. In this problem, players themselves are the subject of recommendations. Using a data-driven approach, we designed a matchmaking framework which determines the recommended opponent(s) for each player such that their resulting match outcomes would result in matches that optimize engagement metrics for the whole population of players.

A common strategy of current practical matchmaking systems is creating fair matches, that is, to create matches where they team up players who have similar skills and where match outcomes are hard to predict apriori. This strategy is inspired by the theoretical work we discussed above, such as Flow and SDT, where optimal engagement is achieved through a balance between challenge and frustration (Sweetser and Wyeth, 2005; Csikszentmihalyi, 1990; Chen, 2007; Graepel and Herbrich, 2006). The strategy essentially uses only one dimension of personal information: player skill, and thus do not link the opponent recommendation to quantitative measurement of player engagement.

However, always creating fairly-skilled matches may not be optimal to all types of players at all times in terms of specific engagement metrics. Let us consider a widely used engagement metric, churn risk, which is the likelihood that one player stops playing a game. Consider a cautious player who cares about protecting his rank among friends, and a risk taker who enjoys difficult matches. Pairing them with the similarly skilled opponents will affect these players’ churn risks very differently. Even for the same player, their churn risk after a match when they just lost three games in a row is different from when they won several games in a row.

We hypothesized that a more dedicated matchmaking strategy is needed to cater individuals’ dynamic desire for optimal engagement. We, therefore, proposed a matchmaking system which can model players with richer states besides skill. And instead of always creating matches with similarly skilled players, the recommended opponents are predicted to result in matches optimal for player engagement, which is measured by a data-driven approach over the player population (Chen et al., 2017).
We provided theoretical analysis about the optimality of and the conditions of the applicability of the proposed system and other existing methods. We built a simulated system using real game data, showing significant advantages of the proposed matchmaking system in optimizing a chosen engagement metric over existing methods. To our best knowledge, our contribution is the first work which has formally treated matchmaking as an optimization problem to maximize quantitatively measured player engagement metrics.

1.2.4 Dissertation Outline

The rest of the dissertation will be structured as follows.

- Chapter 2 introduces background theories as well as related work and gives an introduction to games used in this dissertation.
- Chapter 3 details a winning-effective starting item recommendation system, \textit{Q-DeckRec}, in a one-vs-one CCG.
- Chapter 4 describes a winning-effective character recommendation system, \textit{DraftArtist}, in a team-vs-team MOBA game. Chapter 3 and 4 answer \textbf{R.S.Q. 1} by experimentally showing that the proposed recommendation system are effective and efficient in searching for winning-effective in-game elements.
- Chapter 5 details a data-driven matchmaking framework, \textit{EOMM}, which recommends opponent(s) for each player in order to optimize engagement metrics. It addresses \textbf{R.S.Q. 2} by showing a data-driven approach for improving player engagement directly.
- Chapter 6 discusses applications of our proposed systems and Chapter 7 discusses limitations and future work.
- Chapter 8 concludes the dissertation and outlines the contributions of the dissertation work.
Chapter 2

Related Work and Background

In this chapter, we first introduce several game genres and games that we will refer to in the rest of this dissertation. We then review several theoretical foundations about player competence and engagement that we used. Next, we review existing endeavors in adaptive techniques that specifically target player competence and engagement in video games. Finally, we review churn prediction models, which quantify and predict churn as these are critical to our proposed recommendation systems.

2.1 Game Background

2.1.1 Collectible Card Games (CCGs)

Collectible Card Games (CCGs) is a class of games that is the basis of our proposed in-game item recommendation system for one-vs-one game settings, described in Chapter 3. CCGs have been popular since the 90s, evidenced by the large player base of these kinds of games. For instance, *Magic: the Gathering* has more than 20 million players globally (Guinness World Record, 2016). Additionally, an online free-to-play CCG *Hearthstone* (Blizzard Inc.) reached a record of 40 million registered accounts in 2016 (Lazarides, 2017). Since our experiments in Chapter 3 is based on a simulator of Hearthstone, the rules we describe below are mostly based on Hearthstone; meanwhile, we note that many of them are shared by other CCGs.

A CCG typically has hundreds to thousands of different cards, each of which supports specific in-game rules and effects. When playing CCGs, before each match, every player is asked to build a *deck* comprising of a subset of available cards. While
in game, each player takes turns to draw cards from their respective deck and place them on the game board to combat (e.g., attack, counter-attack, cast spell, etc.) against their opponent cards. By default, a player only draws one card from their deck in each turn; however, certain cards’ in-game effects allow them to draw more than one card. For example, keeping a card in Hearthstone called “Gadgetzan Auctioneer” on the board allows the player draw one card each time he or she casts a spell card.

Cards can be mainly categorized as spells and minions. Spells are played, creating an effect on the battlefield, and then are discarded. Minions, on the other hand, stay in play, can be used to attack the opponent or other minions and endow extra abilities to the player such as drawing more cards from their decks.

In a match, each player is initialized with certain health points. A match terminates by certain criteria, for example, when a player’s health is destroyed. Each player is given a certain amount of mana in each turn. A player can play a card based on its associated rules, e.g., a card can be played when another card is present in hand and the player has sufficient mana to play this card. A player’s turn ends when he or she has no actionable cards in hands or on the board, or voluntarily relinquishes the turn.

A typical game flow of one-vs-one CCGs is shown in Algorithm 1. CCGs are match-based games because any actions in previous matches do not affect upcoming matches. At the beginning of each new match, all players’ state get refreshed, such as health, their decks, and cards in hands and on the board.

**Algorithm 1:** Game flow of one-vs-one CCGs

```
Input : Player1, Player2 with their customized decks

while game not end do
    curPlayer ← alternate between Player1 and Player2
    curPlayer draws cards from his or her own deck
    curPlayer uses the cards in hands and in play to interact with the game
end
```

In general, there is no single deck which can universally win against all other decks. CCGs often design cards with sophisticated synergistic and oppositional relationships. For example, in Hearthstone, there are two distinguished types of decks that counter each other in different phases of a match. An Aggro deck is considered
as an aggressive archetype built with cards capable of dealing damage to the opponents as quickly as possible. In contrast, a control deck is the opposite archetype with cards which can survive long enough to triumph later in the game through powerful but expensive cards or complex combos.

Based on the rules described above, a player’s in-game strategies and strength largely depends on what cards constitute the deck. Therefore, deck building is a crucial step prior to starting a match. Players strive to build winning-effective decks that suit their play style and counter hypothetical opponents. There exist many online forums and websites for players to discuss, analyze and test deck building strategies (e.g., HearthPwn.com, 2017; Icy-Veins.com, 2016). However, deck building has a large and complex solution space, and thus poses great challenges to players. For example, the number of all possible decks in our experiment setting (Section 3.5), which selects 15 out of 312 cards, is $1.4 \times 10^{25}$.

The opponent’s deck is invisible to the player prior or during the match. However we assume when a deck recommendation system recommends a deck to a player, it can access information of both players, including the player and the opponent’s play styles, the opponent’s deck built already. This assumption will be used in Chapter 3, where we propose a deck recommendation system to search for winning-effective decks against specific opponents.

### 2.1.2 Multiplayer Oline Battle Arena (MOBA)

Next, we introduce Multiplayer Online Battle Arena (MOBA) games since in Chapter 4 we will use MOBA games as a test bed for studying winning-effective hero recommendation in a team-vs-team setting.

MOBA is one of the most popular contemporary E-sports games. Games, such as League of Legends (Riot Games) and DOTA 2 (Valve Corporation), have attracted millions of players to play and watch (Minotti, 2015; Tassi, 2016). In a classic match within such games, two teams, each composed of five players, combat in a virtual game map (Figure 2.1). The goal is to beat the opposite team by destroying their base. In order to do that, each player controls an in-game character, known as heroes, and co-operates with other teammates to attack opponents’ heroes, armies,

---

1We follow the terminology of DOTA 2.
Chapter 2. Related Work and Background

Figure 2.1: The game map of DOTA 2. The bases of the two teams are on the ends of the diagonal. Player-controlled characters can move through lanes (Top, Bottom, and Middle) as well as partial areas of jungles. Most MOBA games have similar game maps like this one. Figure adapted from the Internet: https://bit.ly/2LbD3dv

defensive structures, and ultimately base, while defending his own in-game properties. We will refer to in-game characters in MOBA games as heroes, since this is a term widely accepted by the player community.

MOBA games are match-based games (Guo et al., 2012) as defined in Section 1.1, because they are played match by match: a new match starts with random 10 online players and ends whenever a team destroys the opponent’s base. Records within one match do not affect a new match, e.g., every player in a new match starts with level one of his or her character, the same amount of gold, and an empty item inventory.

Heroes are often designed with a variety of physical attributes and skills, e.g. dealing long-distance damage, healing teammates, or spearheading with strong shields, which together add to a team’s overall power. Moreover, there exists sophisticated synergistic and oppositional relationships between heroes. For example, in DOTA 2, hero Clockwerk has high synergy with Naix, because Clockwerk can transport Naix to target enemies directly, making up for Naix’s limited mobility in fighting. In another example, hero Anti-Mage’s mana burn skill reduces an opponent’s mana resource, making him a natural opposition to Medusa, the durability of whom is completely
2.1. Game Background

reliant on how much mana she has. Previous research has also highlighted that interactions between heroes greatly influence match outcomes (Pobiedina et al., 2013; Semenov et al., 2016; Kim et al., 2016), as well as many online discussions about hero pick strategies.²³

The selection of heroes, also known as pick or draft, takes place before each match starts and alternates between two teams until each player has selected one hero. We refer to the 10 heroes in a completed draft as a hero line-up. In a popular match mode named Ranked All Pick, the alternating order of drafting is “1-2-2-2-2-1”, meaning that the first team picks one hero, followed by the second team picking two heroes, then the first team picking two heroes, and so on. The process ends with the second team picking their last hero. During a draft, heroes already selected are visible to both teams. Heroes can only be selected from a fixed pool and no duplication is allowed in the same match. Moreover, a time limit (usually a few dozens seconds) is imposed for each hero pick. Figure 2.2 shows the interface for drafting a DOTA 2 team, where in the central area, images of available heroes are presented for selection. In games like DOTA 2, there are possibly more than 100 heroes that can be picked by a player at the time of drafting. As estimated by Hanke and Chaimowicz²³

²http://www.weskimo.com/a-guide-to-drafting.html
³https://www.reddit.com/r/learndota2/comments/3f9szo/how_to_counter_pick_heroes/
Chapter 2. Related Work and Background

(2017), the number of possible hero line-ups in DOTA 2 is approximately $1.56 \times 10^{16}$.

There are more sophisticated drafting mechanics and rules deployed and interleaved with hero picks in other match modes, such as *banning* (i.e., certain heroes can be prohibited from selection by either team). To make illustration easier and simpler, we assume our system will be deployed under the Ranked All Pick mode, unless otherwise mentioned in Section 4.4.6.

Due to the complex synergistic and oppositional relationships among heroes and the large number of follow-up pick possibilities by other players, as we described above, the hero drafting phase becomes a critical component contributing to match outcomes. Selecting a winning-effective hero is a challenging task to human players especially inexperienced players (Johnson, Nacke, and Wyeth, 2015). Failing to pick heroes that fit teammates’ currently selected heroes and counter opponent heroes may lower players’ confidence and give rise to churn issues (Shores et al., 2014), as well as behavioral issues, such as flaming (i.e., aggressive, hostile, and profanity-laced blame of each other) (Kou and Nardi, 2013), and cyberbullying (Kwak, Blackburn, and Han, 2015). All such issues negatively impact the experience of not only one player, but potentially the whole player community. Therefore, developing better systems that can minimize these issues would be important for such applications.

2.2 Theories of Player Engagement and Competence

The primary motivation of this dissertation is to improve player engagement. A straightforward way to define player engagement is the "continued desire" to play the game repeatedly during the same session or over a longer period of time (Schoenau-Fog, 2011). Player engagement is a complex construct that has been studied extensively in many theoretical frameworks and linked to numerous aspects, such as emotions (Sweetser and Wyeth, 2005; Csikszentmihalyi, 1990; Chen, 2007; Cowley et al., 2008), motivations (Przybylski, Rigby, and Ryan, 2010; Ryan, Rigby, and Przybylski, 2006; Yee, 2006b; Yee, 2006a; Sherry et al., 2006), presence (Lombard and Ditton, 1997; Tamborini and Skalski, 2006), immersion (McMahan, 2003; Brown and Cairns, 2004; Jennett et al., 2008; Ermi and Mäyrä, 2005), pleasure (Costello and Edmonds, 2009), enjoyment (Ravaja, Nacke, and Lindley, 2007; Klimmt, 2003; Mekler et al.,
2.2. Theories of Player Engagement and Competence

One construct of player engagement which has been extensively studied is in-game challenge and competence. Within this construct player competence is defined by enjoyment of victory, accomplishment or dominance over other players while being appropriately challenged (Ryan, Rigby, and Przybylski, 2006; Przybylski, Rigby, and Ryan, 2010; Yee, 2006a; Wu, Wang, and Tsai, 2010; Sherry et al., 2006; Lazzaro, 2004; Schoenau-Fog, 2011). The theories introduced below will provide substantial support that providing proper competence to players will lead to better player engagement, which constitute the theoretical foundation for our work in recommending winning-effective in-game elements.

Self-Determination Theory (SDT) (Ryan and Deci, 2000), a widely researched framework for the study of human motivation, provides support for the importance of competence and its relationship to player engagement. Specifically, SDT defines three basic psychological needs for human motivation: autonomy (the sense of volition or willingness when doing a task) (Deci and Ryan, 2000; Deci and Ryan, 1964), competence (the need for challenge and the feeling of being capable and effective) (White, 1959; Deci and Ryan, 1985), and relatedness (the feel of being connected with others) (Ryan and Deci, 2001; La Guardia et al., 2000). When the psychological needs are satisfied, people develop positive attitude and act effectively and hence experience well-being; however, if they are thwarted, people will more likely conduct negative behavior, such as prejudice and aggression. Ryan, Rigby, and Przybylski (2006) and Przybylski, Rigby, and Ryan (2010) extended SDT and applied it to video games, and found that the three pillars of intrinsic needs: autonomy, competence, and relatedness, could be used to explain why people are engaged playing games.

Flow (Csikszentmihalyi, 1990), as introduced in Chapter 1, is another theory highlighting that competence is an important building block of player engagement because players look for in-game challenge that commensurate with their skills. Flow refers to the optimal mental state of engagement when a person is fully immersed and focused in their activities that he/she loses track of time (Csikszentmihalyi, 1990). The theory identified eight major components which lead to the Flow
state, namely (1) a challenging activity requiring skill, (2) merging of action of awareness, (3) clear goals, (4) immediate, direct feedback, (5) freedom to concentrate on the task, (6) a sense of control, (7) a loss of self-consciousness, and (8) an altered sense of time (Csikszentmihalyi, 1990). Sweetser and Wyeth (2005) adapted the Flow theory to video games and attempted to map elements specific to video games to the eight major components of Flow, from which game design rules required for building engaging games are extracted. According to Sweetser and Wyeth’s mapping, the manifestation of the first component of Flow, “a challenging activity requiring skill” in video games is defined as “challenges in games must match the players’ skill levels” (Sweetser and Wyeth, 2005, p. 5).

Other theoretical works examining motivations underlying video game playing also discussed competition and challenge as important factors. The principle component analysis run by Yee (2006a) revealed three overarching motivation components: Achievement, Social, and Immersion. In this framework, Achievement includes competition referring to the desire to challenge and compete with others. Wu, Wang, and Tsai (2010) verified Yee’s three motivation components in the perspective of Uses and Gratification (U&G) theory (Palmgreen, Wenner, and Rosengren, 1985), which posits that users actively seek out specific media that gratifies their specific needs. Wu, Wang, and Tsai (2010) found that players’ continued motivation is significantly impacted by the perception of gratification (i.e., Achievement, Social, and Immersion). Sherry et al. (2006) identified 6 different gratifications of video game use, including challenge and competition along with social interaction, diversion, fantasy, and arousal. Lazzaro (2004) proposed four key ways where emotions have an active role on engagement: Hard Fun, Easy Fun, Altered States, and the People Factor; Hard Fun refers to game challenges which frequently elicit emotions and experiences such as frustration and Fiero.

2.3 Data-Driven Models for Player Engagement

Player engagement has also been investigated using data-driven approaches, where engagement is mostly measured quantitatively and behaviorally, such as purchases in the game (Xie et al., 2015; Sifa et al., 2015), the number of matches played within
2.3. Data-Driven Models for Player Engagement

a time window (Xue et al., 2017; Weber et al., 2011), or churn risk (Hadiji et al., 2014; Harrison and Roberts, 2012). Churn risk here is defined as the proportion of total players who stopped playing after a specific period of time. These metrics are proposed because they can be easily calculated from collected player data and reflect specific aspects of player engagement practitioners want to examine.

We note some representative data-driven models for player engagement. They share a similar pattern that player engagement modeling is achieved through machine learning models trained to map from player states to engagement metrics. Weber et al. (2011) built regression models to predict the number of games played by a player in a sport simulation game with features describing their game mode preferences, in-game performance and action choices. Hadiji et al. (2014) established a foundation to defining churn prediction in free-to-play (F2P) games. They proposed definitions for various churn behaviors and universal behavioral features, i.e., features that are not game specific, but rather are shared across different F2P games, such as session time and intersession item. They then compared different machine learning models (decision trees, naive Bayes, etc.) across five commercial F2P games; they found that using decision trees could lead to high prediction accuracy. Bertens, Guitart, and Periáñez (2017) proposed a scalable algorithm using conditional inference survival ensembles (Hothorn, Hornik, and Zeileis, 2006) to predict both the level and accumulated time when a player leaves the game based on specific features, which they picked to make sure they are generalizable across games and capture the temporal dynamics of player behaviors, such as days since last purchase and days since last level up. The advantages of their model are the robustness to different data distribution and the ability to make a large volume of churn predictions in real-time due to being able to parallelized.

It should be noted that these prediction models, once trained, can be further interpreted to inform design issues or easily integrated into other systems. Based on their trained regression models, Weber, Mateas, and Jhala (2011) further developed a technique to itemize the important features that can affect engagement prediction, drawing design recommendations that certain in-game controls should be simplified and more clearly presented. Runge et al. (2014) trained a churn prediction model for a casual social game and showed that such a model can be leveraged to decide the
timing for delivering promotion campaigns that could increase the effectiveness of promotions. Churn prediction will also serve as an important building component in our opponent recommendation system proposed in Section 5.

### 2.4 Techniques towards Improving Player Engagement

Here we introduce three kinds of techniques related to our dissertation: Dynamic Difficulty Adjustment (DDA) (Hunicke, 2005), Procedural Content Generation (PCG) (Yannakakis and Togelius, 2011; Togelius et al., 2011) and Recommendation System (Medler, 2011). We will first overview the three kinds of techniques and their relationships, and then introduce each of them in more details.

DDA (Hunicke, 2005) aims to keep players’ perceived competence in the optimal zone to accomplish a state of flow (Csikszentmihalyi, 1990; Sweetser and Wyeth, 2005; Chen, 2007) through manipulating game difficulty adaptively. Such systems are related to our work because recommendations of winning-effective in-game elements, which we study in R.S.Q. 1 may also cause difficulty and player competence may thus change, if players adopt the recommendations. Therefore, in our opinion, techniques we will investigate to answer R.S.Q. 1 are also within the area of DDA.

PCG (Yannakakis and Togelius, 2011; Togelius et al., 2011) refers to methods that automatically (or algorithmically) generate adaptive in-game content. Recommendation systems are used as information filtering tools to present specific in-game elements from a pool of candidates (Medler, 2011). It should be noted that DDA involve various techniques borrowed from both PCG and recommendation areas, and PCG and recommendation systems can be used for various goals including DDA.

#### 2.4.1 Dynamic Difficulty Adjustment

A straightforward way to manipulate game difficulty is *rubber banding*, which means the difficulty of the game rises or falls if the player plays well or poorly. A famous example is from *Mario Kart* series (Nintendo Co., Ltd), in which players who lag behind are more likely to get power-ups to gain speed and catch up to front runners.

Rather than having a monotonous approach to difficulty adjustment for all players, more advanced works leverage player modeling (Yannakakis et al., 2013) as a
tool to identify more granular players’ preferences and deliver more personalized difficulty adaptation (Missura and Gärtner, 2009; Zook and Riedl, 2012). They are inspired by the fact that different types of players desire different scales of difficulty and their skills may vary along time. For example, dealing with the first fact, Missura and Gärtner, 2009 proposed to categorize a player into one of a few player clusters based on a short play trace then adjust difficulty suitable for that particular player cluster. Dealing with the second fact, Zook and Riedl (2012) proposed to use tensor factorization techniques (Kolda and Bader, 2009) to fit player performance data collected historically, which can then be used to forecast player skill mastery in the future.

Another notable kind of advanced models is to formulate difficulty adjustment in a more rigorous probabilistic framework, which enables in-depth analysis of system performance. For example, Missura and Gärtner (2011) formulated adjusting difficulty as an online learning problem (Auer et al., 1995) in which the ranking of finite game difficulty settings is estimated through a trial-and-error fashion with the goal to choose the “just right” difficulty setting as often as possible. The probabilistic framework guides the system to select the proper difficulty setting that is most likely to match the player’s skill (which could even vary temporally) while providing a theoretical bound on the number of mistakes the system would make.

The ways to deliver difficulty adjustment can be as simple as tuning pre-defined parameters which govern environmental changes (e.g., adjusting supply and demand when players are close to death in Hunicke, 2005) or player attributes (e.g., awarding shield and health points to low-performing players in Baldwin, Johnson, and Wyeth, 2014), or it can also be as sophisticated as AI-controlled agents that could adapt to players’ gameplay. The goal of AI-controlled agents for DDA purposes is to select and perform behaviors (actions) leading to small outcome gaps. Back in 2004, Spronck, Sprinkhuizen-Kuyper, and Postma (2004a) equipped dynamic scripting (Spronck, Sprinkhuizen-Kuyper, and Postma, 2004b), an online learning based behavioral script technique, for agents to adapt their behaviors to achieve roughly equal win/loss ratio against opponents. In dynamic scripting, an agent’s behavior is sampled from a rule-based system, where each rule (behavior) is associated with a sampling weight that determines how often this rule should be sampled. Rule
weights are learned in a training phase based on Spronck, Sprinkhuizen-Kuyper, and Postma (2004a)'s proposed techniques in order to scale the intelligence level of the agent to the level of the opponent. Andrade and his colleagues applied Q-Learning (Watkins and Dayan, 1992), one family of Reinforcement Learning algorithms (Sutton and Barto, 1998), to estimate action values in an offline training phase. Then, in the play time after training, every $\tau$ time steps an agent would take the action with its value ranked in 50 percentile of all valid actions, where $\tau$ is a dynamically changing parameter reflecting how rapidly the difficulty needs to be adapted to commensurate with player skill (Andrade et al., 2006; Andrade et al., 2005a; Andrade et al., 2005b). To overcome the fact that the 50th-percentile ranked action might not always be the best action to balance the game and to reduce the manual effort to tune the parameter $\tau$, (Demediuk et al., 2017) equipped AI-controlled agents with Monte Carlo Tree Search (Browne et al., 2012) in another fighting game to fully and automatically determine actions that are promising to lead to the minimal outcome gap.

Several works have conducted experiments and shown that player engagement-related metrics indeed improved due to the use of the proposed methods. In the work by Hunicke (2005), the author adjusted supply and demand in a game and found that expert players reported slightly elevated levels of enjoyment (although novice players did not). In the work by Van Lankveld et al. (2009), players reported decreased pleasure and increased frustration when playing harder games compared to balanced games. Xue et al. (2017) deployed a difficulty adjustment system to a commercial matching-three game (a game like Candy Crush). They discovered that difficulty adjustment could bring as much as $7 \sim 9\%$ improvement in total numbers of rounds played and duration of gameplay. Using a crowdsourcing science game, Sarkar et al. (2017) found that presenting tasks in skill-based difficulty ordering to players led to significantly more attempted and completed levels than random ordering.

2.4.2 Procedural Content Generation

Procedural Content Generation (PCG) is the algorithmic generation of game content with limited or indirect user input (Yannakakis and Togelius, 2011; Togelius et al.,
The goal of applying PCG is not only to eliminate some of the content development burden for developers, but to also adapt game content to satisfy players’ dynamic needs. Togelius et al. (2011) provides a comprehensive taxonomy of different PCG algorithms, such as online-vs-offline (whether game content is generated during the run time or development time of the game), random-seeds-vs-parameter-vectors (generation up to a single seed or a series of specifications), etc.

PCG has been used for various purposes, such as adjusting difficulty and player engagement, depending on what the underlying evaluation function is for assessing generated content. An example of applying PCG for difficulty adjustment is the early work of Jennings-Teats, Smith, and Wardrip-Fruin (2010), in which levels of a platformer game are automatically generated segment by segment, where the next segment was generated with the difficulty level appropriate for how the player performs in the current segment. This relies on a machine learning model (Multilayer Perceptron) which learns the mapping from level specifications to difficulty reported by players themselves. For another platform game, PCG is applied to optimize player engagement (Shaker et al., 2012), with a player engagement prediction model used to guide the generation of levels. The player engagement prediction model, also a Multiplayer Perceptron model, which is based on their former work (Shaker, Yannakakis, and Togelius, 2011), is constructed to predict player’s reported level of engagement using features of level characteristics and players’ playing styles.

Although PCG and recommendation systems are both valid and potential ways to improve player engagement, we focus on the latter in this dissertation. As we introduced in Chapter 1, studying PCG requires access to some test-bed games for modification of game content, which was not available to me during my research. Therefore, we primarily study recommendation systems which rely on collected data and simulators.

### 2.4.3 Recommendation Systems

Recommendation systems have been studied for a long time in a variety of areas such as movies (Amatriain and Basilico, 2012), e-commerce (Linden, Smith, and York, 2003) and news (Das et al., 2007). The primary goal of these recommendation systems is to serve as information filtering tools to ease information overload,
retrieve the most relevant information, and provide personalized services (Isinkaye, Folajimi, and Ojokoh, 2015; Bobadilla et al., 2013; Resnick and Varian, 1997; Adomavicius and Tuzhilin, 2005); without such systems, users would be confronted with millions of items with no guidance on how to search such a large space. These recommendation systems predict users’ preferences on items accurately such that personalized item recommendations can be given based on the ranking of user preference predictions (Liang, Lai, and Ku, 2006).

The video game industry has also used recommendation systems to introduce players to games that they are likely to enjoy (Sifa, Bauckhage, and Drachen, 2014; Orland, 2010; Skocir et al., 2012; Wu et al., 2017). These recommendation systems, like their counterparts in other web-based services, alleviate information overload faced by users, as users are not able to go through all candidate games as this is usually a prohibitively large list.

Video games as a form of interactive entertainment allow game developers to apply recommendation systems not only for recommending next games to play, but also to recommend in-game elements for a particular game. In this dissertation, we focus on recommendation systems for in-game elements within match-based video games which have features different from traditional recommendation systems introduced in the previous paragraph. First, our goal is focused on predicting in-game elements’ influences on winning-effectiveness and engagement, which may be manifested by metrics like winning probabilities, rather than players’ preferences. Second, in-game element recommendations to one player may affect other players’ experience. Therefore, in-game element recommendations have to consider other players. This is different from traditional recommendation systems where recommended items to one user are relatively independent from other users’ experience.

In-game element recommendation systems sharing the same scope as this dissertation have only sporadic appearances in literature (i.e., in-game element recommendations in the pre-match stage in match-based games for influencing player competence and engagement). Our suspicion for the lack of research in this area is that access to data from large-scale commercial games, which have sufficiently sophisticated in-game elements to apply recommendation system techniques to, is hard to come by. The difficulty may drive academic research to focus only on a few publicly
accessibility through accessible datasets or games with a smaller scale of which researchers can have full access and control. Nevertheless, we survey existing works as follows, with each subsection corresponding to recommendation systems that cope with a specific type of in-game element we will investigate in Chapter 3, 4, and 5, respectively.

Starting Item Recommendation

Starting items are commonly seen in-game elements in a variety of genres of games, such as Action Role-Playing Game (Action-RPG) and Multi-Player Online Arena (MOBA), where starting items aid in-game characters heal, scout, defend, and attack when characters are weak in their initial stages. In Collectible Card Games (CCGs) starting items are a set of cards, called a *deck* (See CCG background in Section 2.1.1). Some games have incorporated starting item recommendation features; for instance, when making purchases of starting items in League of Legends, players can browse the full list of all items or a short list of recommended items for easier choices (Goslin, 2017). However, to the best of our knowledge, the underlying recommendation techniques in commercial games are largely undisclosed, probably due to proprietary reasons.

Despite the prevalence of starting items in video games, to the best of our knowledge, we have only seen little academic research in starting item recommendation. Specifically in the problem of deck recommendation for one-vs-one CCGs, which we discuss in Chapter 3, we have only seen search-based solutions using either heuristic search or metaheuristic search (Birattari and Kacprzyk, 2009).

Heuristic search methods suggest which cards to include in a deck based on domain heuristics such as popularity and in-game resource curve (Karsten, 2015; Fancher, 2015; Stiegler et al., 2016). However, they require in-depth human knowledge and lack flexibility to adapt to different play styles and opponents. Metaheuristic searches rely on high-level, problem-independent, approximate search strategies for tackling optimization problems (Birattari and Kacprzyk, 2009). Researchers have used one type of metaheuristic search called Genetic Algorithm (GA) (Holland, 1992) to evolve decks towards higher winning-effectiveness through repeated modifications and selections (García-Sánchez et al., 2016; Bjørke and Fludal, 2017). In GA (Holland, 1992), candidate solutions (called *individuals*) evolve towards better
feasible solutions iteratively with mutation and crossover operators. In each generation, the fitness value of every candidate solution is evaluated. The fitness value is usually the value of the objective function in the optimization problem being solved; in the works of García-Sánchez et al. (2016) and Bjørke and Fludal (2017), the fitness value is the average win rate of a candidate deck against a group of opponent decks while AI bots are used as a proxy for human play. The fit candidate solutions are stochastically selected and modified to form a new generation. Although not requiring human knowledge to guide searches, metaheuristic search algorithms incur a large computational cost for simulation-based evaluation on intermediate solutions, which renders them unsuitable for large-scale or real-time usage. In Section 3.3, we will analyze in details why existing approaches are inefficient.

Character Recommendation

The rich design of characters and recent worldwide popularity of MOBA games have allowed them to become the primary testbed for character recommendation research. Classic MOBA games, like League of Legends and DOTA 2, are usually played 5-vs-5; there are possibly more than 100 characters that can be picked by a player in the pre-match stage. Moreover, as per rules of certain match modes players are supposed to select characters in sequence; players need to consider synergistic and oppositional relationships among champions because a winning-effective character should not only fit to selected characters so far but also projected characters selections by the rest of players.

To recommend winning-effective characters in the character selection phase, Hanke and Chaimowicz (2017) proposed to mine association rules (Agrawal and Srikant, 1994) from historical character selection data and use them as the heuristic to recommend heroes. Here, association rules are character subsets that appear frequently together either in the winning team or in opposite teams. Any character contained in the discovered association rules together with characters picked already is suggested to be a good candidate to pick next. However, this method does not consider which characters will be picked by other players in the rest of the drafting process, hence this is essentially a myopic, greedy-based approach.
Researchers have also proposed to recommend characters based on player selection tendency. Summerville, Cook, and Steenhuisen (2016) models character recommendation as a sequence prediction problem. They train a sequence prediction model for predicting the next character most likely to be selected based on historical character selection sequences. However, the predicted character is “what is likely to be picked, not what is necessarily best” (Summerville, Cook, and Steenhuisen, 2016). Therefore, character recommendation based on such a method may not be an optimally winning character for team victory.

Although there are other works predicting match outcomes (Yang, Brent, and Roberts, 2014; Semenov et al., 2016; Makarov et al., 2017) and analyzing in-game traces (Cavadenti et al., 2016), they do not focus on how to utilize these models for character recommendation in the context of sequential character selection.

**Matchmaking (Opponent Recommendation)**

In practice, the concept of opponent recommendation is often implemented as matchmaking services, which connect players to form matches. As such, players themselves become the subjects of recommendations.

A fair amount of matchmaking systems simply assume that skill balanced games are good for engagement (Graepel and Herbrich, 2006; Sweetser and Wyeth, 2005; Csikszentmihalyi, 1990; Chen, 2007) and hence resort to skill rating algorithms (Glickman, 1999; Elo, 1978; Herbrich, Minka, and Graepel, 2006) to identify similarly skilled opponents. Myślak and Deja (2014) suggests additional information about player preferences in in-game avatar roles can further improve fairness-based matchmaking systems. A few researchers have explored methods to improve player engagement through matchmaking. Delalleau et al. (2012) proposed to train a neural network based architecture which predicts player enjoyment based on their historical statistics. They measured enjoyment by directly asking players for feedback after each match. However, their enjoyment-based matchmaking has not been verified in real games. Plus, whether players are willing to give feedback about enjoyment and how reliable their feedback would be are questionable. Jiménez-Rodriguez, Jiménez-Díaz, and Díaz-Agudo (2011) proposed that matchmaking could be based on preferred roles by players. They argue that a fun match should have players act in roles
with perceivably joyful role distribution. However, it is still a conceptual, heuristic-based method without experiment showing that such matchmaking system indeed improves concrete engagement metrics. To our best knowledge, we have not seen any existing opponent recommendation method that formally treats the opponent recommendation task as an optimization problem to maximize player engagement.
Chapter 3

Deck Recommendation in Collectible Card Games

3.1 Introduction

In this chapter, we investigate the development of a recommendation system for answering R.S.Q. 1 in the one-vs-one setting, which is reiterated here:

How can we design systems working in the pre-match stage that can recommend winning-effective in-game elements, within the settings of one-vs-one and team-vs-team?

(R.S.Q. 1)

As we stated in Chapter 1, our motivation is to increase player competence through helping players make more winning-effective choices of in-game elements. The challenge here is that most modern video games offer a large number of in-game element choices, which makes it hard to select favorable ones. After surveying a few game genres, we find that recommendations of winning-effective starting items in one-vs-one Collectible Card Games (CCGs) is a sufficiently complicated example for the aforementioned research question. While readers can refer to Section 2.1.1 for a detailed background of CCGs, we give a brief description here for readers’ convenience.

One-vs-one CCGs are match-based card games, where two players wield cards with various in-game effects to attack opponents and defend themselves. In a match, a player’s cards are drawn from his or her own deck - a deck is a collection of cards
prepared before the match starts which represents specific play strategies a player may want to implement. CCGs are often designed with an abundant number of cards (a few hundreds to thousands), but the size limit of a deck is usually much smaller (a few dozens). Supposing the number of total cards and deck size are $N$ and $D$, respectively, there could be as many as $\binom{N}{D} = \frac{N!}{D!(N-D)!} = O(N^D)$ ways to assemble decks. Besides, due to sophisticated synergistic and oppositional relationships among cards, there is no single deck which can universally win against all others. All these factors make the task of identifying winning effective decks very challenging for players.

In this chapter, we proposed a recommendation system, named Q-DeckRec, able to efficiently identify winning-effective decks for players in such a setting. The chapter is structured as follows. We first introduce the formal problem formulation and then analyze how existing methods are inefficient to solve the problem. Next, we illustrate our proposed algorithm. Finally, we show experiments and results to compare our method with others.

### 3.2 Problem Formulation

We treated the problem of recommending a winning-effective deck against an opponent as to approximately identify the winning-optimal deck against that opponent. The winning-optimal deck is defined as the one with the highest expected win rate against the opponent. Each of the player (the recommendee) and the opponent is abstracted to a specific play style and deck. On a high level, the problem is to output the approximated winning-optimal deck for the player given both players’ play styles and the opponent’s deck.

As a deck is a combination of cards, identifying the winning-optimal deck can be formulated as a combinatorial optimization problem (COP). Let us briefly review the definition of COP. Generally, an optimization problem consists of an objective function and a set of problem instances. Each problem instance is defined by a set of variables and a set of constraints among those variables. A candidate solution to a problem instance is an assignment of values to the variables. A feasible solution is a candidate solution that satisfies the set of constraints. An optimal solution is a feasible solution
that maximizes the value of the objective function. A COP is an optimization problem whose problem instances have finite numbers of candidate solutions. For many realistic COPs, such as the traveling salesperson problem (TSP), the number of candidate solutions is too large to exhaust in order to find an optimal solution. This is exactly where our problem lies. In other words, the problem can be formulated as searching for the winning-optimal deck to achieve the highest expected win rate among a large, but finite number of deck candidates.

Following the previous notations, suppose a deck is a subset of cards of size $D$ among a total of $N$ cards, with $N > D$ because $N$ is usually several times larger than $D$. A deck can be represented as a binary vector of length $N$, $x \in \mathbb{Z}_2^N$, whose components of 1’s correspond to the cards included in the deck and 0’s otherwise. Since a deck has a fixed size of cards, we have $\|x\|_1 = D$. We use $x_p$ and $x_o$ to differentiate the deck of the player and his opponent.

We use $A_p$ and $A_o$ to capture the play styles of the player and his or her opponent. $A_p$ and $A_o$ are play style-specific simulators that decide which cards to issue given a game instance, henceforth referred to as the AI proxies (artificial intelligence) of respective play styles. The evaluation function $f(\cdot)$ is defined as $f(x_p; x_o, A_p, A_o)$, which returns the winning probability of the player using $x_p$ against the opponent using $x_o$, with their play styles following $A_p$ and $A_o$ respectively. Note that $f(\cdot)$ is a black-box function. We do not have the closed-form expression of $f(\cdot)$, but can approximate its value by simulating $A_p$ and $A_o$ playing against each other for several matches.

The objective function of finding the winning-optimal deck is formulated as:

$$\arg \max_{x_p} f(x_p; x_o, A_p, A_o),$$

subject to $x_p \in \mathbb{Z}_2^N, x_o \in \mathbb{Z}_2^N,$

$$\|x_p\|_1 = \|x_o\|_1 = D \quad \text{(3.1)}$$

the solution of which is denoted as $x_p^*$. In practice, the brute-force approach is infeasible to solve this problem, as we need to evaluate an $O(N^D)$ number of $x_p$ configurations, while a sufficient number of matches need to be simulated to get a stable win rate estimation for each call.
of $f(\cdot)$. Since the simulation needs to apply numerous rules of the game on each move, this is a computationally demanding operation. In our experimental setting in Section 3.5, where $N = 312$, $D = 15$, we would need to exhaust around $1.4 \times 10^{25}$ possibilities if using brute-force. Each evaluation of $f(\cdot)$ was found to take non-negligible time in the order of seconds even on a very powerful server.

### 3.3 Analysis of Existing Methods

As we surveyed in Section 2.4.3, previous works for approximately solving Eqn. 3.1 are mainly search algorithms, which are divided into two categories: heuristic searches and metaheuristic searches. We will analyze below that the former might not be effective and the latter might not be efficient enough to be deployed as a winning-effective deck recommendation system.

Heuristic search methods decide which cards to include based on domain heuristics but requires intensive human knowledge and time to code heuristics considering various factors, such as the large number of cards, the requirement of finding a deck that is synergistic and oppositional, as well as accounting for different play styles. Furthermore, heuristics designed under limited manual resources may not be effective for different opponents since they may only encode some of the card relationships and play styles - in CCGs subtle difference in just a few cards or play styles could lead to decks of very different strengths. Therefore, human generated heuristics are often hard to encode given the complexity of the game rules. Thus, there is a need for an automated and intelligent recommendation system that uses minimal labor, but can search such a large space efficiently.

Metaheuristic searches refer to high-level, problem-independent, approximate search strategies for tackling optimization problems (Birattari and Kacprzyk, 2009). In fact, it is not new to approximately solve COPs through metaheuristic searches. An example that we surveyed in Section 2.4.3 used a Genetic Algorithm (GA) (Holland, 1992) to evolve decks towards higher winning-effectiveness through repeated modifications and selections (García-Sánchez et al., 2016; Bjørke and Fludal, 2017). Although researchers have applied other kinds of metaheuristic searches on a variety of COPs, such as the Cross-Entropy (CE) method (Rubinstein, 1999), tabu search (Glover,
3.3. Analysis of Existing Methods

1986), and simulated annealing (Kirkpatrick, Gelatt, and Vecchi, 1983), GA has been the only one applied on the same COP as defined in Eqn. 3.1, i.e., to approximately identify the winning-optimal deck.

While metaheuristic search algorithms do not require human knowledge to guide searches, we find an inherent disadvantage to using them, which makes them inefficient for a deck recommendation system. They are non-learning search algorithms, in the sense that each time a new problem instance arises they require many objective function evaluations until a sufficiently high-quality feasible solution is found. As we noted in Section 3.2 and will show in the Experiments Section 3.5, the evaluation of the objective function (the simulation-based win rate estimation function $f(\cdot)$) is computationally expensive, because it requires a large number of simulated matches with complicated in-game rules. We note from the previous works of García-Sánchez et al. (2016) and Bjørke and Fludal (2017) that solving a single problem instance using GA often takes hours or days to reach a winning-effective deck. This means metaheuristic searches are not suitable for large-scale or real-time application of winning-effective deck recommendation, such as an online CCG’s backend or a deck analysis website which serves a population of online players, because they would cause a heavy burden on computational resources and are intractable.

Additionally, researchers have also attempted to use supervised learning models to learn the mapping from problem instances (features) to optimal or approximately optimal solutions (labels) (Vinyals, Fortunato, and Jaitly, 2015). This approach has been applied in other COPs, but not to the problem of identifying winning-optimal decks. Optimal or approximated optimal solutions need to be calculated by some solver in advance in order to provide labels for supervised learning. One potential difficulty is to design special model architectures to cope with the discrete nature and constraints of COPs. For instance, in the TSP, the outputs should be constrained to sequences with no duplicated cities (Vinyals, Fortunato, and Jaitly, 2015). For our problem, it may be natural to think of using a multi-label classifier (Tsoumakas and Katakis, 2007), in which the features are samples of opponent decks and the labels correspond to approximately winning-optimal decks obtained by existing methods, e.g., Genetic Algorithm. Still, there are several technical issues worth considering: (1) how to constrain the size of labels to match the size of a deck; (2) how to sample
training data; (3) whether to further model dependencies among labels (Zhang and Zhang, 2010), because of the existence of complicated card relationships. Thinking thoroughly about these technical issues is non-trivial, hence we do not consider using supervised learning models for our problem in this chapter. Future work can explore the difference between our approach presented here and the use of supervised learning.

3.4 Methodology

In this section we present our proposed solution, Q-DeckRec, which is expected to solve Eqn. 3.1 more efficiently than the heuristic search and metaheuristic search methods that we analyze in Section 3.3.

Our method is inspired by a fundamental assumption that will be made and exploited: in practice, players’ play styles come from a pool of AI proxies pre-trained by a deck recommendation system. This is a reasonable assumption because a deck recommendation system may not have sufficient data and resource to approximate every individual player’s play style; rather, it is more tractable to cluster a player’s play behavior into one of several pre-trained AI proxies. For example, each AI proxy from the pool represents a specific play style archetype such as “aggressive” or “conservative”. Under this assumption, each problem instance consists of $x_o$, which may vary, and $A_p$ and $A_o$, which are available. Therefore, there may exist deck building patterns which can be generalized for each pair of AI proxies in an offline stage. For example, if certain $A_p$ is good at using Card A to counter certain $A_o$, then Card A tends to appear in the optimal solution of many problem instances with the two AI proxies as the input. In the rest of the chapter, we assume that we deal with problem instances of Eqn. 3.1 under a specific pair of $A_p$ and $A_o$. Our method will be invariantly applied for other pairs of AI proxies.

In the previous works based on GA (García-Sánchez et al., 2016; Bjørke and Fludal, 2017), the search process is independent among different problem instances, with no information generalized for future problem instances to reduce and avoid the evaluation of the objective function. Since now we assume there could be generalizable deck building patterns under a pair of $A_p$ and $A_o$, it is natural for us to look
for models that can encode those patterns and facilitate the solving process of future problem instances.

Our inspiration above aligns with the algorithms which learn search policies for solving optimization problem instances (Zoph and Le, 2016; Li and Malik, 2017; Chen et al., 2016a; Zhang and Dietterich, 2000; Bello et al., 2016). These algorithms lie in a broader area known as “meta-learning” (Lemke, Budka, and Gabrys, 2015; Brazdil et al., 2008; Vilalta and Drissi, 2002) or “learning to learn” (Thrun and Pratt, 2012). Learning of search policies relies on viewing the optimization process as conducting sequential decision making (Littman, 1996) by an optimizer agent. The optimizer agent starts in some initial state and consecutively applies operators to move to new states. The goal is to end at a final state where a high-quality feasible solution can be extracted. We use the mapping between states and operator choices as the search policy.

If we additionally define a transition function and a reward function, we can formulate the optimization process as a Markov Decision Process (MDP) (Bellman, 1957). The optimal policy that maximizes long-term rewards can be learned or approximated by leveraging reinforcement learning (RL) algorithms (Sutton and Barto, 1998). The key is to properly design the MDP, especially the reward function, such that the learned policy can guide the optimizer agent quickly towards high-quality feasible solutions. For example, Zhang and Dietterich (2000) applied an RL algorithm TD(λ) to obtain the search policy for solving NASA space shuttle scheduling problem instances. Their results show that the learned search policy is more effective in the ratio of solution quality vs. CPU time than the best known non-learning search algorithm on test problem instances. Bello et al. (2016) show that a search policy parameterized as a special structure of neural network can be trained and used to solve unseen instances of TSP.

Inspired by meta-learning algorithms, we proposed to delegate solving the COP of Eqn. 3.1 as a problem of generalizing a search policy in an MDP environment, where an agent navigates in the state space to search for the winning-optimal deck and its search policy generalizes deck building patterns under a pair of AI proxies $A_p$ and $A_o$.

In the MDP, a state $s \in S$ consists of a unique feasible solution $x_p$, together with
$x$, and a step counter $t$ as complement information, i.e., $s = \{x_p, x_o, t\}$. An action $a \in \mathcal{A}$ is defined as a card replacement to modify the current deck $x_p$. An action replaces exactly one card in the deck $x_p$ with another card not included in $x_p$. One special action is to keep the current deck as unmodified. Given the actions we define, the transitions between states $T: \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ are always deterministic. One state applied by an action will transit to only one next state, reflecting the corresponding card modification, denoted as $\{x_p(t), x_o, t\}, a \rightarrow \{x_p(t+1), x_o, t+1\}$. The deck search starts from a random initial state $s_0 = \{x_p(0), x_o, 0\}$ and is limited to take exactly $D$ actions in one episode. We denote the states within one episode as $s_0, s_1, \cdots, s_D$. We limit the length of the horizon to be $D$ because at most we need to replace all the cards in $x_p(0)$ to reach the optimal deck $x_p^*$.

The problem remains: how to design the reward function $R: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. In the MDP, the optimal policy is the one which maximizes a defined long-term reward criterion. The key is to properly design the reward function and long-term reward criterion, such that the optimal policy is indeed the desired search policy which can lead to winning-effective decks from any state.

The long-term reward criterion defines the goal of reinforcement learning. It should encourage the optimal policy to search in the direction of winning-effective decks. We proposed the following long-term reward criterion for each episode:

$$R = \sum_{t=0}^{D-1} r_t,$$  \hspace{1cm} (3.2)

where $r_t$ is the reward function over each transition. Specifically, we defined $r_t$ as the win rate between the opponent deck and the modified deck after step $t$ with exponential amplification:

$$r_t = \exp(b \cdot f(x_p(t+1); x_o, A_p, A_o)),$$  \hspace{1cm} (3.3)

where $b$ is a positive constant to adjust the extent of amplification. We chose this reward function over $r_t = f(x_p(t+1); x_o, A_p, A_o)$ in order to amplify the difference between strong and weak decks. Although the goal of deck building is to land on $s_D = \{x_p(D), x_o, D\}$ with $r_D = f(x_p(D); x_o, A_p, A_o)$ as high as possible, the cumulative
3.4. Methodology

The sum of win rates as in Eqn. 3.2 provides more reward signals along the entire search trajectory than merely optimizing $R = r_D$. For similar reasons, the form of cumulative sum in reward functions has also been adopted in other optimization problems based on sequential decision making (Andrychowicz et al., 2016; Chen et al., 2016a). Since we modeled each episode with finite horizons, we ignored the conventional reward discount factor $\gamma$ in the definition of $R$, which is a mathematical trick to help the convergence of RL learning in MDPs with infinite horizons.

The optimal policy can be obtained by always selecting the action with the highest optimal state-action value at each state:

$$\pi^*(s) = \arg \max_a Q^*(s, a), s = s_0, \ldots, s_{D-1},$$  \hspace{1cm} (3.4)

where $Q^*(s, a)$ is defined as the best state-action value function among all possible policies:

$$Q^\pi(s, a) = \mathbb{E}\left[\sum_{i=t}^{D} r_i | s_t = s, a_t = a, \pi\right]$$  \hspace{1cm} (3.5)

$$Q^*(s, a) = \max_{\pi} Q^\pi(s, a)$$  \hspace{1cm} (3.6)

The larger $Q^*(s, a)$ is, the larger the upper bound of the cumulative rewards the agent would collect until the end of the episode. Intuitively, $Q^*(s, a)$ is a proxy of how winning-effective the final deck $x_p^{(D)}$ could maximally be after applying the modification $a$ on the current deck encoded in $s$. Therefore, following $\pi^*(s)$ generates a series of modifications that faithfully build the winning-optimal deck.

We proposed to use a Reinforcement Learning (RL) algorithm, Q-Learning (Watkins and Dayan, 1992), to learn $Q^*(s, a)$ iteratively through observation tuples $(s, a, r, s')$. The simplest implementation of Q-Learning is a look-up table and a learning rate $0 < \alpha \leq 1$, with the update rule as:

$$\hat{Q}(s, a) = (1 - \alpha)\hat{Q}(s, a) + \alpha(r + \max_{a'} \hat{Q}(s', a'))$$  \hspace{1cm} (3.7)

Theory implies that if each action is tried in each state an infinite number of
times and the magnitude of $\alpha$ meets certain criteria, then $\hat{Q}$ converges to $Q^*$ (Bertsekas and Tsitsiklis, 1989). However, our problem has a huge state space, hence it is not possible to maintain a look-up table for all combinations of states and actions. Instead, we resort to Multi-Layer Perceptron (MLP) with parameters $\theta$ as a function approximator: $Q_\theta(s, a)$ learns to approximate the mapping of the feature representation of the state-action pair, $F(s, a)$, to the optimal state-action value, $Q^*(s, a)$. More specifically, we use an MLP architecture with one input layer, one hidden layer and one output layer. Without requiring any prior domain knowledge, we simply let $F(s, a) = s'$. Therefore, the input layer takes as input a state representation $s'$, which has $2 \cdot N + 1$ dimensions. The output layer outputs a real value representing the predicted $Q^*(s, a)$. The exact specifications can be seen in Section 3.5. The update rule of $\theta$ is in a gradient descent fashion towards reducing so-called TD-error $\delta$:

$$\delta := r + \max_{a'} Q_\theta(s', a') - Q_\theta(s, a) \tag{3.8}$$

$$\theta \leftarrow \theta + \alpha \cdot \delta \cdot \nabla Q_\theta(s, a) \tag{3.9}$$

To learn $\theta$, we need to collect observation tuples $(s, a, r, s')$ through solving “training” problem instances. Solving a training problem instance is to let Q-DeckRec take actions $D$ times based on the current $Q_\theta(s, a)$ function in an episode. In order to generalize $Q_\theta(s, a)$ to various states, we initialize both $x_o$ and $x_p^{(0)}$ in $s_0$ randomly at the beginning of each episode. An $\epsilon$-greedy policy is used during the training, with $\epsilon$ slowly decreasing as the learning proceeds. The policy has $\epsilon$ probability to choose non-optimal actions in the hope to escape any local optimum and discover better policies.

Also, we used prioritized experience replay (Schaul et al., 2015) to improve sample efficiency. Past experiences (i.e., observation tuples) are stored and weighted according to the absolute value of $\delta$. High TD-error associated experiences will be more likely to be sampled for MLP parameter learning. High TD-error associated
3.5 Experiment Setup

experiences are those “surprising” or unexpected observations that the current values of the MLP parameters could explain less well. Sampling them frequently helps the learning process focus on the hardest samples. Prioritized experience replay has been shown to speed up parameter learning in several classic RL benchmarks compared to uniform experience sampling methods (Schaul et al., 2015).

The training phase of Q-DeckRec can be summarized as follows. At the beginning of each training episode, both $x_0$ and $x_p^{(0)}$ are randomly generated. Q-DeckRec decides how to "navigate" through states by $\epsilon$-greedy policy and $Q_\theta(s, a)$ in $D$ steps. All the $D$ transitions are stored into the prioritized experience replay pool. Following that, $m$ previous observation tuples $(s, a, r, s')$ are sampled from the prioritized experience replay as a learning batch for updating $\theta$ as described in Eqn. 3.8 and 3.9. The loop continues after a new training episode is initiated. The training is terminated after a time limit is reached.

After training, $Q_\theta(s, a)$ is fixed. When solving a future problem instance, Q-DeckRec can start from $s_0$ with a random $x_p^{(0)}$ and follows $\pi^*$ as in Eqn. 3.4 in $D$ steps. No call of $f(\cdot)$ is needed during the search. As a comparison, non-learning search algorithms such as Genetic Algorithm require calling $f(\cdot)$ multiple times in order to evaluate fitness values for each problem instance (García-Sánchez et al., 2016; Bjørke and Fludal, 2017), while calling $f(\cdot)$ would take computational resources much heavier than calculating $Q_\theta(s, a)$. Therefore, Q-DeckRec has its superior suitability for large-scale or real-time application.

3.5 Experiment Setup

To test our method and compare it with other methods, we experimented on an open-sourced CCG simulator MetaStone\textsuperscript{1}, which is based on the popular online digital CCG Hearthstone (Blizzard Entertainment, Inc.). All experiments ran on a powerful server with Intel E5 2680 CPU’s @ 2.40 GHz (56 logical CPU cores). Parallelization was implemented in three places: (1) linear algebra operations used in the MLP in Q-DeckRec; (2) match simulations evenly spread on all cores when evaluating $f(\cdot)$;

\textsuperscript{1}https://github.com/demilich1/metastone
(3) random deck sampling from a baseline based on Monte Carlo simulations (introduced later). Each call of \( f(\cdot) \) returns a win rate based on 300 simulated matches, which on average takes 5 seconds and has around 5% standard deviation in the win rate evaluation.

We made a few decisions in setting up our experiments. First, we used the same AI proxy to represent both \( A_p \) and \( A_o \). The AI proxy, provided by the simulator and called \textit{GreedyOptimizeMove}, is a model-based method which decides the best action by evaluating each action’s consequence according to a heuristic. Choosing the same AI proxy for both players is just for simplicity purposes and we expect that the superior efficiency of Q-DeckRec to other methods, which is to be shown in our experiments, can generalize to other pairs of AI proxies. As a side note, we did not choose to use other AI proxies based on search-based methods (Santos, Santos, and Melo, 2017) because they take much longer to complete the simulation of one match. However, there have been attempts to train model-based AI proxies, which can play comparably with search-based methods but with faster speed (Janusz, Tajmajer, and Świechowski, 2017). Second, we assumed that both players are from a specific in-game character class called \textit{Warriors}. The total number of available cards to Warriors is 312. Third, while in the real game certain cards can have at most one copy and all other cards can have at most two copies in the deck, we impose that every included card has two copies. This reduces our search space size for testing purpose, and also follows the postulation that having two copies for every card makes the deck performance more reliable (García-Sánchez et al., 2016; García-Sánchez et al., 2018). As a result, although the deck size is 30, the number of cards to be selected is 15. In summary, we have \( N = 312, D = 15 \) when optimizing Eqn. 3.1. Different CCGs vary in terms of the deck sizes and total number of card. Furthermore, there exists CCGs with larger search space than this setting. We plan to test more combinations of \( N \) and \( D \) in the future.

We set up Q-DeckRec as follows. The underlying MLP has one hidden layer and one output layer. The hidden layer consists of 1000 rectified linear units (ReLU). The output layer is a single unit which outputs a weighted sum from the activation values of the hidden layer. \( \epsilon \) in the \( \epsilon \)-greedy policy starts at 1 and decreases 0.0005 per training episode until it reaches 0.2. The size of a learning batch, \( m \), is set at
For the prioritized experience replay (Schaul et al., 2015), the exponent $\alpha$ is set at 0.6, the exponent $\beta$ is linearly annealed from $\beta_0 = 0$ to 1 with step $1e^{-5}$. The capacity of the experience pool is 100K. The constant $b$ in the reward function is set as 10. All the hyperparameters are chosen empirically without fine-tuning due to large computational resources required.

We compared Q-DeckRec with a Genetic Algorithm (GA), the method used in previous works for solving similar problems (García-Sánchez et al., 2016; Bjørke and Fludal, 2017). We implement GA with an open source library DEAP$^2$. An individual is a candidate deck $x_p$. The fitness value is $f(x_p; x_o, A_p, A_o)$. The mutation and crossover functions are customized to maintain the validity of individuals, similar to what was adopted in Bjørke and Fludal (2017). Specifically, mutation is swapping one card in the deck with one not in the deck and crossover randomly exchanges cards not overlapped by the two decks. The population size of each generation is 10, with the mutation probability and the crossover probability both set as 0.2. Individual selection is based on a commonly used selection mechanism called tournament of size 3.

We also designed an ad-hoc baseline which, like Q-DeckRec, requires a learning phase and does not require calling $f(\cdot)$ for solving future problem instances. The baseline conducted Monte Carlo (MC) simulations using a win rate predictor $\hat{f}(\cdot)$ to locate a solution. We first trained a supervised learning model to approximate $f(\cdot)$. The training data are randomly generated pairs of decks represented as binary vectors. The labels are the evaluated win rates based on $f(\cdot)$. We chose to train an MLP with the same architecture as in Q-DeckRec. Given the same input, $\hat{f}(\cdot)$ would output faster than $f(\cdot)$ because the former does not need a real match simulation. When solving a future problem instance with opponent deck $x_o$, we ran MC simulations according to:

$$\arg\max_{x_p \in X_p} \hat{f}(x_p; x_o; A_p, A_o),$$

(3.10)

$^2$https://github.com/DEAP/deap
where $X_p$ is a set of randomly generated decks. We denote the size of $X_p$ as $X$. A larger $X$ means more thorough sampling.

In these experiments, there are several methods that we did not include or compare with, but instead we left that to future work. Specifically, we did not include any heuristic search method, because we focused on algorithmic deck recommendation systems requiring minimal human knowledge involved. Besides GA, we did not include other metaheuristic search methods; similar to GA, they all require calling the win rate evaluation function $f(\cdot)$ several times while solving each problem instance. We did not include supervised learning models, which directly learns the mapping from problem instances to optimal solutions, because to do so requires designing a specific model architecture to cope with the characteristics of the deck recommendation problem (e.g., outputs are constrained to contain $K$ cards), which has not been studied before and requires non-trivial extra works.

Different wall time (i.e., real elapsed time) limits are imposed as the termination condition for both Q-DeckRec training and GA solving one problem instance. In this way, we can compare how long Q-DeckRec training and a GA run would take to reach similar performances. Wall time limits were chosen empirically based on observations in preliminary experiments and our limited computational resources. For GA, we tried wall time limits as 10, 15, 20 and 25 minutes because performances often plateau after 20 minutes, as evidenced in the result section (Section 3.6). Note that the time limit for GA is in the order of minutes because GA takes hundreds of objective function evaluations during evolution for each problem instance before the fitness function plateaus (to be shown in Table 3.1) and each call of $f(\cdot)$ takes around 5 seconds as we tested. Since we did not have the optimal solutions for test problem instances, we reference the solutions from 25-minute GA searches as approximated ground truths. For Q-DeckRec training, we tested one, two and three days as the wall time limit. As will also be shown in Section 3.6, Q-DeckRec after a three-day training period can already reach the same optimal solution reached by GA with 25 minute limit for solving one problem instance.

For the MC-simulation method, we used a training data set collected in three days and test $X = 67, 670, 6.7K, 67K, 670K$ and $6700K$. Note that $67K$ is around the same number Q-DeckRec calls its learned function approximator $Q_\theta(s,a)$ for solving
3.5. Experiment Setup

a test problem instance\(^3\) whereas higher values of \(X\) than 6700K would require too large computational resources to be practical for large-scale or real-time application.

In the rest of this chapter, we will label the algorithms we discussed based on the specific approach, such as \(GA\), \(Q\text{-DeckRec}\), or \(MC\), plus an associated parameter. For example, \(GA_{20\min}\) denotes Genetic Algorithm with 20-minute limit for solving one problem instance; \(MC_{670K}\) denotes the Monte Carlo simulation-based method with 670K randomly generated decks; \(Q\text{-DeckRec}_{1\day}\) and \(Q\text{-DeckRec}_{2\days}\) are two different algorithms denoting \(Q\text{-DeckRec}\) with training time as 2 days and 3 days, respectively.

To evaluate and compare all algorithms, we generated 20 test problem instances as follows. In our preliminary experiments where test problem instances were randomly generated, we found GA often only needed less than 100 calls of \(f(\cdot)\) to identify decks with 100% win rate. This is because randomly generated \(x_o\) barely had any effective card synergy and could be easily beaten by a mediocre deck. In real-world applications, we believe it is more demanding to build winning-effective decks against competitive decks rather than random decks. In order to generate competitive opponent decks as test problem instances, we adopted a sequential manner: we sampled a deck \(x\) from the outputs of all algorithms for the last test problem instance, where the sampling distribution is weighted by \(f(x;x_o,A_p,A_o)\). We then used \(x\) as the input \(x_o\) for the next problem instance. The first test problem instance was obtained after 10 preliminary runs.

The performance of an algorithm is measured as follows. We ran each algorithm on each test problem instance 10 times. Each run was associated with a random seed, which controlled the initialization of \(x_p^{(0)}\) in \(s_0\) in \(Q\text{-DeckRec}\), and the randomness in evolution behaviors in GA. Then, we used the median of the 10 runs as the performance for the algorithm on the problem instance. To measure the significance of the differences for each pair of algorithms, we also conducted a two-tailed paired Welch’s \(t\)-tests with a confidence level 0.01 over all test instances. The null hypothesis is that the mean difference between the paired algorithms’ win rates is zero.

\(^3\)As in Eqn. 3.4, each optimal action is decided after calculating the state-action values of all possible actions \(((N - D) \cdot D + 1)\) and we need to take \(D\) actions per episode. When \(N = 312\) and \(D = 15\), the total number of state-action value evaluations is 66840.
In order to give a complete view of resource usage, we recorded both wall time and CPU time each algorithm takes to solve a test problem instance.

### 3.6 Results and Discussion

The performances of the three kinds of methods (GA, Q-DeckRec, and MC) are reported in Table 3.1, 3.2 and 3.3. All the reported numbers are the means of performance over the 20 test problem instances. As stated, the performance for each test problem instance is the median of 10 runs. Also, we find that all pairwise comparisons on the win rate are significant, except: (1) $GA_{20\text{ min}}$ vs. $GA_{25\text{ min}}$ (2) $Q\text{-DeckRec}_{3\text{ days}}$ vs. $GA_{20\text{ min}}$ (3) $Q\text{-DeckRec}_{3\text{ days}}$ vs. $GA_{25\text{ min}}$.

First, we observe that the performances of GA and Q-DeckRec improve as the wall time limits increase in our test ranges. This meets our expectation, because approximate COP solvers are supposed to get better solutions when using more computational resources. However, longer wall time limits than 20 minutes bring
3.6. Results and Discussion

diminishing improvement in GA, as we find there is no significant difference in the average win rate between \( GA_{20\text{min}} \) and \( GA_{25\text{min}} \).

From Table 3.1, we observe that GA calls the win rate evaluation function an increasing number of times as the wall time limit increases. As we stated, the win rate evaluation is computationally expensive involving simulating 300 matches. Therefore, all GA algorithms require high CPU time in the order of hours.

As shown in Table 3.2, Q-DeckRec can solve deck building problem instances with as little computational cost as 9.63 seconds in CPU time. Meanwhile, Q-DeckRec after 3-day training can build decks as winning-effective as \( GA_{25\text{min}} \) does, as evidenced by the non-significant difference between \( Q-\text{DeckRec}_{3\text{days}} \) vs. \( GA_{25\text{min}} \). Therefore, from the CPU time perspective, Q-DeckRec is much more efficient than GA (9.63 sec \( \ll \) 15.9 hr), when solving a new problem instance, because the computationally heavy match simulations have been "moved" to the training phase. This proves the merit of Q-DeckRec being a suitable deck recommendation system for large-scale or real-time application.

The number of function calls was 62K during the training of \( Q-\text{DeckRec}_{3\text{days}} \). This means there were 62K state transitions generated from roughly 4K (\( \approx 62K/15 \)) training episodes. Even if each of the 62K state transitions was unique, they still involved a tiny fraction of total states in our formulated state space. (The number of total states is the number of possible opponent decks times the number of possible player decks: \( (^{N_D}_D) \times (^{N_D}_D) \approx 1.97 \times 10^{50} \).) This shows that the MLP-based architecture is a well-chosen function approximator for generalizing state-action values.

For the MC-simulation method, we first report Mean Squared Error (MSE) and \( R^2 \) of the learned supervised learning model. We evaluated them using a standard 10-fold cross validation. On training data, \( MSE = 0.005 \) and \( R^2 = 0.86 \). On testing data, \( MSE = 0.008 \) and \( R^2 = 0.79 \). To our surprise, from Table 3.3, we find that the win rate does not monotonically increase as \( X \) increases. The performance peaks at 0.84, which is significantly lower than \( Q-\text{DeckRec}_{3\text{days}} \). While debugging the method, we observe that the predicted win rate (the outcome of Eqn. 3.10) monotonously increases as \( X \) increases. We suspect that since the supervised learning model cannot perfectly predict the real win rate, deck samples inevitably contain outlier decks.
with spuriously high predicted win rates. These outlier decks “tricked” the MC-simulation method to select them unfortunately. The results showed that the approach of building winning-effective decks in a sequential way as in Q-DeckRec is more robust.

3.7 Summary

In this chapter, we proposed a deck recommendation system named Q-DeckRec for one-vs-one Collectible Card Games, which is able to approximately identify winning-optimal decks in large-scale and real-time after a period of training and requires minimal domain knowledge. We designed experiments which demonstrate the advantages of this approach. Hence, the proposed work answers R.S.Q. 1 by showing an effective and efficient starting item recommendation system in the setting of one-vs-one match-based video games, which was not achievable by previous methods.
Chapter 4

Hero Recommendation in
Multiplayer Online Battle Arenas

4.1 Introduction

In this chapter, we continue to investigate R.S.Q. 1 but in a team-vs-team setting. In one-vs-one games, such as the Collectible Card Games, as we investigated in Chapter 3, we only need to consider one opponent when identifying winning-effective in-game elements; we also assume the in-game elements chosen by the opponent is known to the recommendation system. However, besides more opponents, there are more dynamics that should be taken into account in the pre-match stage in team-vs-team games; specifically, because multiple players may interactively make choices of in-game elements. Therefore, a winning-effective in-game element recommendation system needs to predict not only what other players’ choices will be, but also possible synergistic and oppositional relationships among in-game elements of multiple players. Such required capacities bring another layer of complexity to a recommendation system, which is worth an extra investigation in this chapter.

Multiplayer Online Battle Arenas (MOBA), as we introduced in Section 2.1.2, are an ideal test-bed for studying recommendation of winning-effective characters. They represent a collection of popular team-vs-team games and have a phase of sequential character drafting alternating between two teams; the character drafting phase is complex and challenging for human players due to a large number of possible choices by teammates and opponents which could lead to drastically different team compositions and outcomes. Thus, studying character recommendation in
MOBAs exemplifies R.S.Q. 1 in the team-vs-team setting, and we believe techniques answering it can be applied to other similar team-vs-team games in which winning-effective in-game elements also depend on multiple players’ choices. I will briefly describe these kinds of games in the following paragraphs, but interested readers are encouraged to read Section 2.1.2 for more details.

In a standard match in a MOBA game, two teams, each composed of five players, combat in a virtual game map, the goal is to beat the opposite team by destroying their base. Each player controls an in-game avatar, known as hero, co-operating with other teammates in attacking opponents’ heroes, armies, defensive structures, and ultimately the base, while defending their own in-game properties.

The selection of heroes, also known as pick or draft, takes place before each match starts and alternates between two teams until each player has selected one hero. We refer to the 10 heroes in a completed draft as a hero line-up. In a popular match mode named Ranked All Pick, the alternating order of drafting is “1-2-2-2-2-1”, meaning that the first team picks one hero, followed by the second team picking two heroes, then the first team picking two heroes, and so on. The process ends with the second team picking their last hero. We will first investigate the drafting order in this mode, since this mode does not involve more complicated elements such as banning (i.e., certain heroes can be prohibited from selection by either team), which we will investigate further in Section 4.4.6.

Heroes are often designed with a variety of physical attributes and skills, which together add to a team’s overall power. Therefore, players need to draft heroes that can enhance the strengths and compensate for the weaknesses of teammates’ heroes (i.e., synergy), while posing suppressing strengths over those in the opponent team (i.e., opposition). In games like League of Legends and DOTA 2, there are possibly more than 100 heroes that can be picked by a player at the time of drafting. As estimated by Hanke and Chaimowicz (2017), the number of possible hero line-ups in DOTA 2 is approximately $1.56 \times 10^{16}$. Due to the complex synergistic and opposing relationships among heroes, as described above, and the large numbers of follow-up pick possibilities by other players, selecting a suitable hero that can synergize teammates and counter opponents is a challenging task for human players.

Since MOBA games are team-based, we propose a hero pick recommendation
DraftArtist, for suggesting the approximated “optimal” hero pick for team victory. To this end, we view the drafting between two teams as a combinatorial game, characterized as a two-person zero-sum game with perfect information, discrete and sequential actions and deterministic rewards (Browne et al., 2012). Under this problem formulation, the goal is to decide at each step an optimal hero pick that leads to the hero line-up with the largest predicted win rate on its team, assuming that both teams behave optimally in the remaining picks. This problem, known as sequential decision making, can be solved by search algorithms. However, in the early or mid stage of the draft, exhaustive search might be prohibitive due to the large branching factor of more than 100 candidate heroes per pick. To tackle the large search space, we propose to use Monte Carlo Tree Search (MCTS) (Coulom, 2006), a heuristic search algorithm that efficiently estimates the long-term value of each pick by simulating possible following picks until completion. Each hero line-up is associated with a reward, defined as a predicted team win rate representing the estimated strength of the hero line-up. MCTS then back-propagates the reward to update the values of simulated picks. As more simulations are executed, the estimated values become more accurate, allowing MCTS to approach the next pick optimal for team victory.

The specific version of MCTS we use, namely Upper Confidence Bound applied to Trees, or UCT (Kocsis and Szepesvári, 2006), is an anytime algorithm, i.e., it has the theoretical guarantee to converge to the optimal pick given sufficient time and memory, while it can be stopped at any time to return an approximate solution. In contrast, previous works which we surveyed in Section 2.4.3 either predict player tendency to pick heroes (Summerville, Cook, and Steenhuisen, 2016), or leverage association rules mined from historical hero line-ups as heuristics (Hanke and Chaimowicz, 2017). They are not guaranteed to converge to the optimal hero pick for team victory.

The remaining chapter is structured as follows. We first introduce the problem formulation and then describe our algorithm in details. Next, we show the procedure of our experiments as well as evaluation results. Finally, we summarize and conclude the chapter.
4.2 Problem Formulation

As we introduced above, a draft takes place with players from the two teams picking heroes in an alternating order. We assume all players from the same team share the same goal, which can be stated as: build a hero line-up with the highest predicted win rate as a team prior to the match starting. Therefore, although in real life the five players on a team are different individuals, we regard the whole team as one collective player. As such, the draft can be considered as a game played by two players, each representing a team. Following the terminology in DOTA 2, the two players are called Radiant and Dire, respectively. Without loss of generality, we assume that the problem is formulated with Radiant being the team to pick heroes first. As a side note, although we abstract two teams into two players, the algorithm presented in Chapter 3 cannot be directly applied here because it assumes the choices of the opponent is fully known; specifically, the optimization of Eqn. 3.1 requires the input $x_o$, the opponent’s choice. Therefore the algorithm originally proposed for the one-vs-one setting fails to consider another important characteristic in the setting of MOBAs, the sequential order of character drafting, which means the opponent’s choice may not be fully known at the time of recommendation.

More specifically, the draft can be defined as a combinatorial game (Browne et al., 2012) with the following elements:

- the number of players: $n = 2$.
- game states: $S \subset \mathbb{Z}^N$. A game state $s \in S$ is an $N$-dimensional vector encoding which heroes have been picked by both players, where $N$ is the number of total distinct heroes. The components of $s$ can take value as one, negative one, and zero, corresponding to a hero being picked by Radiant, Dire, or neither of them, respectively:

$$s_i = \begin{cases} 
1, & \text{hero } i \text{ picked by Radiant} \\
-1, & \text{hero } i \text{ picked by Dire} \\
0, & \text{otherwise}
\end{cases} \quad (4.1)$$
4.2. Problem Formulation

The number of components equal to one (or negative one) cannot exceed five, because each player will eventually select five heroes. As per the rule of drafting, states are fully observable to both players.

- the initial game state $s_0 = 0^N$. $s_0 \in S$ is a blank draft with no hero picked yet, so it is a zero vector.

- the terminal states $S_T \subseteq S$, whereby the draft is complete. $S_T$ includes those states with exactly five components being one and five components being negative one, denoting a completed draft.

- the turn function $\rho : S \rightarrow \{\text{Radiant}, \text{Dire}\}$. It decides which player is about to pick in each state. $\rho$ is based on the pick order “1-2-2-2-2-1” between the two players.

- the set of actions $A$. Each action represents a hero pick by the current player. There are a finite number of actions and they are applied sequentially according to $\rho$.

- the transition function $f : S \times A \rightarrow S$. $f(s,a)$ modifies a non-terminal state $s \in S \setminus S_T$ reflecting the hero pick exerted by an action $a$, i.e., deterministically changing a zero component of $s$ to one (if the picking player is Radiant) or negative one (if the picking player is Dire).

- the reward function $R : S \rightarrow \mathbb{R}^2$. $R(s)$ outputs a two-dimension reward, with the first component $R^1(s)$ being the reward assigned to Radiant and the second component $R^2(s)$ being the reward assigned to Dire. Since both teams strive to maximize the predicted win rate of their team based on the completed draft, $R$ is only defined for terminal states $s \in S_T$, whereby $R^1(s) = -R^2(s) = w(s)$ with $w(s)$ denoting the predicted team win rate of Radiant.

Designed with the elements above, the draft is regarded as a combinatorial game characterizing the two-person, zero-sum property, perfect information, deterministic rewards, and discrete and sequential actions.
Chapter 4. Hero Recommendation in Multiplayer Online Battle Arenas

4.3 Methodology

Many classic games like Go, Chess, and Tic Tac Toe are also combinatorial games. A popular approach to solve combinatorial games is the minimax algorithm (Knuth and Moore, 1975), which finds the optimal action for a player by constructing a complete search tree comprising of all possible actions alternated by each player. Minimax assumes that both parties are playing rationally, i.e., each selecting actions that maximizes their accumulated rewards or minimizing accumulated costs. Since minimax is exhaustive in nature, it does not scale well for games with a large search space, such as Go. Therefore, heuristic functions to approximate the actions’ values after limited search depth and/or techniques for pruning search trees (Knuth and Moore, 1975) are needed. Even with these techniques, minimax may still be infeasible or not perform well for complex search problems.

As an alternative to minimax, Monte Carlo Tree Search (MCTS) (Coulom, 2006; Kocsis and Szepesvári, 2006; Nguyen et al., 2014) is a class of heuristic search algorithms for identifying near-optimal actions without having to construct the complete search tree. It has attracted much attention in recent years due to successful application to Go-playing programs (Silver et al., 2016; Silver et al., 2017), General Game Playing (Finnsson and Björnsson, 2008), and Real-Time Strategy games (Balla and Fern, 2009). In essence, MCTS involves iteratively building a search tree at each decision state to estimate the values of the state and available actions at that state. The main idea of MCTS in improving the efficiency of search is to prioritize expanding
the search trees in the direction of the most promising actions.

A draft in a normal MOBA is regarded as a combinatorial game with a branching factor at least 100 and depth 10, since popular MOBA games, such as League of Legends and DOTA 2, support more than 100 heroes to be selected for each of the 10 picks in a draft. As the branching factor is very large in this case, which makes minimax hardly a feasible approach, we propose to apply MCTS to compute the optimal pick for the current player.

Specifically, we proposed to use a particular version of MCTS called Upper Confidence Bound applied to trees (UCT) (Kocsis and Szepesvári, 2006) for this purpose. The search tree of UCT is built iteratively, with each node representing a state and each directed edge to child nodes representing the action resulting in a next state. It starts with a root node, which, in our case, represents the draft state \( s \) right before the current player picks the next hero. Then, the search tree is built based on the following four steps per iteration, until time or memory resource allocated is depleted:

**Selection.** In this step, the algorithm starts from the root node and traverses down the tree to reach a terminal node or an expandable node. A node is expandable if it is a non-terminal state and has child nodes unvisited. The tree traversal is done by successively selecting child nodes, following the tree policy, which in UCT is based on the Upper Confidence Bound (UCB1) criterion (Auer, Cesa-Bianchi, and Fischer, 2002):

\[
\pi_{\text{UCB1}}(s) = \arg \max_a \left\{ \bar{w} + c \sqrt{\frac{\log n(s)}{n(s,a)}} \right\},
\]

where \( s \) and \( a \) refer to a parent node and an action available at that node, respectively; \( \bar{w} \) is the average reward received after taking \( a \) at \( s \); \( n(s,a) \) is the number of times \( a \) is sampled at \( s \), and \( n(s) \) is the total number of times \( s \) is visited; and \( c \) is the exploration term, usually chosen empirically.

What is implied in Eqn. 4.2 is that UCT regards the choice of actions in the selection phase as multi-armed bandit problems: it focuses on exploiting the most promising actions to expand next (controlled by \( \bar{w} \)), while, as a balance, exploring more neglected branches of the tree (controlled by \( c \sqrt{\frac{\log n(s)}{n(s,a)}} \)).

**Expansion.** Unless the reached node from the last step is a terminal state, one of the unvisited actions is randomly sampled and applied to the node. The child node
representing the resulting state is added to the tree.

**Simulation.** From the newly expanded node, the algorithm performs a roll-out until the end of the game. During the roll-out, actions are performed according to a *default policy*, which by default is random sampling. Once the roll-out reaches a terminal state $s \in S_T$, $R(s)$ is calculated.

**Backpropagation.** The reward is backpropagated from the expanded node to the root node. Statistics on each node, i.e., the average reward and the number of visits, on the path are updated accordingly.

As the number of these four-step iterations increases, the algorithm expands the search tree larger and touches on more states and actions. Intuitively, the larger the search tree is, the better the value approximation of state nodes is. When the algorithm terminates, the action leading to the root’s most rewarding child node, i.e., highest $\bar{w}$, is returned. The tree building process of UCT is sketched in Figure 4.1.

It is proven that when proper parameters are configured and rewards are bounded to the range $[0, 1]$, UCT converges to minimax’s optimal action at a polynomial rate as the number of iterations grows to infinity (Kocsis and Szepesvári, 2006). This implies that applying UCT with theoretically sufficient time and memory will eventually converge to the optimal hero for team victory.

There are two benefits of MCTS in general that make it suitable for a hero pick recommendation system. First, state value estimation can solely rely on the backpropagation of the reward on terminal states without needing to resort to a hand-authored heuristic function. This reduces the amount of domain knowledge required. Second, MCTS is an anytime algorithm, which means tree building can be interrupted when a given time or memory constraint is exceeded and estimated values based on the search tree built so far can be returned immediately. This makes it feasible for our MCTS-based method to be deployed in large-scale online matches under real-world resource constraints.

### 4.3.1 Reward Function as a Win Rate Predictor

Now, we describe how the reward, i.e. the team win rate based on a hero line-up, is calculated. Following the previous notation, we use $w(s)$ to denote the predicted team win rate from the view of the Radiant player, based on a complete draft $s \in S_T$. 

This win rate can be computed by a machine learning classification model trained on a large amount of hero line-ups extracted from historical matches. During the training of such a model, the input feature vector is the completed draft as encoded in Eqn. 4.3; the output is a binary label indicating the observed match outcome (+1 for a win or 0 for a loss, from the view of Radiant). There are two properties making a model desirable for our task: (1) it should return the class probability, i.e., the probability of Radiant drafts winning the game, rather than just a binary prediction; and (2) it should be a non linear-based model that can capture the interrelationships between features (i.e., heroes).

4.4 Performance Evaluation

In this section, we detail the set up of our simulation-based experiments and demonstrate the effectiveness of UCT by showing that teams following our algorithm can form stronger hero line-ups against teams following other hero pick strategies. The relevant data and code are available at https://github.com/czxttkl/DraftArtist.

4.4.1 Data Collection

We chose the DOTA2 match dataset collected between February 11, 2016 to March 2, 2016 by Semenov et al. (2016). No major game update affecting the mechanics of the games occurred during the data collection phase. The original dataset contained five million “Ranked All Pick” matches, with each match containing hero line-up information and average player skill level (i.e., normal, high, and very high). To further reduce the impact introduced by different skill levels, we extracted a subset of matches played by players in the normal skill level. In total, our dataset contained 3,056,596 matches with 111 distinct heroes. The dataset was used to train a team win rate predictor as the reward function, as well as provide a basis for our simulations.

4.4.2 Win Rate Predictor Training

To prepare the data for training the team win rate predictor, each match’s hero line-up is encoded as a feature vector of length 111 (Eqn. 4.3) while the match outcome is used as the label.
Table 4.1: Performance of Team Win Rate Predictors

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC</td>
<td>0.53779</td>
<td>0.5</td>
</tr>
<tr>
<td>LR</td>
<td>0.63576</td>
<td>0.68767</td>
</tr>
<tr>
<td>GBDT</td>
<td>0.64172</td>
<td>0.70142</td>
</tr>
<tr>
<td>NN</td>
<td>0.65345</td>
<td>0.71437</td>
</tr>
</tbody>
</table>

We experimented with three commonly used classification models, Gradient Boosted Decision Tree (GBDT) (Friedman, 2001), Neural Network (NN) (Bishop, 2006), and a generalized linear model, Logistic Regression (LR), for the team win rate predictor. GBDT fits the logit of label probabilities with the outputs of a collection of decision trees learned using boosting (Friedman, Hastie, and Tibshirani, 2000). For the NN model, we used one input layer, one hidden layer, and one output layer. The hidden layer is comprised of a number of rectified linear units (ReLU) which transform the original features non-linearly to feed the output layer. The output layer is a single neuron with the sigmoid activation function $\frac{1}{1+\exp(-x)}$. LR models the logit of label probabilities as a linear combination of individual features without explicitly modeling the interactions among features. Therefore, although all the three models can predict the label probabilities on new data, GBDT and NN are more sophisticated and able to capture interactions among features. We also tested a naive baseline model which always outputs the majority class (MC). In our case, it always predicts a win for Radiant because there are 53.75% matches in which Radiant wins.

All hyperparameters like the number of hidden units of NN, the number of trees of GBDT and the regularization penalty of LR were determined through grid search on a 10-fold cross-validation procedure. In each fold, we split the data into the train, validate and test set in an 8:1:1 ratio. Candidate models of the same kind, but with different hyperparameters, were trained on the train dataset; the best hyperparameters were determined according to the classification accuracy on the validation dataset. The classification performance of the best model was measured on the test dataset.

We report the accuracy and area under ROC (receiver operating characteristic) curve (AUC) for each kind of model in Table 4.1, averaged over 10-fold cross validation. NN achieves the best prediction performance in both accuracy and AUC.
Additionally, the accuracy and AUC of NN are statistically higher than those of the other models according to a paired, two-tailed Welch’s t-test with confidence level 0.05. Therefore, we decided to use NN as the reward function in our simulation experiments later.

It is worth noting that the absolute difference between LR and NN is 1.8% for accuracy and 0.03 for AUC, which may give a wrong impression that match outcomes are accountable by the sum of individual heroes’ effects and hero interactions are not as important as we want to emphasize. We propose a possible explanation for this phenomenon: players already tried hard to build closely competitive hero line-ups in the collected matches and so good or bad hero interactions do not stand out. If we train the models with additional matches in which players (or human-like AI bots) are forced to play random heroes, NN may have a larger edge over LR. We will investigate such an issue in the future.

4.4.3 Simulation Setup

We designed four strategies for hero pick recommendation and test their effectiveness in terms of team victory. For every pair of strategies, we conducted 1000 simulations: in each simulation, two teams participate in the drafting process, with each team adopting one of the two strategies. At the end of each draft, we collected the predicted win rate as a measure of strength of the built hero line-ups. Finally, we report the mean of the predicted win rates over the 1000 drafts for each pair of strategies. The procedure of one simulation is summarized in Algorithm 2.

The four experimented strategies include:

- **UCT**: this strategy is what we proposed in Section 4.3. We use $\text{UCT}_{n,c}$ to denote a specific UCT version with $n$ iterations allowed and an exploration term $c$.

- **Association Rules (AR)**: this strategy was proposed by Hanke and Chaimowicz (2017). Two sets of association rules, namely “ally rules” and “enemy rules”, are extracted from the collected matches. Ally rules and enemy rules represent hero subsets that appear frequently in the same winning team or in the opposite team, respectively. At each turn, the strategy looks for the extracted association rules containing both the heroes picked already and the
Algorithm 2: Simulation of one match

Input: team1, team2, win rate predictor $M$
Output: $w(s)$

Initialize a new draft $s := s_0$

while $s \notin S_T$ do
  team = $\rho(s)$
  if $s$ equal to $s_0$ then
    $a = \text{weighted_sample_hero}()$
  else
    $a = \text{team.recommend_hero}(s)$
  end
  $s = f(s, a)$
end

$w(s) := \text{predicted by } M \text{ with input } s$

Table 4.2: Mean predicted win rate of $UCT_{n,1}$ (row) vs. $UCT_{n,c}$ (column). Numbers are from the view of the row strategy. Bold cells indicate the best value of $c$ for $UCT_{n,c}$ for each $n$, as they force $UCT_{n,1}$ have the lowest win rate.

<table>
<thead>
<tr>
<th></th>
<th>$UCT_{100,1}$</th>
<th>$UCT_{200,1}$</th>
<th>$UCT_{400,1}$</th>
<th>$UCT_{800,1}$</th>
<th>$UCT_{1600,1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$UCT_{100,2}^{-1}$</td>
<td>0.5</td>
<td>0.459</td>
<td>0.497</td>
<td>0.561</td>
<td>0.606</td>
</tr>
<tr>
<td>$UCT_{100,2}^{-2}$</td>
<td>0.5</td>
<td>0.435</td>
<td>0.476</td>
<td>0.544</td>
<td>0.577</td>
</tr>
<tr>
<td>$UCT_{100,2}^{-3}$</td>
<td>0.5</td>
<td>0.442</td>
<td>0.469</td>
<td>0.527</td>
<td>0.563</td>
</tr>
<tr>
<td>$UCT_{100,2}^{-4}$</td>
<td>0.5</td>
<td>0.444</td>
<td>0.469</td>
<td>0.509</td>
<td>0.563</td>
</tr>
<tr>
<td>$UCT_{100,2}^{-5}$</td>
<td>0.5</td>
<td>0.447</td>
<td>0.469</td>
<td>0.488</td>
<td>0.545</td>
</tr>
<tr>
<td>$UCT_{100,2}^{-6}$</td>
<td>0.5</td>
<td>0.464</td>
<td>0.492</td>
<td>0.485</td>
<td>0.510</td>
</tr>
<tr>
<td>$UCT_{100,2}^{-7}$</td>
<td>0.5</td>
<td>0.464</td>
<td>0.5</td>
<td>0.492</td>
<td>0.510</td>
</tr>
<tr>
<td>$UCT_{100,2}^{-8}$</td>
<td>0.5</td>
<td>0.485</td>
<td>0.525</td>
<td>0.492</td>
<td>0.510</td>
</tr>
<tr>
<td>$UCT_{100,2}^{-9}$</td>
<td>0.5</td>
<td>0.5</td>
<td>0.525</td>
<td>0.510</td>
<td>0.516</td>
</tr>
<tr>
<td>$UCT_{100,2}^{-10}$</td>
<td>0.5</td>
<td>0.540</td>
<td>0.534</td>
<td>0.516</td>
<td>0.516</td>
</tr>
</tbody>
</table>
heroes not picked yet. Those not picked yet will be selectively added to a candidate pool, from which the recommended hero will be uniformly sampled. In our implementation, we adopted the best criteria claimed by the authors: association rules are mined with 0.01% minimum support, and the metrics for selectively adding heroes to the candidate pool are “win rate” and “confidence” for ally rules and enemy rules, respectively. Readers can refer to the original paper Hanke and Chaimowicz, 2017 for more details on the approach.

- **Highest Win Rate (HWR):** each time, pick the hero not selected yet with the highest win rate, based on frequency counts on our dataset.

- **Random (RD):** each time, uniformly sample a hero not yet selected.

We did not implement the strategy based on sequence prediction models as proposed by Summerville, Cook, and Steenhuisen (2016), because their model requires training on hero pick sequences, which are not readily available in our dataset. In fact, hero pick sequences are not currently downloadable from the official APIs provided by the game and one will need to resort to third-party organizations for such data. We can implement the strategy in the future should we have access to such data of hero pick sequences.

Some additional implementation details are as follows. For each simulated draft, regardless of the strategy adopted, the first hero was sampled following the probability distribution reflecting how frequently each hero is picked in our dataset. This helped our experiments cover more possible scenarios. To ensure fairness when comparing pairs of strategies, among the 1000 simulations, each strategy was set to start first for half of the simulations. A shared random seed was set for each 500-simulations, to make sure that randomness is the same for both strategies. All strategies followed the rule that no hero can be picked twice. All experiments were conducted on a PC with an Intel i7-3632QM 2.20GHz CPU.

### 4.4.4 Parameter Search for UCT

We first ran simulations to determine the exploration term $c$ for the UCT strategy. We chose $UCT_{n,1}$ as benchmarks (i.e., UCT with the exploration term $c = 1$), where
Table 4.3: Mean predicted win rate of row strategy vs. column strategy. Simulations are based on the drafting rules of match mode “All Ranked". Numbers are from the view of the row strategy. The strategies are sorted in ascending order by their strengths, from top to bottom, and from left to right. Win rates symmetric to diagonal and always sum to one, thus half of them are omitted.

<table>
<thead>
<tr>
<th></th>
<th>RD</th>
<th>AR</th>
<th>UCT&lt;sub&gt;100,1&lt;/sub&gt;</th>
<th>UCT&lt;sub&gt;200,2&lt;/sub&gt;</th>
<th>HWR</th>
<th>UCT&lt;sub&gt;400,2&lt;/sub&gt;</th>
<th>UCT&lt;sub&gt;800,2&lt;/sub&gt;</th>
<th>UCT&lt;sub&gt;1600,2&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD</td>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AR</td>
<td>0.682</td>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UCT&lt;sub&gt;100,1&lt;/sub&gt;</td>
<td>0.783</td>
<td>0.663</td>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UCT&lt;sub&gt;200,2&lt;/sub&gt;</td>
<td>0.897</td>
<td>0.833</td>
<td>0.712</td>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HWR</td>
<td>0.883</td>
<td>0.846</td>
<td>0.715</td>
<td>0.516</td>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UCT&lt;sub&gt;400,2&lt;/sub&gt;</td>
<td>0.920</td>
<td>0.863</td>
<td>0.763</td>
<td>0.568</td>
<td>0.556</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UCT&lt;sub&gt;800,2&lt;/sub&gt;</td>
<td>0.928</td>
<td>0.878</td>
<td>0.776</td>
<td>0.591</td>
<td>0.593</td>
<td>0.524</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>UCT&lt;sub&gt;1600,2&lt;/sub&gt;</td>
<td>0.930</td>
<td>0.880</td>
<td>0.787</td>
<td>0.606</td>
<td>0.611</td>
<td>0.539</td>
<td>0.513</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Table 4.4: Average wall time per pick of different strategies (unit: millisecond, match mode: All Ranked).

<table>
<thead>
<tr>
<th></th>
<th>RD</th>
<th>AR</th>
<th>UCT&lt;sub&gt;100,1&lt;/sub&gt;</th>
<th>UCT&lt;sub&gt;200,2&lt;/sub&gt;</th>
<th>HWR</th>
<th>UCT&lt;sub&gt;400,2&lt;/sub&gt;</th>
<th>UCT&lt;sub&gt;800,2&lt;/sub&gt;</th>
<th>UCT&lt;sub&gt;1600,2&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.02</td>
<td>11</td>
<td>43</td>
<td>96</td>
<td>0.1</td>
<td>281</td>
<td>562</td>
<td>1120</td>
</tr>
</tbody>
</table>

n = {100, 200, 400, 800, 1600}. For each UCT<sub>n,1</sub>, we then created multiple UCT<sub>n,c</sub> strategies to play with, with c ranging from 2<sup>-6</sup> to 2 at a scaling rate of 2.

The results are shown in Table 4.2. Tuning c was not helpful when UCT was only allowed with 100 iterations. This is because c controlled the exploration strength for tree node selection in the selection step, which never kicked off within 100 iterations due to the large number of available actions<sup>1</sup>. We can infer that the best value of c to couple with n = {200, 400, 800, 1600} is 2<sup>-5</sup>, 2<sup>-2</sup>, 2<sup>-1</sup> and 2<sup>-1</sup>, respectively, because they forced UCT<sub>n,1</sub> to have the lowest win rate (indicated by the bold cells in Table 4.2). Note that there is a general trend that the best c increases as n increases.

4.4.5 Simulation Results

Based on the results from the parameter search, we finally chose a list of strategies to compare with each other: RD, AR, HWR, UCT<sub>100,1</sub>, UCT<sub>200,2</sub>, UCT<sub>400,2</sub>, UCT<sub>800,2</sub>, and UCT<sub>1600,2</sub>. The simulation results are summarized in Table 4.3, in

<sup>1</sup> We have 111 distinct heroes and 10 turns in a draft, which means applying UCT at any turn will start from a root state with more than 100 possible child nodes (actions) to expand. Since one iteration only expands one new child node, after 100 iterations, the root node is still expandable. Therefore, no selection step will happen.
which each cell contains the mean predicted win rate of the strategy displayed on the respective row.

We defined the strength of a strategy as the number of strategies it can defeat with more than 50% win rate. Therefore, the weakest to strongest strategies are: RD, AR, UCT$_{100,2}$, UCT$_{200,2}$, HWR, UCT$_{400,2}$, UCT$_{800,2}$, and UCT$_{1600,2}$. Given a sufficient number of iterations ($\geq 400$), UCT strategies can outperform all non-UCT strategies, which proves the effectiveness of our proposed algorithm. There is a general trend that UCT improves the win rate as the number of iterations increases. Specifically, UCT$_{1600,2}$ can beat HWR with a 61.1% win rate, highest among the other UCT-based strategies with fewer iterations. However, we can observe the phenomenon of diminishing gain as the number of iterations exceeds a certain threshold: UCT$_{1600,2}$ has a 51.3% win rate against UCT$_{800,2}$, only marginally better than 50% but with double the number of iterations.

Among non-UCT strategies, the HWR strategy that always picks the highest win rate hero achieves the best performance, which can defeat UCT with 200 iterations with a 51.6% win rate. However, HWR cannot prevail over UCT strategies that are allowed more than 200 iterations. AR adopted from Hanke and Chaimowicz (2017) defeats RD with 68.2% win rate for our implementation, while the original authors reported a 76.4% win rate against RD. The discrepancy may be due to different datasets being used to mine association rules and train the win rate predictor.

To show the efficiency of different strategies, we report the average wall time (i.e. real elapsed time) needed per pick in Table 4.4. The time was recorded excluding the first pick, since it was based on weighted random sampling. UCT-based strategies
take 1.12 seconds or less per pick, which is a small fraction of the 25 second time limit per pick for human players in real matches (Gamepedia, 2018). This demonstrates the feasibility of applying the UCT-based hero recommendation system online in large scale and real time.

### 4.4.6 Extension to Other Match Modes

So far in this chapter, we have focused on the drafting rules of match mode *All Ranked* in DOTA 2. However, our MCTS-based algorithm is not limited solely to this mode and can be adapted to other modes, whereby additional drafting rules and mechanics are deployed. In this section, we detail how our algorithm can be extended to another popular match mode called *Captain Mode* to demonstrate our algorithm’s generality.

During the drafting process in Captain Mode matches, there is another type of action, called *hero banning*, that players can take besides hero pick. In the turns of banning, a team can designate certain heroes to be prohibited from selection by either team. The drafting order is represented in Figure 4.2: two teams first alternate to ban three heroes, then alternate to pick two heroes, then alternate to ban two heroes, and so on. As the result, the drafting is 22 steps long, instead of just 10 as in the case of *All Ranked*. The interleaving nature between bans and picks adds another level of complexity to the drafting, as players need to consider more possible strategies to prevent opponents from selecting their desired heroes.

Despite being more complex, the drafting in Captain Mode matches can be formulated as a combinatorial game similar to that in All Ranked matches but with a few minor adjustments:
• The components of a game state \( s \) can take an additional value of a special symbol \( \Xi \) denoting corresponding banned heroes, i.e.,:

\[
\begin{align*}
    s_i &= \begin{cases} 
        1, & \text{hero } i \text{ picked by Radiant} \\
        -1, & \text{hero } i \text{ picked by Dire} \\
        \Xi, & \text{hero } i \text{ banned} \\
        0, & \text{otherwise}
    \end{cases}
\end{align*}
\]  

(4.3)

The terminal state set \( S_T \) will also change accordingly. A terminal state \( s \in S_T \) include five ones, five negative ones, and 12 special symbol \( \Xi \).

• An action can be either a ban or a pick. If it is a ban, \( f(s,a) \) changes a zero component of a non-terminal state \( s \in S \setminus S_T \) to \( \Xi \).

• The turn function \( \rho \) will also be updated to reflect the drafting order in Figure 4.2.

• When a terminal state \( s \in S_T \) is fed into the reward function \( R(s) \), its components of \( \Xi \) will be treated as zeroes.

Given the above formulation of the Captain Mode drafting, in the form of a combinatorial game, the UCT algorithm can be applied directly in the same way as described in Section 4.3. Within the drafting rules of Captain Mode matches, we conducted the same simulation-based experiments to compare UCT with the baselines RD, AR and HWR. When taking the banning action, AR is executed from the perspective of the opponent - identifying candidate heroes to ban as if the opponent is to pick; HWR always chooses among the available heroes with the highest win rate for both the pick and ban actions. Regarding other implementation details, the selection frequency-weighted sampling was executed for the first hero to ban rather than to pick. \( c \) becomes effective in \( UCT_{100,c} \), since the simulation step may kick off after the starting root node is no longer expandable. We find the best \( c \) to match with \( n = 100 \) is \( 2^{-5} \), while the best \( c \) values to couple with other \( n = \{200, 400, 800, 1600\} \) remain the same as in All Ranked experiments.
The evaluation results are shown in Table 4.5. First, we observe that the strength order of the tested strategies remains the same, with \( RD \) being the weakest and \( UCT_{1600,2^{-1}} \) being the strongest. Second, the win rate of UCT strategies against non-UCT strategies usually sees a 1-3% improvement in absolute value compared to the counterparts in All Ranked experiments. For example, \( UCT_{1600,2^{-1}} \) has a 62.7% win rate against HWR in Captain Mode, which is larger than the win rate \( UCT_{1600,2^{-1}} \) has against HWR in All Ranked (61.1%). This highlights the advantage of employing simulation-based approaches such as MCTS: that the advanced planning brought by the MCTS algorithm could further widen its gap over baseline methods in more sophisticated settings.

We also observe a negligible change in the average wall time needed per pick in Captain Mode-based simulations, as compared to All Ranked (Table 4.4), so we do not report it here. This implies that the overhead incurred by the deeper search in UCT is relatively smaller than that in other components, such as computing the reward function.

### 4.5 Summary

In this chapter, we presented a solution to R.S.Q. 1 for the team-vs-team setting. In particular, we proposed a recommendation system, DraftArtist, that can effectively and efficiently search for the optimal hero pick that can achieve team victory based on Monte Carlo Tree Search. We designed and conducted simulation-based experiments based on two kinds of drafting rules, confirming that MCTS-based recommendations can lead to stronger hero line-ups in terms of predicted team win rates, as compared to other baselines.
Chapter 5

Engagement Optimized
Matchmaking

5.1 Introduction

This chapter aims to answer R.S.Q. 2, which we reiterate here:

How can we design systems working in the pre-match stage which recommend in-game elements for improving player engagement through data-driven approaches?

(R.S.Q. 2)

Techniques that we studied in the previous two chapters give players a better chance of winning. Guided by theories such as Self-Determination Theory (SDT) (Ryan and Deci, 2000) and Flow (Csikszentmihalyi, 1990), such techniques are useful to help incompetent players regain competence and enter cognitive states like flow, which we believe will then lead to engagement. In this sense, techniques from the previous two chapters are categorized as indirect and theory-driven methods.

In another perspective, R.S.Q. 2 takes a more direct stab by investigating data-driven methods to influence player engagement using in-game element recommendations. Data-driven approaches determine recommendations purely based on analysis and interpretation from past player data, without the need to rely on psychological and sociological theories as in R.S.Q. 1. To achieve this goal, the system features the abilities to predict the change of player engagement resulting from each in-game element and efficiently search for the in-game element leading to the optimal player
engagement, where player engagement is often modeled and predicted quantitatively based on machine learning models. Bypassing the intermediate links between winning-effectiveness and player competence, and between player competence and engagement, **R.S.Q. 2** offers an alternative approach to improve player engagement.

We answered **R.S.Q. 2** with an opponent recommendation system. Opponent recommendation is also known as “matchmaking” (Medler, 2011). Since the term matchmaking has been used more frequently in existing literature, we will use matchmaking and opponent recommendation interchangeably in the rest of this chapter.

In this work, we specifically developed an opponent recommendation system for Player-vs-Player (PvP) games, a subset of match-based games described in Chapter 1. In this case, players themselves are the subject of recommendation and the goal is to connect players to form PvP matches. Back in Chapter 1, we discussed that participants can be either human players or AI bots in match-based games, unless those as the recipients of our recommendations should be human players. PvP games allow only human players, and in the problem we study here, every player is the recipient as well as the subject of recommendation.

We first look at the definition of opponent recommendation (or matchmaking). Most existing works have restricted the definition of matchmaking as players are exactly divided and matched. In our work, we also adopt this definition. This definition constrains the search space, since no player is allowed to be matched into multiple matches. For example, matchmaking applied in a one-vs-one PvP game will pair each player to exactly one other player.

In practice, a matchmaking system often operates on several constraints, including physical and technical. For example, the system may be constrained by players’ geo-location and network latency because connection lags caused by long geographical distance between multiple players may incur interrupted play experience. Thus, the system may prioritize to matchmake players from close regions.

Beyond physical and technical constraints, the strategy a fair amount of matchmaking systems employ is creating fair games. This goal is inspired by the theories such as Self-Determination Theory (SDT) (Ryan and Deci, 2000) and Flow (Csikszentmihalyi, 1990), as we introduced in Section 2.2, which proposed that one component of player engagement is from challenges matching with skills. This means
that these matchmaking systems are often developed with the goal of matching closely skilled players to create competitive but fair games, where appropriately challenged players are matched up.

In order to deliver on creating fair matches, numerous skill models have been proposed, such as Elo (Elo, 1978), Glicko (Glickman, 1999) and TrueSkill (Herbrich, Minka, and Graepel, 2006). The main idea of skill models is to estimate player skills in formal mathematical frameworks such that outcomes between players can be predicted numerically. Based on numerical outcome prediction, matchmaking systems can then divide players into groups which are projected to have balanced outcome probabilities (as close as possible to 50% win rates). Skill models will be reviewed in more details in Section 5.2.1.

In this chapter we challenge the goal of always creating fairly matched games because such a goal was developed based on theories but is worthy of deep investigation in the lens of a data-driven approach, in which player engagement is examined individually depending on personal experience and is measured quantitatively. Our hypothesis is that while theories provide general guidelines for player engagement improvement, they fail to differentiate personal experiences and thus are not optimal for player engagement that are quantitatively measured. Let’s review some hypothetical examples where such assumptions break. Consider a cautious player who cares about protecting his rank among friends, and a risk-taker who enjoys difficult matches. Pairing these players with similarly skilled opponents will have different effects: the cautious player may be frustrated with just a 50% win rate because his rank cannot improve further, while the risk-taker may feel bored even at a 50% win rate. Additionally, the experience the player had with the game in the previous matches, such as how many matches he won before or lost, can affect their expectations and their experience as well as their performance.

To give a more concrete example, see Table 5.1, which displays data from a popular PvP game developed by Electronic Arts, Inc. Please note that we cannot disclose the name of the game in this dissertation due to contractual limitations. In the table, we show an example of how win/loss in previous games can affect players’ engagement, in this case defined in terms of 7-day churn risk. It measures the likelihood
of one player stopping playing the game in the next 7 days, or from a population-wise perspective, the ratio of a group of players who stop playing within the next 7 days time after a match. We can see from Table 5.1 that the churn risk varies drastically based on players’ recent match outcomes, thus raising the need to cater player engagement by a data-driven approach.

Given this information, we then devised a new matchmaking framework, called Engagement Optimized Matchmaking (EOMM). In this system, we rely on two key insights: (1) the effectiveness of matchmaking needs to be measured quantitatively; and (2) matchmaking should depend on dynamic and individual player states. We therefore formulated matchmaking into an optimization problem, where we maximized the overall player engagement, or equivalently, minimized the overall player disengagement through pairing two players. Player disengagement is a chosen quantitative metric at the disposal of practitioners, for example, churn risk within a period of time, such as a week. For the illustration purpose, we also use churn risk as the player disengagement metric in this chapter. Next, we modeled all players to be matched as a complete graph, where each player is a node, and an edge between two players is their sum churn risk, if paired. The churn risk depends on individual player states at the moment of matchmaking, which include but not limit to player skills. We then developed an optimization system that solves a minimum weight perfect matching (MWPM) problem finding non-overlapping pairs with the minimal sum of edge weights on a complete graph. In order to implement and test this system, we developed a simulated system based on real data of a popular game made by Electronic Arts, Inc. (EA). As we shall show in the results section, the system showed improvement in enhancing player engagement as compared to equal-skill based and other matchmaking methods. At the time of writing, we were unable to test our system in production due to time constraint and would leave that for future work.

EOMM is both flexible and computationally feasible. The final system is composed of three components: a skill model, an engagement (e.g., churn) prediction model and a graph matching model. All can be efficiently implemented and independently upgraded. The disengagement metric in the engagement prediction model can be selected differently for various interests, e.g., in-game time, or even
5.2 Related Work

Table 5.1: An example of the impact of player states on their engagement. Data is from a popular PvP game made by EA. Average churn risks vary drastically upon players’ recent three match outcomes (Win, Lose or Draw). Churn risk is measured by the ratio of the players who stop playing within a period time (7 days in this table) after a match. The churn risk of some states with repeated losses (5.1%) is almost twice as much as those of other “safer” states (2.6%-2.7%).

<table>
<thead>
<tr>
<th>Last 3 Outcomes</th>
<th>Churn Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLW</td>
<td>LLW</td>
</tr>
<tr>
<td>WWW</td>
<td>3.7%</td>
</tr>
<tr>
<td>DLL</td>
<td>LWL</td>
</tr>
<tr>
<td>LLL</td>
<td>4.9%</td>
</tr>
<tr>
<td>....</td>
<td>5.1%</td>
</tr>
</tbody>
</table>

spending.

The rest of this chapter is organized as follows. After reviewing the related work, we will present the formulation of matchmaking as an optimization problem on a graph. Then we describe theoretical findings comparing EOMM and other matchmaking methods. We then show the case study applying EOMM on real data. Finally, we will conclude with a discussion of the results and future directions.

5.2 Related Work

In this chapter, we introduce three relevant fields, namely skill modeling, player engagement prediction, and graph matching. Each field is actually one critical component of our proposed matchmaking framework.

5.2.1 Skill Modeling

Skill modeling aims to model player skills in mathematical frameworks and gives numeric prediction on match outcomes between players. Skill models can facilitate EOMM in the decision of player assignment. We would like also to compare our work to fairness-based matchmaking systems that use skill models. Therefore, we will review the models proposed for skill modeling here.
Skill modeling has had a long history. Dating back to 1952, the *Bradley-Terry model* (Bradley and Terry, 1952) was developed to deal with repeated pairwise comparisons among a group of subjects. In the Bradley-Terry model, a player $i$ is assumed to have a fixed, positive skill scalar, $r_i$, and the winning probability of player $i$ against player $j$ is the ratio of player $i$’s skill in the sum of skills of both players. In its original form, the Bradley-Terry model estimates player skills only after observing all pairwise comparisons. While feasible for small groups of players, requiring $O(n^2)$ matches is prohibitive for large player pools. One can show that the Bradley-Terry model is equivalent to a logistic regression model (Agresti and Kateri, 2011) in which each coefficient $w_i$ corresponds to $\log(r_i)$.

The *Elo system* (Elo, 1978) addresses the relative skill ratings in player-versus-player games, such as chess, by proposing a probabilistic model. Elo captures player performance, $p_i$, as a random variable following a one dimension Gaussian distribution with a mean, $r_i$, and a fixed variance, $\beta^2$, shared by all players. In the Elo system, $r_i$ is updated depending on the extent of agreement between expected outcomes and real outcomes. For example, a player beating a higher skilled player yields a large update in adjusting their skill means closer. Unlike the original Bradley-Terry model, in this model $r_i$ can be updated at an ongoing basis, i.e., as soon as after every match of player $i$.

The *Glicko system* (Glickman, 1999), a Bayesian ranking rating system, was later introduced. Besides mean player skill, $r_i$, it also models the belief about a player’s skill as $RD_i$ (rating deviation). As they play more games, the belief about their skills become stronger hence $RD_i$ decreases. However, $RD_i$ increases when a player ceases to play for long time. To achieve high efficiency, Glicko uses an approximation Bayesian algorithm to update $r_i$ and $RD_i$.

Neither the Bradley-Terry model, the Elo system or the Glicko system was initially applicable to team-oriented games until some researchers looked into how to generalize these models for use in team games (Herbrich, Minka, and Graepel, 2006; Huang, Lin, and Weng, 2004; Menke and Martinez, 2008). *TrueSkill system* (Herbrich, Minka, and Graepel, 2006) extends the Elo system to games with flexible numbers of players and teams. TrueSkill assumes that the outcome of a team-vs-team match results from the discrepancy of team performance; team performance is the sum
of individual performances, whereas an individual performance is governed by his or her skill rating. Given TrueSkill’s form in a generative model, skill ratings can be computed by Bayesian inference using a specific technique named Expectation Propagation (Minka, 2001). Huang, Lin, and Weng (2004) derived analytic update rules that are easy to interpret and implement using a Bayesian approximation method. Their model works for games with multiple teams and multiple players by treating a $k$-team match as several two-team matches. Menke and Martinez (2008) proposed to use Artificial Neural Network (ANN) for extending the Bradley-Terry model to team-based games.

Researchers have proposed more advanced skill models to overcome limitations of aforementioned skill models. First, the aforementioned models only abstract player skills into a single dimension but in fact many games require skills from multiple facets. To improve upon that, the works by Chen and Joachims (2016) and Stanescu (2011) model player skills in multi-dimensions, such as offensive and defensive abilities. Delalleau et al. (2012) proposed a neural network based skill model which learns multi-dimensional latent skill embeddings of players and is claimed to outperform TrueSkill in a team based game. Second, it is desirable to incorporate more domain-specific information that might help estimate player skills more accurately therefore researchers have devised skill models for specific game genres. For example, Di Fatta, Haworth, and Regan (2009)’s skill model for chess integrate move and position evaluation. Chen et al. (2016b) and Suznjevic, Matijasevic, and Konfic (2015) used skill models to capture players’ general skills plus their skills in specific characters or roles in Multi-player Online Battle Arena (MOBA) games. Moreover, a recent technical report from Microsoft (Minka, Cleven, and Zaykov, 2018) reveals the latest effort to improve the original TrueSkill model (Herbrich, Minka, and Graepel, 2006) in several ways, including that a player’s in-game statistics such as kill and death counts, besides team win/loss, can be incorporated into TrueSkill’s probabilistic framework, and a player’s performance in other game modes could also help skill estimation when he starts a new game mode.
5.2.2 Graph Matching

Graph matching is another critical component used to search for optimal player pairing in EOMM. In a graph $G = (V, E)$, a matching is a set of pairwise non-adjacent edges West, 2001; that is, no two edges share a common vertex. A perfect matching is a matching with every vertex in $G$ incident on exactly one edge in the matching. In a weighted graph $G$, a minimum weight matching (MWM) is the matching with the lowest sum of edge weights. A minimum weight perfect matching (MWPM) is the perfect matching with the lowest sum of edge weights.

As we will show in Section 5.3, the EOMM framework converts the problem of determining optimal match assignment to the problem of seeking MWPM in a weighted graph. MWM/MWPM have broad applications in other fields, including creating pairs following specific rules in chess tournaments Ólafsson, 1990, scheduling training sessions among NASA shuttle cockpit simulators Bell, 1994 and transmitting images over networks Riskin et al., 1994. In a similar spirit, Ólafsson Ólafsson, 1990 leverages MWPM algorithm to determine opponents. Their goal, however, was to create matches maximally adhering specific rules of chess tournament, which is different from our goal, which is to optimize for player engagement.

The first attempt to solve MWPM is the polynomial time blossom algorithm proposed by Edmond Edmonds, 1965a; Edmonds, 1965b. Since then, researchers have steadily improved upon this algorithm. We will compare and discuss those improved methods later when we introduce EOMM.

5.2.3 Player Engagement Prediction

Another important building component of EOMM is player engagement prediction. During one step of matchmaking, EOMM needs to quickly predict player engagement resulting from every hypothetical matchmaking. Here we rely on engagement prediction based on machine learning models (i.e., data-driven approaches) because machine learning models can be trained from massive player data usually readily collected by game companies and be used for prediction with fast speed. Readers can refer to Section 2.3 for literature overview, in which we have introduced several
common player engagement metrics as the labels for machine learning models, as well as several representative models with their application contexts.

5.3 Engagement Optimized Matchmaking

In this section, we introduce the EOMM framework which formulates matchmaking as an optimization problem. Here we will describe the details of match assignment for 1-vs-1 games. We will discuss how EOMM can be extended to the matches with more players in the final section.

In contrast with existing matchmaking methods that heuristically pair similarly skilled co-players, EOMM aims to match players in an optimal way that maximizes overall player engagement.

5.3.1 Optimization Objective

In practice, matchmaking is applied to a pool of players, \( \mathcal{P} = \{p_1, \ldots, p_N\} \), who are waiting to start 1-vs-1 matches. We assume \( N \) to be an even number, such that all players can be paired. The objective of EOMM is to maximize the overall player engagement, or equivalently, minimize the overall player disengagement.

For illustrative purpose, we use churn risk as a concrete metric of disengagement, although EOMM is designed to be a general framework working with other engagement/disengagement metrics. The term “churn” is used by convention, which represents a status of disengagement, i.e., a player not playing any games within a subsequent time frame, not necessarily a permanent churn. We denote the churn risk of player \( p_i \) after matchmaking with player \( p_j \) as \( c_{i,j} \), which is a function of both players’ states, i.e., \( c_{i,j} = \Pr(p_i \text{ churns}|s_i, s_j) = c(s_i, s_j) \). A player state is a collection of features that profile an individual player, including but not limited to install date, skill, play frequency, performance and etc. We will elaborate on learning \( c_{i,j} \) in the subsequent sections. Note that \( c_{i,j} \neq c_{j,i} \) since two players in a paired match may be impacted differently. We use a list of player tuples, \( \mathcal{M} = \{(p_i, p_j)\} \), to denote a matchmaking result, i.e., a pair assignment, in which all players in \( \mathcal{P} \) are paired once and only once. Defining the overall player disengagement as the sum of individual churn risks, EOMM seeks for an optimal pair assignment \( \mathcal{M}^* \) such that:
Chapter 5. Engagement Optimized Matchmaking

![Diagram](image)

**Figure 5.1**: Model matchmaking on a complete graph. Each node represents a player, and every edge is associated with the sum engagement metric of two players if paired. EOMM amounts to finding an optimal pair assignment on $\mathcal{G}$.

\[
\mathcal{M}^* = \arg\min_{\mathcal{M}} \sum_{(p_i, p_j) \in \mathcal{M}} c(s_i, s_j) + c(s_j, s_i)
\] (5.1)

We construct a graph, $\mathcal{G}$, to model this environment (see Figure 5.1). Each player $p_i$ is a node of the graph, who has a player state, $s_i$, before matchmaking. The edge between two players $p_i$ and $p_j$ is associated with a weight $c_{i,j} + c_{j,i}$, which is the expected sum disengagement metric if they are paired. Note that $\mathcal{G}$ is a complete graph in that all pairs of players can be possibly connected. Once all $c_{i,j}$ are computed, finding $\mathcal{M}^*$ in Eqn. 5.1 is converted to a minimum weight perfect matching problem, i.e., finding a pair assignment with the minimal sum weights of edges on graph $\mathcal{G}$.

5.3.2 Predicting Churn Risks

We learn the function $c_{i,j} = c(s_i, s_j)$ as a churn prediction problem. In its original form, the churn risk $c_{i,j}$ of player $p_i$ after matchmaking depends on the states from both the player and their opponent. Unfortunately, the well-established churn prediction studies cannot be employed here, because they only use features of players themselves without considering those of opponents. Also, naively feeding both player states as input will double the feature dimension, which makes the prediction
unintelligible and harder since much more training data is needed.

One way to simplify the prediction of $c_{i,j}$ is to base it only on player $p_i$’s own state, $s_i$, and the resulting match outcome, $o_{i,j}$, from the view of $p_i$. This works because the opponent’s state, $s_j$, such as skill, play history and style, does not directly interact with player $p_i$’s churn risk $c_{i,j}$. It, however, influences the upcoming match outcome, which is directly perceivable by player $p_i$ and thus affects $p_i$’s churn. Once the match outcome $o_{i,j}$ is known, $c_{i,j}$ becomes conditionally independent to the opponent’s state, $s_j$. Formally, this property is represented as:

$$
\Pr(p_i \text{ churns}|s_i, s_j, o_{i,j}) = \Pr(p_i \text{ churns}|s_i, o_{i,j}),
$$

which can be written in a concise form:

$$
c(s_i, s_j, o_{i,j}) = c(s_i, o_{i,j})
$$

In this chapter, we assume that game outcomes are sampled from a finite set, $O$, such as Win, Lose and Draw. For example, $o_{i,j} = W$ means that $p_i$ wins over $p_j$, while $o_{j,i} = L$ represents the same outcome from the view of $p_j$. To predict game outcomes, we employ the standard skill models (Elo, 1978; Glickman, 1999) that are widely adopted in the video game industry. These models use both players’ skills, which are a proxy of their entire player states, as the input for the prediction. We denote player $p_i$’s skill representation as $\mu_i$, which is, for example, Elo score (Elo, 1978) or Glicko mean and RD (Glickman, 1999). Note that $\mu_i$ is part of player state $s_i$. As a result, we have:

$$
\Pr(o_{i,j}|s_i, s_j) \approx \Pr(o_{i,j}|\mu_i, \mu_j),
$$
Putting them together, we can efficiently predict the churn risks of paired players in Eqn. 5.1:

\[
\begin{align*}
    & c(s_i, s_j) + c(s_j, s_i) \\
    &= \sum_{o_{ij} \in \mathcal{O}} \Pr(o_{ij} | s_i, s_j) \left( c(s_i, s_j, o_{ij}) + c(s_j, s_i, o_{ij}) \right) \\
    &\approx \sum_{o_{ij} \in \mathcal{O}} \Pr(o_{ij} | \mu_i, \mu_j) \left( c(s_i, o_{ij}) + c(s_j, o_{ij}) \right),
\end{align*}
\]

where the first equality is a marginalization on game outcome, \( o_{ij} \). In the approximate equality, the conditional independence of \( c_{ij} \) on \( s_j \) given \( o_{ij} \) (Eqn. 5.3) and the game outcome prediction (Eqn. 5.4) are used.

Now \( c(s_i, o_{ij}) \) can be efficiently learned based on any preferred churn prediction model. The input features are the updated player state based on the predicted game outcome of the hypothetical matchmaking, i.e., \( s_i^{\text{update}} \leftarrow s_i \) and \( o_{ij} \). We can decompose the original player state as \( s_i = [o^K_i, \hat{s}_i] \), where \( o^K_i \) is a vector of the latest \( K \) game outcomes (for example, \( o^K_i = \text{LWLDL} \) when \( K = 5 \)), and \( \hat{s}_i \) represent the rest of features in \( s_i \). If \( p_i \) is hypothetically matched with \( p_j \), \( s_i \) will be updated as:

\[
\begin{align*}
    s_i^{\text{update}} &\leftarrow s_i \text{ and } o_{ij} \\
    &= [o^K_i, \hat{s}_i] \text{ and } o_{ij} \\
    &= [o^{K+1}_i, \hat{s}_i^{\text{update}}]
\end{align*}
\]

We use \( \hat{s}_i^{\text{update}} \) to indicate that non-game-outcome features are also updated after the new match. For example, the total number of games played increments by one.

5.3.3 Finding the Optimal Pair Assignment

Given the predicted churn risks of each pair of players, i.e., the weight of every edge in \( \mathcal{G} \), EOMM reduces to a minimum weight perfect matching (MWPM) problem. The goal is to find a pair assignment, \( M^* \), on a complete graph, \( \mathcal{G} \), which has the minimal sum weights of edges.

For a graph with \( N \) node, the brute-force method is to compare all \( \binom{N}{2} / 2^N \) possible pair assignments and find the best one, but the time complexity is too high to
be feasible in practical systems. Fortunately, many polynomial time algorithms exist for the MWPM problem. For example, several algorithms can solve the problem in the worst time complexity $O(N^3)$ (Gabow, 1974; Lawler, 2001). If engagement measurements are pure integers, there exists a slightly faster algorithm (Gabow, 1985) with running time $O(N^{2.5} \log K)$ where $K$ is the largest magnitude of an edge weight. There also exist greedy algorithms, such as Drake and Hougardy (2003) and Duan and Pettie (2014), with faster running time to find suboptimal solutions. Moreover, MWPM can be solved in parallel as proposed by (Osiakwan and Akl, 1990).

5.4 Theoretical Findings

Besides generating optimal matchmaking assignments, EOMM provides a framework to conduct theoretical analysis on other matchmaking related problems. We use this framework to compare EOMM with other matchmaking strategies under different hypothetical situations to obtain many insights. Without loss of generality, we focus our discussion on 1-vs-1 games with possible game outcomes sampled from Win, Lose and Draw.

Using the same notation in Section 5.3, we investigate a pair of players $p_i, p_j \in \mathcal{P}, i \neq j$. When $c(s_i, s_j) = c(o_{ij})$, i.e., a player’s churn risk only depends on the game outcome of the upcoming match, regardless of all other states. This simplification for Eqn. 5.8, where $s_i^{\text{update}}$ only considers $o_{ij}$ but ignores $s_{ij}$, has interesting implications.

- If $c(\text{Win}) + c(\text{Lose}) > 2 \cdot c(\text{Draw})$, i.e., the sum churn risk of two matched players in a tied game is lower than that in a non-tied game. Under this circumstance, the equal-skill based matchmaking is equivalent to EOMM, as both strive to form matches with Draw outcomes as many as possible. This explains the intuition and popularity behind equal-skill matchmaking. But we should be aware of its conditional applicability, while EOMM is instead always optimal.

- If $c(\text{Win}) + c(\text{Lose}) < 2 \cdot c(\text{Draw})$, equal-skill based matchmaking is actually worst among all matchmaking schemes, as its goal to create close matches contrarily minimizes the overall player engagement. Although this situation contradicts with the common intuition that fair matches are good, it is possible
for a real game. Therefore, validating the assumptions with real game data is critical before applying an equal-skill based matchmaking algorithm.

When $c(s_i, s_j) = c(s_j)$, i.e., a player’s churn risk is determined by his state before matchmaking, then it does not matter whom they will play. In this case, EOMM can do no better than a random matchmaking. Random matchmaking, from this perspective, is not trivial. It is a relative safe and stable baseline choice in lack of prior information. While equal-skill based method can perform the worst under certain conditions, random matchmaking will never fall into the worst case.

The analysis above shows that the existing matchmaking methods, such as equal-skill based and random matching, arise within the EOMM framework on different conditions. Practitioners can safely apply EOMM while gathering more information about their game and players.

5.5 Case Study

To test the proposed matchmaking framework, we ran simulation which is configured based on the real data from a popular PvP game made by Electronic Arts Inc. (EA). In the simulation, we compared different matchmaking methods applied to the same player population. In the end, EOMM retained significantly higher number of players than other matchmaking methods.

5.5.1 Data Collection

We collected 1-vs-1 matches from a popular game made by EA. There are three possible match outcomes, namely Win, Lose and Draw. In total, we collected 36.9 million matches played by 1.68 million unique players in the first half of 2016.

5.5.2 Preparation

To create a realistic environment for simulation, the following models and functions are needed. We computed them based on real game data.

Player Skills We needed to establish a distribution of player skills for the population we simulate on. The distribution was learned from real game data. We sorted
the collected real matches temporally and applied Glicko (Glickman, 1999) to compute each player’s final skill. For each player $i$, the skill vector is represented by mean $r_i$ and variance $RD_i$, i.e. $\mu_i = (r_i, RD_i)$. In simulation, we assumed that the game and player skills are stationary. The population’s skill distribution is constant, where each player’s skill does not change any more over time.

While Glicko scores can be used to estimate the winning probability of player $i$ over player $j$, $Pr(i > j|\mu_i, \mu_j)$, they cannot provide the probability of draws. We defined a set of rules to allow the estimation of win/lose/draw probabilities from Glicko scores:

\[
Pr^\ast (i = j) = 20\% \tag{5.11}
\]

\[
Pr^\ast (i > j) = \frac{80\% \cdot Pr(i > j|\mu_i, \mu_j)}{Pr(i > j|\mu_i, \mu_j) + Pr(j > i|\mu_i, \mu_j)} \tag{5.12}
\]
\( \Pr^*(i < j) = 1 - \Pr^*(i = j) - \Pr^*(i > j) \quad (5.13) \)

Basically, the draw probability (Eqn. 5.11) is set to 20% regardless of skill gaps. This is based on our findings that 1) draw outcomes only have \(-0.05\) correlation with the difference of skill means in the collected game data; 2) around 20% matches are draws regardless of skill gaps. The win/lose probabilities are normalized such that the probabilities of win, lose and draw sum up to 1. Figure 5.2 shows that the predicted win probabilities using Glicko scores based on our rules are well aligned with the real match outcomes.

**Churn Prediction Model** We trained a logistic regression model for predicting whether a player will be an eight-hour churner after a match. The input features describe the upcoming match and the player’s 10 most recent matches. A player is labeled as an eight-hour churner if they do not play any 1-vs-1 match within the next eight hours after playing this match. As discussed in Section 5.3, the term of “churn” is used by convention. It represents “stopping playing” within a period of time, which is a metric of disengagement.

We use Eqn. 5.7 to estimate \( c(s_i, s_j) + c(s_j, s_i) \). The model takes as input the player’s state \( s_i \) before matchmaking along with the upcoming match outcome \( o_{ij} \).

Specifically, the input features consist of:

- *Each of the player’s 10 most recent matches*: win/lose/draw status, time passage since the previous match, time passage to the upcoming match, and goal difference against his opponent
- *Upcoming match*: one-hot encoding of the upcoming match’s outcome win/lose/draw
- *Other*: the number of 1-vs-1 matches played in the last eight hours, one day, one week and one month.

We used 5-fold cross validation and grid search to determine the proper \( L_2 \) regularization strength when training the model. The predicted probabilities were well aligned with the real churn probabilities, in particular when churn risk is less than 0.8, as shown in Figure 5.3. While the performance of the predictive model still has
5.5. Case Study

Figure 5.3: Predicted churn risk vs. real churn risk. Real churn risk is the ratio of matches, with similar predicted churn risks, which are indeed the last match before churn.

room to improve, the flexibility of EOMM allows one to easily refine or replace the model if better ones are found.

**Player States** In simulation, each player’s state was sampled from a collection of states, which were established based on real players’ states in the collected data. We first randomly sampled a subset of matches. Both players’ states in those matches were gathered to create this collection. A player state contained the needed features for churn prediction, as well as the player’s skill score.

5.5.3 Simulation Procedure

In the simulation, we compared EOMM with three matchmaking mechanisms: random matchmaking (RandomMM), which randomly pairs available players in the waiting pool, skill-based matchmaking (SkillMM), which pairs every two consecutive players after sorting them by skills, and worst matchmaking (WorstMM), which does the opposite of EOMM by minimizing the objective function of EOMM. SkillMM always seeks “fair games”. We added WorstMM as a validation.
All methods are applied on the same population (waiting pool), where the same player skill distribution, churn model and player state distribution as described in Section 5.5.2 are used. EOMM follows Eqn. 5.7 to estimate churn risk $c(s_i, s_j) + c(s_j, s_i)$. We used the perfect matching algorithm (Gabow, 1974; Lawler, 2001) implemented by an open-source library (Rantwijk, 2013).

For each matchmaking method $M$, the procedure within each round of simulation is as follows:

1. Create a waiting pool of $P$ players, whose player states are sampled from the player state collection.
2. Use $M$ to determine the pair assignment (matchmaking).
3. Simulate match outcomes according to the win/lose/draw probability predicted by the skill model.
4. For each player, simulate if he will churn according to the predicted churn probability by churn model.
5. Record the number of retained players.

In experiments, we tested $P = 100, 200, 300, 400$ and $500$. For each setting of $P$, we repeated the simulation by 10,000 rounds of matchmaking. We compared different matchmaking methods by the average number of their retained players per round, i.e., the players who continue playing in the next eight hours. In order to test statistical significance, we conducted Welch’s $t$-test between every pair of the matchmaking algorithms.

5.5.4 Results and Discussion

The results are shown in Table 5.2. All pairwise differences of retained players are statistically significant ($p$-value < 0.01) except EOMM vs. RandomMM (when $P = 100$) and SkillMM vs. RandomMM (when $P = 400$). In all other scenarios, EOMM outperforms the other three matchmaking methods. The results prove the applicability of EOMM to act as an engagement optimizer. When $P = 100$, EOMM does not retain a significantly higher number of players than RandomMM, and even
Table 5.2: Average number of retained players per round of matchmaking simulation. 10,000 rounds of matchmaking were simulated.

<table>
<thead>
<tr>
<th>Method</th>
<th>P=100</th>
<th>P=200</th>
<th>P=300</th>
<th>P=400</th>
<th>P=500</th>
</tr>
</thead>
<tbody>
<tr>
<td>WorstMM</td>
<td>51.50</td>
<td>103.39</td>
<td>154.57</td>
<td>206.65</td>
<td>258.21</td>
</tr>
<tr>
<td>SkillMM</td>
<td>52.52</td>
<td>103.96</td>
<td>156.05</td>
<td>207.43</td>
<td>259.24</td>
</tr>
<tr>
<td>RandomMM</td>
<td>51.81</td>
<td>103.97</td>
<td>156.09</td>
<td>207.09</td>
<td>259.65</td>
</tr>
<tr>
<td>EOMM</td>
<td>51.90</td>
<td>104.24</td>
<td>157.50</td>
<td>209.37</td>
<td>261.19</td>
</tr>
</tbody>
</table>

retains fewer players than SkillMM. It is possibly because that when \( P \) is small, the randomness has higher impact, and also, the room for arranging opponents is smaller. More rounds of simulations might be needed to show significance in this case.

The improvement of EOMM over SkillMM, the most common matchmaking method, in terms of the average number of retained players are 0.3%, 0.9%, 1.1%, and 0.6% when \( P = 200, 300, 400 \) and 500 respectively. On average EOMM retains 0.7% more player compared with SkillMM after one round of matchmaking. Notably the benefit of retention will accumulate over time in a constant population. For players who play 20 rounds of matchmaking games within eight hours, there will be 15% more players retained \((1.007^{20} \approx 1.15)\) by EOMM over those by SkillMM. The more rounds of matchmaking are conducted, the more significant is the accumulative advantage of EOMM in engagement.

We did not find a consistent climb in retention boost as \( P \) increased. This may suggest that when the player pool reaches certain size, the choices of opponents are enough to rescue those players on the edge of churn. Beyond this size, a larger player pool may not bring in significantly extra benefits in engagement maximization.

As a validation, WorstMM consistently retains the fewest players in the pools of all sizes. This result verifies the optimum of EOMM from the opposite side. It is also interesting to note that SkillMM does not consistently outperform RandomMM, which is aligned with our discussion in the theoretical findings in Section 5.4, that is, balanced matches are not always optimal for engagement.
5.6 Summary

This chapter attempts to answer R.S.Q. 2 by presenting a novel framework, Engagement Optimized Matchmaking (EOMM), to achieve optimized engagement for a population of online players through recommendation of opponents. It formulates matchmaking as a problem of maximizing the player engagement, and solves the optimization efficiently. EOMM employs three components, a skill model, an engagement predictive model and a minimum weight perfect matching algorithm, each of which can be tailored flexibly for specific applications. We ran simulations whose configurations were based on real data from an online PvP game. The results show that EOMM significantly outperforms all other methods in the number of retained players. EOMM also provides a theoretical framework to analyze various matchmaking algorithms.

EOMM provides a measurable and flexible matchmaking framework. It has well-defined quantitative objectives that can be monitored, evaluated and improved. Within the EOMM framework, the core building components, skill model, churn model and graph pairing model, are uncoupled so that they can be tuned and replaced independently. Moreover, we can even change the objective function to other data-driven metrics of player engagement, such as play time, retention, or spending. EOMM allows one to easily plug in different types of predictive models to achieve the optimization.
Chapter 6

Applicability, Application and Ethical Issues

This dissertation opens doors to many kinds of future applications. Match-based video games, either for serious or entertaining purposes, can benefit from using our proposed systems to boost player engagement. However, practitioners should also be aware of conditions when the proposed recommendation systems are better applied than other existing systems. In this chapter, we list applicability conditions for each of our systems (Q-DeckRec, DraftArtist, and EOMM). We will also list several other hypothetical use cases different than the games on which our systems were experimented. At last, we try to address ethical issues while applying our systems.

6.1 Applicability and Application

6.1.1 Q-DeckRec

Q-DeckRec aims to present winning-effective starting items to novice players, potentially boosting their winning probabilities and consequently improving their competence and engagement. The first applicability condition of Q-DeckRec is that a match-based game offers a very large space of in-game elements that render other methods inefficient or intractable. As an example, in Hearthstone introduced in Chapter 3, the number of possible decks is $O(N^D)$ where $N$ is the number of total cards and $D$ is the deck size. This number grows gigantically easily as the game becomes complex. Heuristic searches require domain knowledge that is labor-intensive to build,
and metaheuristic searches like Genetic Algorithm would take more than 20 minutes to obtain winning-effective recommendations, which is not a tolerable time to solve each recommendation task in large-scale or real-time application (analyzed in Section 3.3). When heuristic or metaheuristic searches do not work, Q-DeckRec could be a solution to real-time or large-scale recommendation.

Another applicability condition of Q-DeckRec is that the winning-effectiveness of in-game elements could be evaluated using computational resources neither too heavy nor too light. As in our study, we used match simulations to evaluate \( f(\cdot) \), the win rate between any two decks. The time needed by \( f(\cdot) \) is roughly 5 seconds as reported in Section 3.5, which is acceptably not too heavy because it makes the overall training time limited within 3 days, an tolerable time for offline training before the system goes online or between consecutive patches of the game. On the other hand, if the computational resources needed to evaluate \( f(\cdot) \) is extremely light to the extent that applying meta-heuristic searches in real-time is affordable, Q-DeckRec might not be worthy to implement.

One hypothetical use case of Q-DeckRec, besides building decks in Collectible Card Games, is to recommend items in match-based combats in a large and complex Role-Playing Game (RPG), one like the popular title Diablo III (Blizzard Entertainment). In such a game, the player assumes to act out a character in a fictional setting, and the character could equip with a large number of items, from defensive items like armors and helmets, to offensive weapons like swords and axes. The game is complex because there could be thousands of different items falling into the same item category. For example, in Diablo III, the weapon category axes is divided into 37 kinds of one-handed axes and 27 kinds of two-handed axes. Even the same kind of axes could have different “instances”, each differing in exact numbers of damage per second and specific magical properties. A magical property gives a bonus to an attribute of the character (e.g., collecting golds 15% faster) or other equipped items (e.g., bonus attack happens when two weapons work together). We suppose that there is a one-on-one, match-based combat mode, much like the brawling mode in Diablo III, where two players can first equip their characters freely and then

\[\text{According to } \text{https://us.diablo3.com/en/item/#filter=wizard}\]

\[\text{Due to various reasons, this mode was no longer supported after 2013.}\]
join a designated arena for battling. A player could collect a variety of items but only equips a subset of them that are most powerful against the specific opponent. One can imagine that though match-based combats is not the only element in RPGs, frequent losses in combats could easily incur frustration and disengagement. \textit{Q-DeckRec} could be applied in this case to recommend items that are winning-effective in the upcoming combat. Similarly to the procedure in Chapter 3, the practitioner should first design sparse vectors encoding the items players equip (corresponding to $x_p$ and $x_o$ in Eqn. 3.1). The practitioner should also prepare a range of AI proxies that imitate play styles of different players (corresponding to $A_p$ and $A_o$ in Eqn. 3.1). We expect that in this case \textit{Q-DeckRec} will boost the winning chance of novice players who frequently fail match-based combats due to inexperience in equipping items, dragging them away from the blink of disengagement.

6.1.2 DraftArtist

\textit{DraftArtist} recommends winning-effective in-game elements in multi-player settings, where multiple players’ decisions on in-game elements are unfolded in a sequential order. Applicability conditions for \textit{DraftArtist} are three-folds. First of all, the sequence of in-game element choices is influential on the match outcome. In DOTA 2 where \textit{DraftArtist} is tested, the sequence of character drafting is crucial in determining team synergy and counter effects. The opponent team can take advantage of careless planning in the drafting. \textit{DraftArtist} is not suitable when

Second, the choice space of in-game elements is large enough that other methods do not yield good results. For example, in DOTA 2, the estimated number of total possible hero line-ups is $1.56 \times 10^{16}$. Association rule-based heuristics or the baseline of picking the highest win rate hero could not lead to teams as effective at winning as those formed by more sophisticated planning by \textit{DraftArtist}, as shown in Table 4.3 and 4.5.

Third, the winning-effectiveness of final choices of in-game elements after the sequential picking phase can be evaluated and evaluated quickly enough for a number of simulations in the tree search in \textit{DraftArtist}. As in our study, we used a machine learning model to evaluate the win rate of a character line-up with acceptable time such that \textit{DraftArtist} can output winning-effective recommendations in 1.12 seconds.
90  Chapter 6. Applicability, Application and Ethical Issues

or less (Table 4.4), which is much shorter than normal time limit for human players to draft a hero (25 seconds).

*DraftArtist* can be applied in other game genres with a sequential picking phase of in-game elements. One hypothetical use case is a multi-player fighting game, which could be an extension from the popular one-on-one title *Street Fighter* (Capcom) to the multi-player setting. This game features numerous fighting characters with different fighting skills and sophisticated fighting skill oppositional relationships among those characters. Two teams, each consisting of five players, alternate to select fighting characters, while no character can be selected twice by either team. After both teams finish drafting the fighting characters, the order of appearance of those fighting characters in the following matches are determined. The first match is between the first-drafted fighting characters of the two teams. A match ends whenever a fighting character is knocked out, then the fighting character who survives will begin the next match with the next drafted fighting character in the opponent team. Whichever team has any fighting character survived in the last wins. The pool of fighting characters can be potentially large, which goes beyond novice players’ abilities to estimate what order of appearance of fighting characters can be advantageous. *DraftArtist* can be applied here to recommend fighting character drafts. As in Chapter 4, the practitioner needs to formulate their recommendation problem as a combinatorial game, with a reward function predicting the game outcome probability at the end of the drafting. The reward function takes as input the sequence of appearance of fighting characters drafted by both teams and outputs the winning probability predicted for both teams. With a proper formulated game, the practitioner can kick off Monte Carlo Tree Search in *DraftArtist* in real time to serve recommendation requests.

### 6.1.3 EOMM

EOMM is a bit different from the other systems and has its own scope of applications. As modern match-based video games typically have different game modes that give players different mentalities, EOMM is especially suited for casually competitive modes but not for game modes meant to be strictly competitive. We use
ranked and non-ranked game modes to illustrate the difference between strict competitiveness and casual competitiveness, as the two game modes are often observed in modern match-based video games. Ranked game modes usually maintain a public profile for each player; a player can improve the statistics of the profile by winning over similarly skilled players, however the profile sinks if the player loses. Achieving a high performing ranked profile endows the player proud and fulfillment among peers as it is akin to a certificate of the player’s skill. Thus, players in the ranked mode can be extremely cared about match outcomes, usually playing with a high level of competition, concentration and aggressiveness. Meanwhile, players in the non-ranked mode can practice and discover the game freely or play together with friends without concern about negative match outcomes damaging their public profiles. The goal of EOMM is to arrange players for the optimal engagement rather than the smallest skill gaps; therefore EOMM is not suited for strictly competitive game modes like the ranked mode. However, players in the non-ranked game expect opponents from various skill levels, thus applying EOMM would not contradict with their mental models.

EOMM can serve for different engagement metrics. In the case of our illustration the churn risk of all involving players (Eqn. 5.1) is chosen as the engagement metric. As the way EOMM is designed, the practitioner can plug in other engagement metrics such as play time or spending. Once the engagement metric is defined, the practitioner should train a corresponding model for predicting the engagement metric among the players in one match, which will be used in estimating edge weights in the matchmaking graph constructed by EOMM.

As competition is a common element introduced in serious games, it is easy to imagine a use scenario where a match-based crowdsourcing game would need a matchmaking system like EOMM to improve player engagement as a way to increase the number of completed tasks. To be more specific, this game can be similar to a one-vs-one image labeling game introduced in “Games with a purpose” (Von Ahn, 2006). The motivation of this game is that image labels generated by humans are more accurate than existing algorithms and can facilitate many other image learning tasks. In the game, two paired players locate image areas that are associated with given labels through designed match competition mechanisms, while the
specific competition mechanism may vary game by game. Different players have different “states” (i.e., a collection of features that profile an individual player, such as match history, skill, and play frequency, as described in Chapter 5) before the upcoming matches. Some players may feel bored because of the opponents are less challenging in the past few matches and thus look for more challenging opponents; while some players may feel frustrated by losing in a row. To keep players engaged in playing the game and completing more tasks, EOMM can be applied to match players, with the optimization objective set as the churn risk. The practitioner would need to implement the three components of EOMM, namely skill model, churn model and graph pairing mode. Each component can be tailored to specific needs of the image labeling game. For example, the practitioner may want to use skill models specifically tailored for crowdsourcing tasks (Mavridis, Gross-Amblard, and Miklós, 2016; Rahman et al., 2015).

6.2 Ethical Issues

While our proposed systems have a broad horizon of applications, we are deeply aware of many ethical issues caused by misuse of our proposed systems. Hence, we want to raise them to practitioners and propose several suggestions to address them.

The first ethical issue is addiction. While the intent of our recommendation systems is to engage players, it remains unclear whether overuse of them would lead to addiction. Recent research found that high engagement seems to relate to the so-called peripheral addiction criteria, which is defined as (cognitive salience, tolerance, and euphoria). On the other hand, addiction is characterized by the core addiction criteria (conflict, withdrawal symptoms, relapse and reinstatement, and behavioral salience) (Charlton and Danforth, 2007). Researchers have found that high engagement could be a stepping stone towards addiction (Charlton and Danforth, 2007), while addiction is considered to be pathological, causing higher risk of psychological problems (Lehenbauer-Baum et al., 2015; Brunborg et al., 2013) and reduction in social relationships (Kraut et al., 2002; Blais et al., 2008). To address the potential of our systems for “excessively” engaging players, we suggest
that practitioners integrate detection mechanisms to distinguish between highly engaged and addicted players (Charlton and Danforth, 2007; Fisher, 1994; Griffiths and Hunt, 1998; Brown, 1997; Griffiths, 1996; Seok and DaCosta, 2014), and adopt regulations immediately when addiction arises. Another suggestion is to regularize the optimization objective (Eqn. 5.1) in EOMM with some quantitative measurement of addiction level. As such, the optimization does not simply optimize player engagement but also balances the time of play to prevent addiction.

The second issue is whether our proposed systems may become pure monetary tools for game companies to increase their profits, while players’ game experience is sacrificed. Game companies may use winning-effective in-game recommendation described in Chapter 3 and 4 with a priority to increase profit from players. However, we note that using the proposed approach while they improve the recommendee’s winning chance and potentially lead to better engagement, they may also lower the winning chance and engagement of the recommendee’s opponents. Game companies may also use profit-related metrics, such as spending, as the optimization objective in the matchmaking framework described in Chapter 5. To combat this issue, we would suggest more regulation on how game companies employ these systems and urge game companies to reveal how their systems work in a proper manner, which leads to the next issue.

The third issue is how much information regarding algorithmic control introduced by one of these three systems should be revealed to players. On the one hand, game developers want to engage players in an implicit way because disclosing too much information may interrupt the game experience or spoil the original intention of engaging players. On the other hand, treating our systems totally as a black box and providing no explanation to the underlying mechanisms may create mistrust and make players feel manipulated. Indeed, “calibrating” transparency is always a challenge of any Artificial Intelligence system (Bostrom and Yudkowsky, 2014; Ananny and Crawford, 2018; Scherer, 2015). So far, we think the best way for game developers and companies going forward is to conduct honest discussion with the player community to determine the best level of transparency.
Chapter 7

Limitation and Future Work

At a high level, we envision several future directions for continuing the research in in-game element recommendation systems for improving player competence and player engagement. First, we are eagerly looking forward to testing our systems online in a systematic way. Currently, none of our proposed systems have been thoroughly tested online for player engagement. Admittedly, this would require deep cooperation with game developers and companies who support A/B testing on existing games.

Second, besides starting items, characters, and opponents, we would like to investigate recommendation systems on different kinds of in-game elements, including but not limited to weapons, maps, play strategies, teammates, and buildings. Each kind of in-game elements may require specific design of recommendation techniques.

Third, we did not investigate when we should deliver winning-effective recommendations. We expect that these recommendations be triggered by some kinds of an alert model, which determines which players are in need of help or are about to quit out of frustration. However, more empirical experiments need to be conducted on recommendation systems integrated with such alert models. Moreover, the delivery of winning-effective recommendation should also depend on the type of the opponent, i.e. is it a human player or an AI program? When the opponent is a human player, winning-effective recommendation should not jeopardize the opponent’s competence and engagement, but instead mutually benefit the opponent who happens to look for more challenge. This means the alerting model being used should monitor both parties’ information.
Lastly, we would like to design a unified recommendation system which could coordinate recommendations of various in-game elements for improving overall engagement of each player. This comes from our intuition that multiple in-game elements could simultaneously influence a player’s engagement and recommendation on one type of in-game element could influence the effect of recommendation of other in-game elements. More research is needed to understand the interrelationships of recommendations across different in-game elements and their influence on the player’s overall engagement. To the best of our knowledge, such an idea has not been explored before.

We also specify limitations of each of our proposed systems and suggest how to address them in the future.

The limitations of Q-DeckRec are:

1. Q-DeckRec relies on AI proxies which are supposed to accurately model players’ play styles. The current used AI proxy is only based on a greedy heuristic rather than trained on human play traces. Therefore, the optimal decks obtained in our experiments cannot be directly recommended to human players. Training human-like AI proxies and integrating them to Q-DeckRec will be an important direction in our future works.

2. Q-DeckRec currently only recommends winning-effective decks against specific known opponents, which requires to access the opponent’s information, including play style and the deck already built. However, in certain match modes the opponent will not have been determined at the time of deck building. In such situations, Q-DeckRec can only recommend winning-effective decks against a hypothetical opponent with a predicted deck and play style.

3. We can also improve sample efficiency of Q-DeckRec. Currently, each training episode starts with a random initialized state. Were it generated from a card distribution learned from real matches, Q-DeckRec might focus on exploring in a smaller but more useful state space.

4. As online CCGs often release patches to introduce new cards and modify existing cards’ in-game effects, we would like to investigate how Q-DeckRec can
transfer and update its knowledge without totally re-training the model (Taylor and Stone, 2009).

5. We have only investigated the deck recommendation problem towards a single opponent. There remains a question of how to design the feature representation of state-action pairs in Q-DeckRec if the problem is extended to recommend winning-effective decks against a group of opponent decks. Naive feature representations for the opponent deck group could be simply concatenating each of the opponents’ deck representation vector in the group. However this creates a large feature space which may not be efficient for learning. A more advanced feature representation may represent the opponent deck group in a continuous vector space, similar to word-embedding techniques from Natural Language Processing (NLP) (Mikolov et al., 2013). We intend to investigate all of these feature representation approaches in the future.

The limitations of DraftArtist are:

1. We have not considered player-specific information, such as player skills in their selected characters, when recommending characters. It is possible that a character recommended by our algorithm, which is based solely on the current hero line-up, may not be played well by a player who is not familiar with it. Our algorithm can be extended to integrate player skills, by augmenting the game state with player information and training a more advanced win rate predictor as the reward function which takes as input both hero picks and player-specific information, once we have access to needed player-specific data.

2. Were hero pick sequences from real match data available, we would integrate them as prior information to improve the tree policy and default policy in MCTS, thereby improving capabilities to build search trees more effectively and efficiently (Gelly and Silver, 2007; Chaslot et al., 2009).

3. We would also like to investigate how our recommendation systems can be customized to account for additional drafting rules and extended to other real-world scenarios such as player drafting in sports (Staw and Hoang, 1995).
Chapter 7. Limitation and Future Work

The limitations of EOMM are:

1. So far we have discussed EOMM in 1-vs-1 game scenarios. This framework also applies to match-based games that involve teams of players, where every component needs to be extended with additional care. The skill model can be simply applied to a team by adding up skills for all team members (Herbrich, Minka, and Graepel, 2006). For churn prediction, we can use the same idea that one player’s churn risk is conditionally independent with other players’ states given that their influence on the player’s own state, such as the game outcome, is known. Last, the minimum weight perfect matching algorithms for pairs are no longer applicable. Instead of a pair assignment, we seek a grouping assignment on a complete graph. A related area to investigate is perfect matching in hypergraphs (Berge, 1984), where an edge can connect more than two vertices.

2. EOMM is a greedy algorithm in the sense that it optimizes player engagement in the next match. However, we have not considered that player engagement could be fostered through multiple matchmaking outcomes. This requires viewing matchmaking through sequential decision making (Littman, 1996), in which the next match’s outcome could be just a step leading to the optimal outcome sequence.

3. We expect EOMM equipped with more advanced models, such as skill models and churn models, can have higher optimal bound. We will explore EOMM performance in more realistic situations, where practical restrictions are applied, such as network connectivity, regions and friend/black lists. More restrictions would result in fewer edges in the constructed graph of EOMM and perhaps faster algorithms for solving minimum perfect weight matching.
Chapter 8

Conclusions

This dissertation describes three in-game element recommendation systems for starting item selection, character selection, and opponent selection, designed to be used in the pre-match stage to indirectly or directly improve player engagement based on theoretical foundations or data-driven approaches. They are a proposed solution to the fundamental research question: How can we design in-game element recommendation systems working in the pre-match stage, which can improve player engagement in match-based video games? The overall contribution of this dissertation is the enrichment of an arsenal of recommendation techniques that can recommend pre-match in-game elements which are either winning-effective or positively influential in player engagement measured quantitatively.

The starting item and character recommendation systems, within the context of one-vs-one and team-vs-team settings respectively, attempt to identify winning-effective in-game elements efficiently from a large amount of candidates. Their usefulness is based on the assumption that presenting winning-effective in-game elements to players could improve their competence. This is also based on theories, such as Self-Determination Theory and Flow Theory, which postulate that player competence is linked to engagement. Thus, the starting item and character recommendation systems can be seen as an indirect approach to improve player engagement. Experiments were conducted and demonstrated that the proposed systems are able to recommend equally or more winning-effective in-game elements than previous methods with computational resources efficient enough for large-scale or real-time usage.
The third system aims to directly optimize player engagement through match-making, i.e. recommendations of opponents. We defined player engagement quantitatively as churn risk and rely on a data-driven approach to determine the recommendation of opponents that is optimal for the engagement metric. As far as we know, this is the first system that has formally treated matchmaking as an optimization problem for quantitatively measured player engagement. We built a simulated system using real game data, showing significant advantages of the proposed matchmaking system in retaining players over existing methods.
Bibliography


Browne, Cameron B et al. (2012). “A survey of monte carlo tree search methods”. In: IEEE Transactions on Computational Intelligence and AI in Games 4.1, pp. 1–43.


Chen, Yutian et al. (2016a). “Learning to Learn without Gradient Descent by Gradient Descent”. In:


Cooper, Seth et al. (2010). “Predicting protein structures with a multiplayer online game”. In: Nature 466.7307, p. 756.


Edmonds, Jack (1965a). “Maximum matching and a polyhedron with 0, 1-vertices”.


Graepel, Thore and Ralf Herbrich (2006). “Ranking and matchmaking”. In: Game Developer Magazine 25, p. 34.


Hadiji, Fabian et al. (2014). “Predicting player churn in the wild”. In: IEEE Conference on Computational Intelligence and Games (CIG). IEEE, pp. 1–8.

Harrison, Brent E and David L Roberts (2012). “When Players Quit (Playing Scrabble).” In: AIIDE.


Lazzaro, Nicole (2004). “Why we play games: Four keys to more emotion without story”. In:


Littman, Michael Lederman (1996). “Algorithms for sequential decision making”. In:

Lombard, Matthew and Theresa Ditton (1997). “At the heart of it all: The concept of presence”. In: *Journal of Computer-Mediated Communication* 3.2, pp. 0–0.

Looi, Wenli et al. (2018). “Recommender System for Items in Dota 2”. In: *IEEE Transactions on Computational Intelligence and AI in Games*.


Palmgreen, Philip, Lawrence A Wenner, and Karl E Rosengren (1985). “Uses and gratifications research: The past ten years”. In:


Runge, Julian et al. (2014). “Churn prediction for high-value players in casual social games”. In: *IEEE Conference on Computational Intelligence and Games*. IEEE, pp. 1–8.


Santos, André, Pedro A Santos, and Francisco S Melo (2017). “Monte Carlo tree search experiments in hearthstone”. In: *Computational Intelligence and Games (CIG)*. IEEE, pp. 272–279.


Shiyko, Mariya et al. (2016). “Effects of playing a serious computer game on body mass index and nutrition knowledge in women”. In: JMIR serious games 4.1.


Weber, Ben G, Michael Mateas, and Arnav Jhala (2011). “Using data mining to model player experience”. In:


Xie, Hanting et al. (2015). “Predicting player disengagement and first purchase with event-frequency based data representation”. In: Computational Intelligence and Games (CIG), 2015 IEEE Conference on. IEEE, pp. 230–237.


