POLITECNICO DI MILANO
Master Degree in Computer Science and Engineering
Department of Electronics, Information and Bio-engineering (DEIB)

Improving Automatic Playtesting through Gameplay Simulation with Player Modeling Techniques

Internal Supervisor: Prof. Pier Luca Lanzi
Politecnico di Milano

External Supervisor: Prof. Pawel Herman
Royal Institute of Technology

Industrial Supervisor: Stefan Freyr Gudmundsson
King Digital Entertainment Ltd

Master Thesis by:
Davide Anghileri
Matricola N. 879194

Academic Year 2017-2018

The important thing is not to stop questioning.
Curiosity has its own reason for existing.
Albert Einstein

Abstract

In this thesis we present two approaches to improve automatic playtesting using player modeling. By modeling various cohorts of players we are able to train Convolutional Neural Network based agents that simulate human gameplay using different strategies directly learnt from real player data. The goal is to use the developed agents to predict useful metrics of newly created game content.

We validated our approaches using the game *Candy Crush Saga*, a non-deterministic match-three puzzle game with a huge search space and more than three thousand levels available. To the best of our knowledge this is the first time that player modeling is applied in a match-three puzzle game. Nevertheless, the presented approaches are general and can be extended to other games as well. The proposed methods are compared to a baseline approach that simulates gameplay using a single strategy learnt from random gameplay data. Results show that by simulating different strategies, our approaches can more accurately predict the level difficulty, measured as the players' success rate, on new levels. Both the approaches improved the mean absolute error by 13% and the mean squared error by approximately 23% when predicting with linear regression models. Furthermore, the proposed approaches can provide useful insights to better understand the players and the game.

Keywords — Player Modeling; Automatic Playtesting; Gameplay Simulation; Convolutional Neural Network.

Sommario

La creazione di contenuti nei videogiochi è un compito di cruciale importanza per sviluppare e mantenere alta la qualità del gioco proposto. In particolare, i livelli nei videogiochi devono creare divertimento e soddisfazione nei giocatori. Se un livello è troppo facile rischia di annoiare il giocatore ed al contrario se è troppo difficile rischia di creare frustrazione. In entrambi i casi il giocatore potrebbe decidere di abbandonare il gioco. Essere in grado di creare contenuti di gioco che soddisfino le aspettative dei giocatori è un compito non semplice che richiede esperienza e professionalità. L'intelligenza artificiale è uno strumento che può aiutare i designer di videogiochi nell'ottenere utili metriche riguardo i contenuti creati. In questa tesi proponiamo due approcci che attraverso la creazione di modelli dei giocatori e la simulazione di gioco permettono di ottenere metriche più accurate riguardo i contenuti di gioco creati dai designers.

Abbiamo validato gli approcci proposti utilizzando come esempio il videogioco *Candy Crush Saga*, un videogioco non deterministico di tipo puzzle match-3 con più di tremila livelli disponibili e milioni di giocatori attivi ogni giorno. Inoltre, gli approcci proposti sono generali ed è possibile utilizzarli anche in altri tipi di videogiochi. I risultati provano che è possibile ottenere stime riguardanti la difficoltà di nuovi livelli in modo più accurato rispetto al caso in cui non si consideri nessun modello dei giocatori. Infine, attraverso l'utilizzo di questi modelli è possibile sviluppare analisi che permettono di capire in modo più approfondito non solo i giocatori ma anche il videogioco stesso.

Acknowledgments

I would like to thank my academic supervisor, Prof. Pier Luca Lanzi at the Department of Electronics, Information and Bioengineering (DEIB) at the Politecnico of Milan (Polimi), for the help and support through the writing of this thesis. I want to express my gratitude to Prof. Pawel Herman at the school of Computer Science and Communication at the Royal Institute of Technology (KTH) for the helpful feedback and discussion through the research and writing of this thesis. Furthermore, I wish to express a sincere thank to my industrial supervisor at King, Stefan Freyr Gudmundsson, for his continuous support, advice and discussions throughout the journey of this research. His guidance and extensive knowledge helped me in all the time of writing and research of this thesis.

Furthermore, I want to thank my colleagues in the Artificial Intelligence team at King, Marcus Adamsson, Rachael Ann, Philipp Eisen, Erik Poromaa, Alex Nodet, Sami Purmonen, Vivek Grimm, Christian Schmidli and Lele Cao for their support that helped me to overcome the obstacles I faced in this research. Especially, I want to say a particular "thank you" to Philipp Eisen who set the foundations for this research and to Marcus Adamsson for all the fun we have had in the period of this thesis. Moreover, I would like to thank Bartlomiej Kozakowski, for the help with statistical analysis and the interesting feedback on my research.

Finally, I want to express my very profound gratitude to my family that helped me reaching this milestone of my life and I want to direct a special "thank you" to my girlfriend Daiana Concilio, for the motivation, patience and continuous encouragement throughout my years of study and the process of writing of this thesis. This accomplishment would not have been possible without them.

Milan, June 28, 2018
Davide Anghileri

Contents

List of Figures

List of Tables

Acronyms

AI	Artificial Intelligence.
AP	automatic playtesting.
CNN	Convolutional Neural Network.
DCNN	Deep Convolutional Neural Network.
ELU	exponential linear unit.
F2P	Free-to-Play.
FPS	first-person shooter.
GAP	global average pooling.
GDPR	General Data Protection Regulation.
GLM	Generalized Linear Model.
ILSVRC	ImageNet large scale visual recognition challenge.
MAE	mean absolute error.
MCTS	Monte Carlo tree search.
MSE	mean squared error.
NN	artificial neural network.
np-hard	non-deterministic polynomial-time hard.
OLS	Ordinary Least Squares.

P2P Pay-to-Play.

PCA Principal Component Analysis.

PM player modeling.

ReLU rectified linear unit.

RL Reinforcement Learning.

SL Supervised Learning.

SR success rate.

SSE sum of squared errors.

UL Unsupervised Learning.

UPGMA unweighted pair group method with arithmetic mean.

vCPU virtual central processing unit.

WCSS within-cluster sum of square.

Chapter 1

Introduction

Within the recent years, we have seen an incredible growth in the gaming industry. Newzoo, a leading provider of games market intelligent, reported that "the gaming industry is growing faster than expected, up 10.7% to $116 billion 2017" [1]. Most of the revenue comes from the mobile industry that accounts for almost half of the global game market and the perspectives show a continuous growth at an annual rate of 8.2% to 2020. Everyday, more and more people use their smartphone to play mobile games and a recent report on Verto Analytics [2] showed that gamers play an average of 24 minutes each day. Being able to continuously engage users is a key capability that game companies need to possess. Another current trend is the shifting from the traditional Pay-to-Play (P2P) business model to the Free-to-Play (F2P) or "freemium" business model. The F2P business model allows users to acquire and play a game free of charge and at the same time encourages them to pay for in-game additional content. More than that, it allows game companies to increase the popularity of a game and to collect revenues in a second moment. This fundamental shift is even more preeminent in the mobile game industry as demonstrated by the success and highly profitable results of many F2P mobile titles (see e.g. [3]–[8]).

To exploit the F2P business model, game companies need to continuously release new content (e.g. levels, characters, items to collect) or add new features to engage users and retain them as long as possible. It is extremely important that the content generated by game designers matches the expectations of the players [7]. In this direction, one of the most important features is the user´s perceived difficulty of a game [9]. If a game is too hard, the player gets frustrated and contrariwise if a game is too easy the player gets bored. Being able to balance the difficulty of a game [10], and so its perceived quality, is the primary challenge that game designers need to deal with every day. Sometimes, game designer's intuition is not enough to balance a game, especially for those features whose impact on the difficulty is not obvious. A common approach to balance a game is to iteratively test and adjust game content, like a level or a character, until a predefined design goal is achieved.

The focus of this research is on understanding and testing game content, leaving the creation process to game designers. Human testing has been widely used in the past but at the same time, it is expensive, time consuming and relies on subjective feedback [11]. To solve this problem several approaches have been proposed, leading to what is called automatic playtesting [12].

1.1 Background

The work of this thesis can be classified under the broad category of Artificial Intelligence (AI). AI is understood as the study and development of "intelligent agents". Agents that mimics *cognitive* functions of humans, like reasoning, learning or memory. Since its foundation at a workshop at Dartmouth College in 1956, AI experienced phases of interest and funding as well as phases of critics and disappointments as demonstrated by the two major "AI winters" in 1974–80 and 1987–93. Since then, as machines have become more powerful, the definition of which task requires "knowledge" and as a consequence, what makes an agent "intelligent", has changed and it still continue to change. Nowadays, AI is widely applied in different fields and disciplines and examples of actual challenges are: autonomous driving, executing medical diagnosis or competing with humans in games. Since the birth of AI, games and especially board games, have been a popular domain of research [13] due to the formal and constrained environments, yet complex decision making problems. The combination of game and AI is demonstrated by the increasing popularity of meetings, like the *Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*, the *Conference on Computational Intelligence and Games (CIG)* and the *Game Developers Conference (GDC)*, or journals, like the *Transactions on Games (ToG)*, that continuously presents the major results in the field. Within the AI area, this thesis involves three major categories: automatic playtesting, player modeling and gameplay simulation.

In the last years, automatic playtesting has become more and more popular thanks to the increasing performances of computational resources and the notable results in the machine learning field. Zook *et al.* [11] used active learning techniques to reduce the amount of playtesting needed to adjust game parameters. They reduced the cost of playtesting by intelligently picking the parameters to test, but they still used human testing to evaluate various parameter configurations. Isaksen *et al.* [14] used automatic playtesting and player modeling to tune game parameters to reach a predefined difficulty. They implemented a simple AI agent that performed better than humans in an action game that requires only motor skills but no strategy or planning. Then, they simulated humans by adding dexterity, reaction time and accuracy errors. Finally, they used survival analysis to explore the game space by adjusting low-level parameters of the game.

Holmgård *et al.* [15] used linear perceptron and Reinforcement Learning (RL) to simulate a priori defined styles of playing. While later, [16] they compared two different methods of representing human decision making styles. They illustrated that a top-down approach from expert knowledge performs equally good as a bottom-up approach from human play-traces. However their analysis is limited to a simple dungeon exploration game and there is no evidence that the same results apply for different or more complex games. Isaksen *et al.* [17] used variants of Q-learning-based agents to simulate several types of players with different skills. Subsequently they used the trained agents to estimate the strategy and dexterity required on each level. Drachen *et al.* [18] used self-organizing maps to construct models of players and inform game designers if players interact with the game as intended. Hoorn *et al.* [19] used multiobjective evolutionary algorithms to train an agent that not only performs good in the game but at the same time imitates human players. They concluded that imitating humans is a hard task and as a fact, their method performed poorly on this task. Eisen [20] used a Convolutional Neural Network (CNN) to predict human moves in the *Candy Crush Saga* game. Consequently he used the CNN-based agent to simulate gameplay and finally, using the agent performances he estimated difficulty of new levels. However, each player has his own way of reasoning and a game content can be perceived difficult by one player and easy by another one. As a consequence, modeling different types of players' behaviour can be beneficial to understand how different people interact with the game and which difficulties they face.

1.2 Problem

In the literature, many variants of playtesting have been discussed and investigated. However, some of the proposed approaches [19]–[25] consider difficulty as a single measure while, indeed, difficulty can be perceived differently by various type of players. Instead others [12], [14], [15], [17], [26] model different AI agents to simulate different players but they do not consider real player data while learning the policies and rely on designers' knowledge or RL to differentiate the agents. We believe that using player modeling to simulate strategies of various cohorts of players while directly learning the polices from real player data can led to both a closer to human and heterogeneous simulation. Therefore, we can have a better estimate of the quality of a game, helping game designers to faster and more accurately develop the desired content.

To validate our idea, in this research we use the *Candy Crush Saga* game as an example. However our approach is general and can be adapted to work with other games too. At the moment of writing, there exists an implementation of a Convolutional Neural Network (CNN)-based agent that simulates human gameplay to predict the player success rate (SR) for the selected game [20]. Since this method reached the best performance to date, we will use it as a baseline. However, the existing method uses only an average policy learnt from a random subsample of players, without considering any difference in the skills or strategies of different cohorts of players, nor any other information concerning them. We believe that being able to incorporate these characteristics of the players into the model allows to improve the prediction accuracy of the network as well as providing useful insight about the behaviour of the players and the characteristics of the levels. The measure used to evaluate the difficulty of a level in this thesis is the SR. As a consequence, the research question examined is:

Can player modeling improve the players' SR estimation of a CNN-based agent simulating human gameplay?

1.3 Goals and Purpose

The thesis illustrates how player modeling techniques can be used to model strategies and skills of different groups of players by simulating them using CNN-based agents. The primary goal of the thesis is improving automatic playtesting using player modeling techniques. We illustrate how the developed agents can be useful for predicting the human SR on new levels for the game selected as test bench in this research. Subsequently, we aim to provide game designers with level difficulty measures perceived by the different groups of players to iteratively balance game content and create the best possible player experience. Furthermore, this work illustrates how the new approach outperforms the previous state-of-the-art [20] in the human SR estimation task. We discuss how simulating different human gameplay can provide valuable insights about the behaviours of the players and the content of the game. Finally, we propose a player modeling approach to estimate the strategy requirement of different game contents.

1.3.1 Benefits, Ethics and Sustainability

Being able to model different types of players can be beneficial for all the companies in the game industry that can use the same approach to better understand their game and their players. Most of the benefits regard game designers that can save time by automatically testing a game content in few minutes and obtaining accurate estimates of how a content is perceived by different players. Nevertheless, some ethical considerations need to be addressed.

An emerging risk we need to take into account is the excessive growth of player profiling techniques especially in tech companies. It is a recent news [27], that Cambridge Analytica, a British political consulting firm, harvested private information from more than 50 million Facebook users without their permission, making it one of the largest data leaks in the social network's history. Profiling technologies raises not only ethical but also privacy, security, liability and equality issues. To better understand the risks and implications of these technologies we refer to [28], [29]. Privacy is one of the primary issues. Tracking and monitoring individual's behaviour may reveal personal information that the monitored individuals might not even be aware of themselves. Since in our research we track data from real players, to guarantee the privacy of the players, at King, we comply with the General Data Protection Regulation (GDPR) [30] that has become enforceable in the European Union from 25 May 2018. For this reason, all the data are anonymized in such a way that is not possible to go back to the real player. As the player models get improved, we need less data about individual players to predict their behaviours. However, the intent of this thesis is improving automatic playtesting to advance player experience and gaming entertainment. Consequently, we respect and promote the code of professional ethics for modeling and simulation [31] defined by T. I. Orel. A more tragic ethical issue has been exposed by Stephen Hawking [32] at BBC in 2014. Hawking said that the increasing effort into developing thinking machines poses a threat on humanity existence. However, he refers to a full AI that is able to think better than humans, and it is able to recursively improve itself. On the contrary, our AI agent is only able to perform a specific task, simulating humans while playing a puzzle game. For this reason we do not elaborate more on the threat of AI, referring to Toby Walsh [33] for a deeper reflection.

If we expand the scope, another ethical issue regards jobs that can become obsolete with the outcome of this work. As automatic playtesting become more popular, jobs that consist of testing game content could be damaged. However, automatic playtesting regards only those jobs that consist of repetitively performing the same task many times and measuring it. These monotonous jobs can be shifted to others more suitable for humans such as developing, improving and maintaining the testing infrastructure or analyzing the test results. Furthermore, in our opinion, human testing is still the best approach when it concerns subjective or qualitative measurements like fun or player experience and it should be used in conjunction with our approach. Finally, for a more exhaustive reflection on ethics of AI we refer to Bostrom and Yudkowsky [34].

The last consideration regards sustainability that is becoming a central aspect in our decisions and our lifestyles. In 2015, governments, companies and civil society defined the 17 sustainable and development goals with targets for the next 15 years. Our work is beneficial for both the goal number 8 *"decent work and economic growth"* because it removes the repetitive component in the human testing work and for the goal number 12 *"responsible consumption and production"* because it reduces the number of iterations needed to adjust and test a game content, diminishing the computational resources used and therefore the consumed energy.

1.4 Methodology and Method

Our work is inspired by previous researches in the automatic playtesting field as described in Section 1.1. In this research we use an inductive [35] and data-driven approach because we base our decisions on analysis of player data and we try to generalize from these examples. We use a quantitative method [36] since we statistically test our results and because we want our outcomes to be comparable with previous researches. We attempt to create a framework that extracts the player strategies from gameplay data and simulates them. Data is collected by tracking some of the real players while playing the *Candy Crush Saga* game for a period of three months. We use player modeling together with a CNN-based approach to model and simulate gameplay of various groups of players. Subsequently, simulating different strategies, we estimate the human SR on new levels and we compare our approach with the previous state-of-the-art [20]. Finally, we use the defined player models to characterize levels.

From a philosophical point of view, we aim to develop agents that are able to "act intelligently", or more concretely, to play a specific game "intelligently". In our case, we define *intelligence* as the capability of the agent to simulate a human strategy and as a consequence to imitate *human thinking*. We do not aim to develop super-human agents but we are interested modeling various types of real players, including players with low performances or players that use non-optimal strategies. Our work can be considered as an example of "weak AI" since the scope of our agents is limited to a specific and narrow task. In contrast, the "strong AI" concept refers to the development of machines with the ability of apply *intelligence* to any problem and they are not limited to one specific task.

1.5 Delimitations

Within the scope of this thesis, player modeling is limited to the study of the players' behaviour in terms of strategies used, defined by different policies to select actions in the game, and skills possessed by each player that allow them to solve the levels. At this moment, we do not consider the affective and psychological perception of the players that often requires specific and intrusive systems to be measured and modeled. In spite of that, a measure of the perceived entertainment could be worth when designing the game.

Despite the generic nature of the question posted in the thesis, we focus solely on the game *Candy Crush Saga*, a match-three puzzle game developed by King in 2012, as a test case to answer our research question. However, the methods and analysis of this work are general and can be easily extended to similar games e.g. *Farm Heroes Saga* or *Bejeweled*, and they might be valuable even for other kinds of games. *Candy*

Crush Saga can be considered as a casual match-three puzzle game. In a match-three puzzle game the player has to manipulate tiles in order to create a sequence of three or more adjacent elements of the same type. Furthermore, *Candy Crush Saga* is a non-deterministic game because game content is randomly generated during the game. It mainly consists of a series of levels to be fulfilled in sequence. On each level, the player has different objectives to reach. Almost all the levels are limited by a maximum number of available moves to reach the objectives. However, there are few levels (4.73% of total levels) that are limited by time. This type of levels are not considered in this research since a different approach to model them would be necessary. Moreover, in the game there are few levels (2.83% of the total levels) that contain a special item called "Candy Frog". Differently from any other element, the "Candy Frog" can be moved to any cell in the game board. Since a large number of additional actions would be necessary only to model this element, we decide to not consider levels that contain the "Candy Frog" in our research. Finally, the levels in *Candy Crush Saga* are non-deterministic polynomial-time hard (np-hard) problems as demonstrated by Toby Walsh in [37], meaning that in the worst case, solving them requires exponential time. Additionally, this type of game, compared to other types e.g. shooting, adventure or managerial games, introduces more complexity in the player modeling stage due to the fact that a strategy cannot be easily described by words or using a high-level abstraction. In fact, in this thesis, the word "strategy" refers to a policy represented by a model, in our case a CNN. In a sense, the policy function approximated by the CNN is *latent* since it cannot be observed directly. Following the taxonomy defined in [38], the intent of player modeling in this research is on prediction and reproduction without necessary answering *why* players exhibit certain behaviours. As a consequence we do not directly focus on description or interpretation of the generated player models.

1.6 Outline

The rest of this thesis is structured as follows: Chapter 2 presents the background and illustrates the related work in the area of this thesis. Chapter 3 describes the method and methodologies used during this research. Chapter 4 describes the results of our attempts to improve automatic playtesting using player modeling. Chapter 5 discusses interpretation, implications, consequences for the field and future work. Finally, conclusions are reported in Chapter 6.

Chapter 2

Background and Related Work

2.1 Automatic Playtesting

In the last decade, automatic playtesting has become a very active field of research, mainly due to the promising results and the increase of available data. Many approaches and various way of helping game designers testing their work and better understanding players' behaviour or game mechanisms have been proposed. Automatic playtesting allows to obtain fast and low-cost feedback regarding the generated game content without requiring humans to play the game.

2.1.1 Related Work

We present here a summary of the most relevant approaches related to automatic playtesting. Zook *et al.* [11] used active learning to reduce the number of playtesting required to balance the low level parameters in a shoot-'em-up game. Even if effective, their method still requires human testers to evaluate different combinations of parameters and it is not completely automatic. Holmgård et al. [15], first used Q-learning while later [16] used linear perceptrons to model archetypical decision making styles representing the behaviour of different classes of players. Subsequently, they used the developed gameplay agents to automatically test the generated game content. However, both of their work rely on handcrafted playing strategies, defined by different reward functions, used by the agents to learn the corresponding behaviour. A different approach was used by Isaksen *et al.* [17] to evaluate the impact of strategy and dexterity in two popular puzzle games: *Tetris* and *Puzzle Bobble*. Their approach consists of applying variants of Q-learning algorithms to simulate various types of human gameplay. They defined *strategy* as the ability to select the best move and they modeled it with various handcrafted heuristics. At the same time, they defined *dexterity* as the capability of correctly executing a predefined move and they estimated it by performing a small pilot study with real players. They developed several agents by adding human errors in different quantities to a previously developed artificial agent. Finally, they used the resultant agents to measure the strategy and dexterity requirements for each level and the effects of the scoring system in the game.

Focusing on the Candy Crush Saga game, Poromaa [21] developed an AI agent trained with Monte Carlo tree search (MCTS) to estimate level difficulty. The algorithm considers both the final result (win or loss) and partial objectives fulfillment during the roll-out phase to compensate the extremely large search space and the limited number of simulations per move. More recently, Purmonen [24] used a CNN-based agent trained with the previous MCTS simulated gameplay data to estimate level difficulty. This method only simulates the MCTS bot and not a human player. However, Purmonen showed how his approach is useful to estimate the average human SR in substantially shorter time compared to the MCTS-based agent, while being equally good in prediction accuracy. Later, Eisen [20] developed a similar CNN-based agent but trained on player gameplay data. Then, he used the agent to simulate human gameplay and estimate difficulty of new levels. The new approach decreased the mean absolute error (MAE) in the SR prediction compared to the previous MCTS-based agent developed by Poromaa [21]. However, Eisen's implementation does not consider any player feature, e.g. player's skill, and as a consequence it is only able to simulate the average strategy between all the players.

To the best of our knowledge, no one in the research area is using player modeling to simulate human gameplay of different cohorts of players while directly learning the policies from real player data. In the next section we describe what player modeling is and how it is applied in automatic playtesting.

2.2 Player Modeling

Player modeling is a wide concept that can involve different research areas and techniques from modeling body alterations during gameplay to predicting human behaviour in strategy games. We define player modeling as the process of understanding, describing and analyzing the dynamic interactions between a game and the players, to distinguish it from player profiling that refers to categorizing players based on static information, e.g. age, gender or personality. Even if not strictly necessary, nowadays player modeling mainly refers to the use of AI and computational resources to create models of the players. Despite the fact that several papers and researches use a different terminology when talking about player modeling, Smith *et al.* [39] and Yannakakis *et al.* [40] defined a useful high level taxonomy of player modeling that we will use during the writing of this thesis. A distinction that is worth mentioning is between direct modeling or model-free approach and indirect modeling or model-based approach, where the earlier refers to the use of Supervised Learning (SL) techniques and human player data while the latter refers to the use of Unsupervised Learning (UL) techniques, mainly RL, to create AI agents that simulate human gameplay. Although these two approaches represent the extreme cases, the large majority of researches can be described as hybrid between the two. For a better understanding we will refer to human players with the term *players* while we will use the term *agents* to refer to computer-based players.

2.2.1 Player Modeling Metrics

Player metrics are an invaluable resource for understanding players and games. Tychsen and Canossa [41] identified, monitored and discussed several player metrics, like frequency of death or choice of weapon, in the game *Hitman: Bloody Money* to discover patterns of play and building modeled representations of player styles. Similarly, Holmgård *et al.* [15] identified five different potential sources of positive or negative reward (moving, hitting a monster, collecting treasures, dying and reaching the exit) in the *MiniDungeons* game and used them to model different types of playing. The five sources were directly defined by the researchers after an analysis of the game mechanics. Finally, using the defined metrics they modeled five distinct personas: the *Exit* persona who mainly cares about reaching the exit of the level, the *Runner* persona who tries to reach the exit as fast as possible, the *Survivalist* persona who tries to avoid damage to the largest extent possible, the *Killer* persona who tries to hit every monster in the level and the *Treasure collector* persona who cares about collecting all the possible treasures before reaching the exit.

2.2.2 Related Work

The success and the continuous research in the player modeling field is due to the promising and wide range of applications that benefit from its results and it is not only limited to the game industry but affects many disciplines like economics, sociology and psychology. In the last years, the primary use of player modeling techniques has been to classify players based on their interactions with the game. The fundamental idea is to collect data from the players, reduce the dimensionality retaining only the relevant features and then define different behavioural profiles that can be used to test or to better understand a game. One of the first attempt was done by Bartle [42] that classified players into four categories relying on his own intuition to define the clusters. Others [43], [44] relied on behavioural theory to partition the players. Drachen *et al.* [18] used self-organizing maps trained on high-level player data to identify models of players for the game *Tomb Rider: Underworld*. First, they used k-means to determine the number of existing clusters and then used unsupervised learning to identify and visualize clusters of player styles. Drachen *et al.* [45] illustrated a first attempt to identify behavioural profiles by clustering large scale and high dimensional data from two major commercial computer games. They showed how the selection of the clustering algorithm can lead to fundamentally different insights and investigations of the player population. While k-means is useful to extract the general distribution of player behaviours, another clustering algorithm called Simplex Volume Maximization is more useful to discover players with extreme behaviour. Holmgård *et al.* [16] showed that a bottom-up approach built from game designers' knowledge performs equally well to a top-down approach built on player data at representing player decision making styles. Furthermore, Holmgård *et al.* [38], demonstrated that generative agents built on expert knowledge are useful for characterizing and classifying human players and can be used as a playtesting tool in the game design process to interpret human decision making. Nevertheless, they evaluated the two researches on a relative small game compared to most commercial games, called *MiniDungeons*. We believe that a bottom-up approach can better capture player behaviours and explore strategies that are not easily detected by designers' intuition.

The large amount of research in the game of *Go* domain also contributed in different ways to the player modeling field. Sutskever and Nair [46] tried to predict moves made by expert *Go* players in order to narrow down the search space for a computer player. Similarly, Stern *et al.* [47] used Bayesian learning algorithm to learn a probability distribution of expert players' moves given a board position. The idea of distinguishing between expert and non expert players can be considered as a broad type of player modeling. A more accurate approach that considers the skills of the players, was used by Maddison *et al.* [48] to develop a Deep Convolutional Neural Network (DCNN) able to play the game of *Go*. Maddison *et al.* provided the network with an input parameter indicating the rank of the player. This created a dynamic bias to the network depending on the players' skill.

Furthermore, other researchers used player modeling for different purposes. Some interesting examples of these applications are: customized game content generation where the narrative and the game content are personalized to match each player's preferences and expectations [49]–[53], game balancing to adapt the difficulty of a game based on the skills of the players [10], [54], [55], monetization to predict which player is more willing to pay for in-game content using massive amounts of player data [5], game outcome prediction in multiplayer games [56]–[59] and player behaviour prediction, e.g. when a player will stop playing a game or how much time he will take to finish it [60]. Recently, a horror-themed game called *Nevermind* [61] was developed to adapt the game content accordingly with the player's physiological state that was tracked by biological feedback. Pirovano *et al.* [62] developed a rehabilitation station that integrates video games to support rehabilitation at home. The station performs real-time and automatic adaptation of the gameplay based on the patient's conditions. Moreover, player modeling has been used to create agents that not only play well, but also play believably or human-like [22], [63], [64]. Hoorn *et al.* [19] used multiobjective evolutionary optimization to produce agents that imitate players and concurrently maximize the performances in the game. Results demonstrated that, in the specific case of a racing game, performing well is a relative easy task, while imitating humans is very hard with the suggested approach. In 2008, a new version of the Turing Test was proposed [65]. Participants needed to submit AI agents that behave as human in a game and a number of judges must distinguish if a player is a human or a bot. Results showed that the submitted bots do not behave as human at that moment. Nevertheless, in 2012 two AI agents [66], [67] passed the test in the *Unreal Tournament 2004* game.

2.3 Gameplay Simulation

Since the early 50s, pioneers of computers wrote programs to play games. In 1950, Shannon [68] published a paper describing how to program a computer to play Chess using move selection and position scoring. His work is considered one of the first attempt to program a computer for playing Chess. Similarly, in 1953, Turing [69] used a Minimax algorithm to play Chess, trying to demonstrate that computers can execute tasks that require "intelligence". In 1952 A. S. Douglas wrote an algorithm to play the Tic-Tac-Toe game, while in 1959, Samuel [70] wrote an algorithm to play Checkers, learning to play against itself and inventing what is now called reinforcement learning. More recently, the focus shifted on more complex games with a huge search space, like backgammon or Chess. The intent is to develop algorithms that are better than any human in that specific game. In 1994, Schaeffer *et al.* [71] wrote *Chinook*, a computer program that plays checkers and that is the first program that won a world champion title competing against humans. In 1995, Tesauro [72] developed *TD-Gammon*, a temporal difference learning algorithm that reached top-human capabilities in the backgammon game. In 1997, *Logistello*, a computer program developed by Michael Buro that plays the Othello game, beat the human world champion Takeshi Murakami six games to none [73]. Famously in the same year, IBM [74] developed *Deep Blue*, the first algorithm that reached super-human performance in Chess, defeating the then-reigning World Chess Champion, Garry Kasparov. Currently, the focus of gameplay simulation is on even more complex board games, like Go, and on video games different from board games, like first-person shooter (FPS) games or real-time strategy games. The benchmark for the near future might be *StarCraft II* (Blizzard Entertainment, 2010), as demonstrated by the recently research on it [75] and the interest of big companies like Google, Facebook [76], [77] and Alibaba [78]. *StarCraft II* is a real-time strategy game with a search space of approximately 10^{1685} [77] (game of Go has about 10^{170} states) and at this moment, the best *StarCraft II* computer program only reaches an amateur level [13].

2.3.1 Candy Crush Saga

Candy Crush Saga is a free-to-play game released by King on April 2012 that in five years has been downloaded 2.73 billions of times [79]. It can be considered as a casual match-three puzzle game where the core activity is to swipe two adjacent candies on the board of the game to make a row or a column of three or more matching-colored candies. A match of less than three candies is allowed when it involves special candies or items. When a match is performed, the matched candies disappear and the candies above them fall into the empty spaces while new candies are randomly generated at the top of the board. Moreover, making matches of more than three candies creates one of the 6 types of special candies that can be used to clear entire columns, rows or section of the board. The board consists of a 9x9 grid that can contains candies, special candies, special items, blockers or empty spaces that are filled by blocks of various colors. There exist 13 different types of blockers, each one with specific characteristics that obstacle the player in performing the swaps and several special items each one with specific effects on the game. The player earns points for each match of candies performed. The game is presented as a series of levels that need to be completed in sequence. The game contains more than $3,000$ levels and new ones are released every week. To be considered finished and allow the player to progress in the sequence, each level has one or more objectives to be fulfilled. The objectives, that also categorize the levels, can be: reach a predefined number of points (*score levels*), remove jellies at certain positions (*jelly levels*), move items, called ingredients, to specific positions on the board (*ingredients levels*), remove a predefined number of certain items (*candy order levels*), reach a predefined number of points within time limit (*timed levels*) or a combination of the mentioned objectives (*mixed levels*). Jellies are items that cover other candies and are removed if the candy they cover is involved in a match or if it is destroyed by a special effect. Ingredients are special items that cannot be part of a match but they can be swapped if the other candy involved in the swap lead to a legal match. Furthermore, each level contains three score thresholds and for each threshold reached, the player obtains a star. Obtaining at least one star is a requirement on all the levels. Except for timed levels, all the other objectives need to be fulfilled within a specific number of moves. Even if with a large variance, on average, a level has a depth of 32 moves and each board state has, on average, a branching factor of 10.28 actions. This lead to a search space of approximately 10^{32}. However, we are not taking into account the randomness in the game. Considering that most of the levels replace the removed candies with random candies selected between 4 or 5 different colours, that cascade effects increase the number of states traversed between actions and that some levels randomly generate not only coloured candies but also items or special candies, the actual search space is extremely bigger. This motivates the difficulty of evaluating the action to perform at each given state. Another core characteristic of the game is accessibility. From the human perspective is not always easy to notice all the

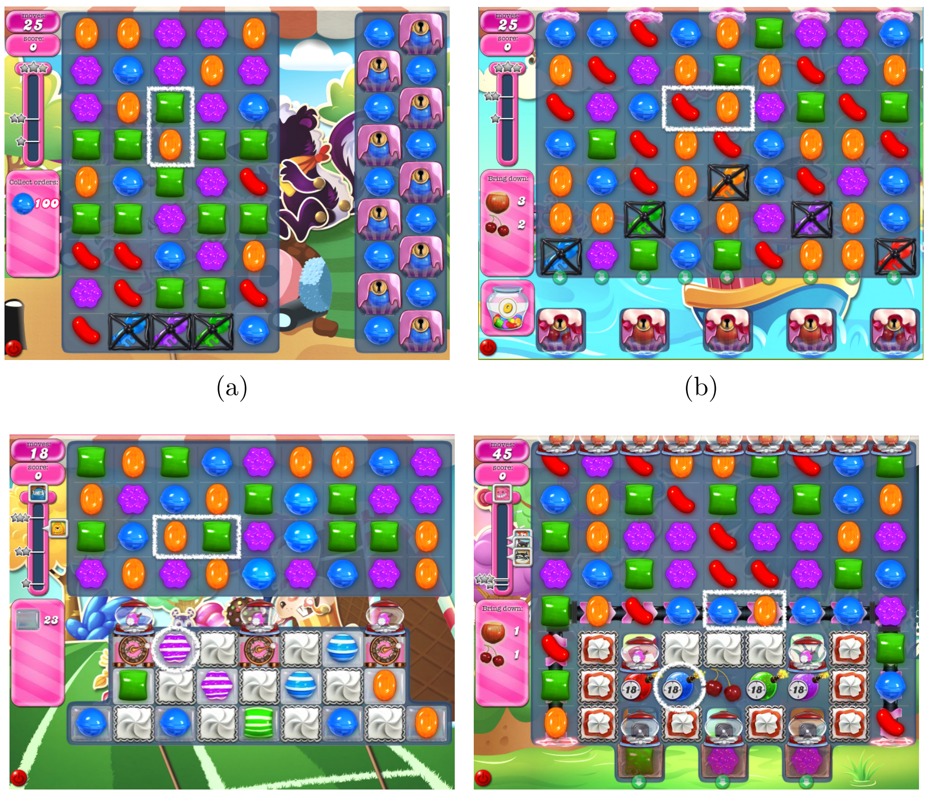


Figure 2.1. Examples of legal moves in the game and examples of items. White boxes illustrate examples of possible swaps while white circles illustrate special candies or items.

possible moves in each state, due to the colorful, animated and quite large game board. It is also worth to mention that several players play the game just for fun or to relax without considering all the possible moves or evaluating them and this can be considered as a source of noise in our data when we try to model different player strategies.

Figure 2.1 shows four different game states and various types of candies and items of four different game levels. In Figure 2.1a the white box shows a possible swap that, if executed, creates a special candy called "Colour Bomb" that when switched with a regular candy, it deletes all the candies of that colour from the board. In Figure 2.1b the white box illustrates a swap that creates a "Striped Candy". When a "Striped Candy" is matched, it deletes all the candies in its row or column. In Figure 2.1c the white box shows a regular swap while the circle illustrates a "Horizontal Striped Candy". In Figure 2.1d the white circle shows a "Candy Bomb" with a timer of 18 available moves before it explodes and the player loses the game. Also, the white box shows a regular swap on a "Conveyor Belt" that is an item that moves the candies on it by one position in the direction of the belt every turn. Moreover, the examples in Figures 2.1 contain a lot of others special items, each one with its own characteristics and effects. The images in Figure 2.1 show different level objectives. The level in Figure 2.1a is a *candy order level*, the level in Figure 2.1c is a *jelly level* while levels in Figure 2.1b and 2.1d are examples of *ingredients levels*. This illustrates the diversity between the levels in the game.

The game allows players to use boosters (sometimes known as power-up). A booster is an item that can be used to facilitate the gameplay and solving a level more easily. In *Candy Crush Saga* there are 15 different boosters each one with different properties. They can be grouped into three main categories:

1. **Pre-level boosters.** Activated before the game starts.
2. **In-level boosters.** Activated during the gameplay if the conditions of the game board create any effect when the booster is used. As an example, a player cannot use a "Bomb Cooler" booster if there are no "Bombs" on the game board.
3. **Consolation boosters.** Activated at the end of the attempt to prevent failure and retrying.

Each booster has a number of charges. Every time a player uses a booster, one charge is consumed. Some boosters can be used only in specific type of levels. In Appendix A.1 a full description of all the boosters with their effects and usage conditions.

2.3.2 Gameplay Metrics

In the last years, game metrics have become popular in the game industry as a source of valuable information about game design. Game metrics are the result of the interaction between the players and the game. Compared to user feedback or surveys, game metrics have the advantage to be objective, not biased by player emotions and they are easier to gather on a large scale. Since our goal is to estimate level difficulty, we use the SR on a specific level as a measure of the perceived difficulty. The success rate $sr_{i,k}$ on a level i perceived by a group of players k is defined as follow:

$$sr_{i,k} = \frac{s_{i,k}}{a_{i,k}}, \tag{2.1}$$

where $s_{i,k}$ is the sum of the number of successes on level i of each player in group k and $a_{i,k}$ is the sum of the number of attempts on level i of each player in the considered group k. The reason why we use the SR as a measure of the perceived difficulty is because we believe that if a player needs to try a level many times before solving it, and thus the SR is low, he perceives the level as a difficult one, while on the contrary, if a player solves a level in few attempts, he perceives the level as an easy one and the SR is high.

Another reason why we use the SR is to allow a comparison with the state-of-the-art approach. However, since the baseline does not consider different groups of players, we furthermore use a combination of the SR of all the k groups to estimate sr_i that is the overall difficulty perceived by all the players on level i.

2.3.3 Related Work

Until Silver *et al.* [80] published their work in 2016, presenting the Google DeepMind's *AlphaGo* system, most of the research in the game of *Go* was focused on RL to teach computer playing the game. The state-of-the-art were MCTS programs that simulates thousands of self-play games to estimate the optimal policy. Silver *et al.* [80] demonstrated how deep neural networks trained on human expert player data can effectively estimate the "value network" used to evaluate board positions and approximate the "policy network" used to select moves in the game of *Go*. The two networks were trained by a combination of reinforcement learning and supervised learning using human expert data. Their research was based on prior work on predicting expert moves in the game of *Go* using supervised learning techniques [46]–[48]. Their solution is the first computer *Go* program that beat a human professional *Go* player without handicaps. In 2015, *AlphaGo* defeated Fan Hui, a three times European *Go* champion, in all of the 5 matches disputed. In 2016, it beat Lee Sedol, a 9-dan professional player considered one of the best players at *Go*, in a five-game match.

Similarly to what Silver *et al.* did with *AlphaGo*, Hlynur [23] illustrated how CNN can be designed to predict expert moves for the *Othello* game, exceeding the previous state-of-the-art by 5.3%. He used a DCNN trained with handcrafted features and the raw board state as input, showing that removing the handcrafted features decreases the accuracy by only 0.9% but at the same time allows for a much faster computation. Chen and Yi [25] demonstrated that CNN can provide similar performance while requiring fewer resources and training time compared to more complex methods such as deep Q-learning in challenging policy estimation task. More recently, the new Google *AlphaGo Zero* program [81] trained solely by self-play reinforcement learning achieved super-human performances, winning 100-0 against the previously published *AlphaGo* system. However, a perfect-play agent that always plays the optimal move and leads to the best possible outcome is out of the scope of this thesis. We aim to model human players and even the best player does not always select the optimal action due to the difficulty of identifing and evaluating each possible move in the *Candy Crush Saga* game. For this reason, it is important that the AI agent plays in a human-like manner, making the same sort of errors and therefore using a similar strategy to human players.

The game of *Go* has several similarities with the game used in our research. First, the grid shape topology of the board and the discrete game action space that make CNN a valid approach to simulate gameplay. Second, the vast search space that makes computationally hard to evaluate each possible state-action pair in the game. Third, the observability, since both games have perfect information. Fourth, the time granularity, since both are turn based games and finally, the Markov property, that practically holds for both the games. *Candy Crush Saga* can be considered, for practical purposes, to have the Markov property, even if there are few special cases where specific game items violate this assumption. There is a special candy called "Chameleon candy" that alternates between two colors every turn. However, since this type of candy is really rare in the game, we will treat the game as a Markov process. Even if there are some differences, like the fact that *Go* is a two player zero-sum adversarial game while *Candy Crush Saga* is a single player game or that *Go* is deterministic while *Candy Crush Saga* it is not, we rely on the results of previous work in the game of *Go* to guide our research. Finally, in the last three years, some related work on the *Candy Crush Saga* game has been done. As mentioned in Section 2.1.1, Poromaa [21] used a MCTS approach while Eisen [20] and Purmonen [24] used a CNN-based approach to simulate gameplay, similarly to what have been done in the *AlphaGo* system.

2.4 Theory

2.4.1 Linear Regression

Linear regression is one of the simplest approach for regression analysis. It models a linear relationship between a dependent variable and one or more independent variables, also called explanatory variables or predictors. Linear regression is largely applied in biological, behavioural and social sciences as well as finance and economics to describe the relationships between different variables. Linear regression models are used both for prediction and for quantifying the relationship between response and explanatory variables. When it is used for prediction, as in this thesis, a linear regression model is obtained by fitting the observed training data. Then, if new values of the explanatory variables are collected, the model is used to predict the response variables.

Given a data set $\{y_i, x_{i1}, \dots, x_{id}\}_{i=1}^{n}$ where n is the number of examples in the data, y is the response variable and $\mathbf{x}$ is the d-vector that describes the explanatory variables, the linear relationship is modeled as:

$$y_i = \beta_0 1 + \beta_1 x_{i1} + \cdots + \beta_d x_{id} + \epsilon_i, \qquad i = 1, \dots, n, \tag{2.2}$$

where ϵ_i is the error term that adds noise to the linear relationship. If written in matrix form the linear equations can be expressed as:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}, \tag{2.3}$$

where

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & x_{11} & \dots & x_{1d} \\ 1 & x_{21} & \dots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \dots & x_{nd} \end{bmatrix}, \quad \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_d \end{pmatrix}, \quad \boldsymbol{\epsilon} = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix}. \tag{2.4}$$

Several procedures to estimate the parameters $\boldsymbol{\beta}$ have been proposed. One of the most popular is Ordinary Least Squares (OLS) since it leads to a closed-form solution. It computes the estimated values $\hat{\boldsymbol{\beta}}$ by minimizing the sum of squared residuals. The estimates are computed as:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} \tag{2.5}$$

If the errors have finite variance and are uncorrelated with the predictors then the estimator is consistent and unbiased.

$$\mathbb{E}[\mathbf{x}_i \epsilon_i] = 0 \tag{2.6}$$

OLS method computes the best fitting line for the observed data. Furthermore, once a regression model has been fit, an examination of the deviations of the observed values from the fitted line (residuals), allows to investigate the validity of the obtained linear model. For more information regarding linear regression analysis we refer to [82].

2.4.2 Convolutional Neural Networks

A Convolutional Neural Network is a specific type of feed-forward artificial neural network that uses a shared-weights architecture and has the translation invariance property. It is mainly used for processing data with a grid-like topology, e.g. time series or images. CNNs were inspired by the animal visual cortex where neurons are activated by stimuli from a restricted region of the visual field. A CNN consists of an input layer, an output layer and a variable number of hidden layers. Layers can be repeated many times and the output of one layer becomes the input of the next one. An example of a CNN architecture is illustrated in Figure 2.2.

A key characteristics of CNNs is that they have sparse connectivity, meaning that when a convolution operation is performed, the output is determined only by a subset of the input (called receptive field) that depends on the size of the filters. However, in a deep CNN, deeper layers may have indirect connections with larger subset of the input. This enable the network to learn complex functions combining simple building blocks. Each hidden layer typically consists of one of the following three types: a convolutional layer, a pooling layer or a fully connected layer.

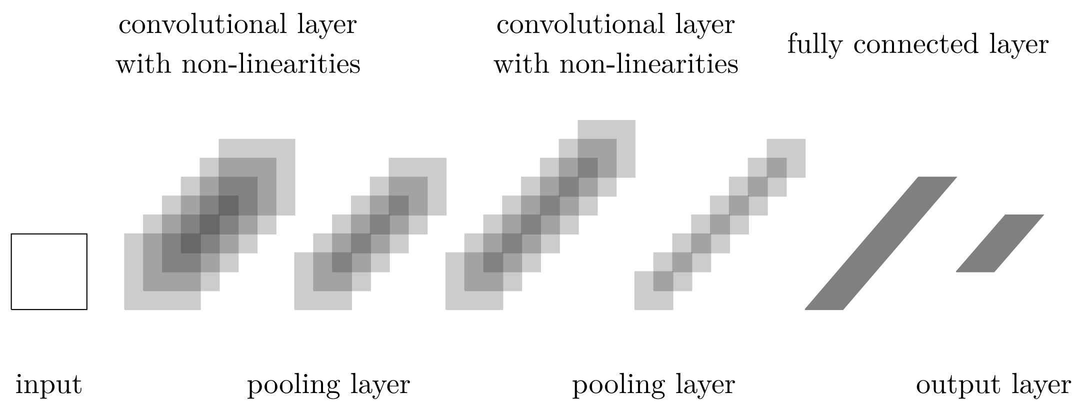


Figure 2.2. Example of a CNN architecture. As first introduced by LeCun et al. in 1989, this network alternates between convolutional layers with hyperbolic tangent activation functions to introduce non-linearities and pooling layers. In this representation, the convolutional layers include the activation functions. The grey squares represent the activation maps. The last two layers are fully connected layers. Usually, the output layer uses a softmax activation function. Image generated with code adapted from [83].

Convolutional layer: It applies a convolutional operation to the input and it drastically reduces the number of free parameters allowing to operate with inputs that have high dimensionality like images. Instead of having a connection from each input to each output, as in a fully connected feed-forward neural network, it uses filters (typically of a small size, e.g. 3x3) that are shared between inputs. In this way, each parameter of the filter is used in every position of the input, with various design choices for managing the boundaries. The filters are moved step by step over the input and at each step, the dot product between the filter and the input is computed. The number of units by which the filters shift over the input is the filter stride. On top of that is typically added a non-linear activation function. Each convolutional layer uses one or more filters and each of them, moving over the input, generates an activation map. Finally, all the generated activation maps are stacked together and they constitute the input of the next layer. As a result, the network learns filters that are activated when they detect specific features in the input. Sometimes batch normalization is added at the beginning of the convolutional layer to speed up training and to reduce overfitting.

Pooling layer: It reduces the size of each activation map combining multiple outputs of a convolutional layer into a single value. In practice, many types of pooling layers exist and as an example the *max pooling layer* combines multiple output of the previous layer retaining only the maximum value. The idea is to generate smaller representations maintaining only the relevant information introducing invariance to small shifts or distortions. Furthermore, the pooling layer progressively decreases the size of the representation, reducing the number of parameters and the amount of computation required by the network.

Fully connected layer: A fully connected layer connects every neuron in the input to every neuron in the output. This type of layer is usually added at the end of the network to feed a *softmax* function and generate a distribution over the output classes. Lin *et al.* [84] proposed another strategy called global average pooling (GAP) to replace the fully connected layers. When using GAP, the last convolutional layer has to generate one feature map for each target class. Then, instead of having a fully connected layer, the average of each feature map is computed and the output is fed into a *softmax* function. The advantage of this approach is that since there are no parameters to optimize in the GAP layer, overfitting is avoided at this layer. Finally, since from each map only the average value is retained the network is more robust to input spatial translations.

The idea of CNN [85], [86] dates back to the 80s. One of the first applications was to recognize hand-written digits on checks or mails but due to slow computational resources and limited amount of labeled data the research on CNNs progressed slowly until 2012. In that year, Krizhevsky *et al.* [87] famously developed a network called "AlexNet" and they won, by a large margin, the ImageNet large scale visual recognition challenge (ILSVRC) [88]. After this successful result, many variants of CNNs have been proposed and new applications have been found. At the time of writing, examples of the most popular and prominent architectures are the inception network and the residual network. The inception network architecture [89] was introduced by Google and mainly consists of parallel 1x1, 3x3 and 5x5 convolutional filters combined into several modules. This architecture is able to reduce the number of features, and so the performed operations, compared to the ones of "AlexNet", leading to a very efficient architecture. The main idea of the residual network [90] is to add skip connections allowing the output of a convolutional layer to bypass the next two layers. This approach helps the learning of deep CNNs with many layers by better gradient backpropagation. For a deeper understanding about artificial neural networks and their applications we refer to [91].

2.4.3 Clustering

Clustering is an exploratory data mining process that consists of grouping together objects such that elements in the same group (called cluster) are more similar compared to elements in different groups. The term "clustering" refers to the general task and many algorithms to perform it have been proposed. In this thesis we use hierarchical clustering and k-means.

Hierarchical clustering

The algorithm iteratively connects elements to form clusters based on distances between elements. Two main hierarchical clustering approaches exist: the top-down approach starts considering all the elements as a single cluster and iteratively splits them into smaller ones, while the bottom-up approach starts considering all the elements as single clusters and iteratively merges them. Different distance functions can be used, e.g. euclidean distance, hamming distance, cosine distance. Also, the user needs to chose the linkage function, since a cluster consists of multiple data point, the algorithm needs a criterion to select which element to use when computing the distance function. Popular linkage functions are the *single-linkage* function, the *complete-linkage* function and the *average-linkage* function. The *single-linkage* function, considers the minimum of the element distances. As a consequence, the distance $D(X, Y)$ between two clusters X and Y is described by:

$$D(X, Y) = \min_{x \in X, y \in Y} d(x, y), \tag{2.7}$$

where $d(x, y)$ denotes the distance between two elements x and y. The *complete-linkage* function considers the maximum of the element distances and it is described by:

$$D(X, Y) = \max_{x \in X, y \in Y} d(x, y), \tag{2.8}$$

instead, the *average-linkage* function, also known as *unweighted pair group method with arithmetic mean* (UPGMA), considers the average of the element distances and it is described by:

$$D(X, Y) = \sum_{x \in X, y \in Y} \frac{d(x, y)}{|X| \cdot |Y|}, \tag{2.9}$$

where $|X|$ and $|Y|$ are the cardinalities of clusters X and Y respectively. Finally, since the general complexity of hierarchical clustering is $\mathcal{O}(n^3)$ this algorithm performs well only for small data sets. However, it does not produce a single partitioning but an extensive hierarchy of merged clusters that can be visualized with a two dimensional diagram known as dendogram [92]. Example of a dendogram is illustrated in Figure 2.3. We can observe how starting with six separate data points the algorithm iteratively merges them until a single cluster containing all the points is obtained.

K-means

The algorithm was introduced in 1976 by Stuart Lloyd [93], which is why it is also referred as Lloyd's algorithm, however it was not published outside the Bell labs until 1982 and the term "k-means" was introduced in the same year by James MacQueen. The algorithm aims to partition n observations into k $(\leq n)$ clusters in which each observation belongs to the cluster with the nearest mean.

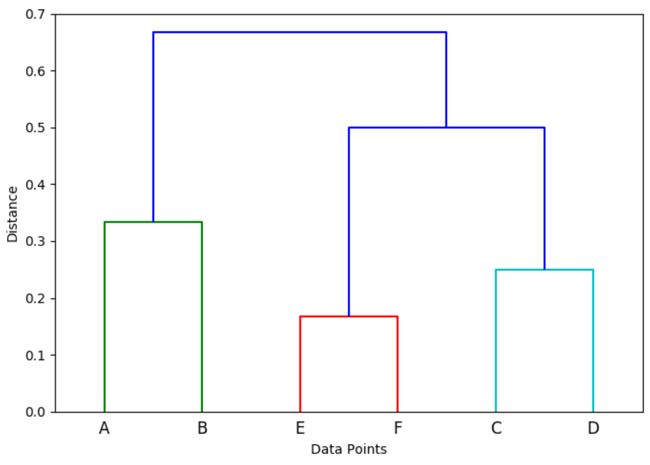


Figure 2.3. Example of a dendogram. On the x-axis the data points to be clustered, on the y-axis the distance measure. The distance between merged clusters is monotone increasing with the level of the merger. The height of each merge is proportional to the dissimilarity between the data points in each merged cluster. Using a bottom-up approach, the first generated cluster is {E,F}, then the second generated cluster is {C,D}, the third is {A,B} and finally, the last two clusters: {E,F,C,D} and the cluster with all the data points {A,B,E,F,C,D}.

More formally, given a set of observations $(x_1, x_2, \ldots, x_n)$, where each observation is a d-dimensional array, k-means partitions the n observations into k sets $S = S_1, S_2, \ldots, S_k$ trying to minimize the within-cluster sum of square (WCSS) distances:

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2, \tag{2.10}$$

where μ_i is the mean of the points in S_i. The most common version of the algorithm uses iterative refinement to assign each observation to a cluster. The number of clusters k is an input parameter of the algorithm and its choice significantly impacts the quality of the output. The algorithm, starting with a set of k-means, alternates between two steps until convergence:

1. Assignment step: assign each observation to the nearest cluster,
2. Update step: compute the new means of each cluster, called centroids.

The algorithm converges when no observation changes cluster during the assignment step, however it does not guarantee to find the optimal solution. A common approach to select the parameter k is the elbow criterion [94]. The main idea is to run the algorithm for a range of values of k and computing the sum of squared errors (SSE) for each value.

Plotting the SSE against the number of clusters k, we can chose k by looking at the "elbow" of the line. The idea is to select a small value of k that still has a low SSE. Sometimes this value cannot be unambiguously identified. Regarding the initialization of the centroids, many variants have been proposed and a common approach is to randomly initialize them. For more details about k-means we refer to [95].

Chapter 3

Methods

The approach we use for improving automatic playtesting can be divided into three major stages:

1. **Player modeling.** In this first stage, we model different cohorts of players based on their features. Furthermore, we create artificial agents that simulate their strategies in the game. Since we use human gameplay data to learn the strategies, using the taxonomy defined in [40], we can classify our method as a direct modeling approach. However, since we do not have any direct strategy feature, we use unsupervised learning techniques in this stage. To model the players we evaluate the following two approaches:

   (a) **Clustering players.** First we cluster the players and then we train a different CNN-based agent on each player cluster.

   (b) **Clustering simulated strategies.** We train a CNN-based agent feeding as input both the player features and the game board. Then, during prediction, by changing the players' input parameters, we can simulate several type of players. Lastly, clustering the simulated strategies we select different agents.

2. **Gameplay simulation.** On each level, various agents, simulating different player strategies, play the entire level several times. Based on the result of the simulation we compute the SR of each agent.

3. **Players' SR prediction.** The final stage is to use the agents' SR on the training levels to fit a prediction model. Using this model we can predict the players' SR on new levels.

To perform automatic playtesting, the first stage is executed only one time or when major changes, that made the player models previously defined no longer reliable, are introduced in the game. On the contrary, stages two and three need to be performed each time a new generated game content needs to be tested.

The novelty of these approaches relies in stage 1. Differently from existing approaches, we model strategies of various cohorts of players by directly learning them from player gameplay data. Some of the existing approaches for automatic playtesting consider only a single strategy while others, even if they simulate various strategies, rely on designers' knowledge or reinforcement learning to model them, without considering player data. Furthermore, to the best of our knowledge, this is the first attempt to use player modeling in a match-three puzzle game. In the next section, we describe the data used in our research. Then, we describe in detail the stages of our two approaches and the evaluation measures used. Finally, we propose an approach to estimate the strategy required by each level to be solved.

3.1 Data

3.1.1 Data Collection

Since the number of daily active players for the *Candy Crush Saga* game is large and the generated data is greater than what can be used as training data for a CNN-based agent in a reasonable amount of time, we decided to track moves and players' statistics only from a subset of all the active players (~1%). The tracking process of the moves lasted for three months collecting a good and diverse sample of player data. By contrast, the players' statistics, e.g. player's SR or number of boosters used, have been continuously collected while the player was playing the game. This allows to have accurate measures of the player behaviour not only on the levels played during the tracking period. We collected data from level 1 to level 2945. Levels in the range [1, 2500] are used as training data for the player modeling stage, while levels in the range [2501, 2945] are used as test data in the players' SR prediction stage. The split ensures that training data contains all the possible elements of the game since no item is introduced for the first time after level 2500. Furthermore, this split reproduces the same process that is performed in practice where existing levels are used as training data while predictions are performed on future ones. We remind that, as mentioned in Section 1.5, we removed levels constrained by time or that contain the "Candy Frog" from the data set. Furthermore, due to an error in the game engine interface that is used by the agents to simulate gameplay in *moves levels*, we removed also this type of levels. Nevertheless, these levels represent only 4.6% of the total number of levels. This leads to 2161 training levels and 414 test levels.

3.1.2 Data Representation

Game board features: Each state is represented by 101 feature layers each one of size 9x9. The feature layers can be categorized into four main types:

1. **Item layers.** There are 80 binary input layers that encode, with a binary representation, the items on the game board. Each of these layers is associated to a single item. Each position in the grid is set to 1 if the item is present and on the contrary, it is set to 0 if the item is not present. Since on each game board, only a subset of all the possible items is present, the layers associated with items that are not on the game board are filled with 0s.

2. **Objective layers.** There are 19 binary feature layers of this type. Each layer represents a different objective that need to be achieved to win the level. It is filled with 1s if the objective is not yet fulfilled in the given state and contrarily it is filled with 0s if the objective is already fulfilled or if the objective is not requested for the given level.

3. **Moves left layer.** A layer that represents the number of moves left to finish the level. In the game the maximum number of available moves is 75 and as a consequence the moves left can assume values in the range $[1, 75]$. To give more importance to the last available moves we encode this feature as the ratio of 1 on the number of moves left. Adding this non-linearity has the effect of creating bigger changes in the feature when the moves available are only a few. A change of 1 move left when the available moves are few has a bigger impact on the feature rather than the same change of 1 move when the available moves are a lot. This is because if a player has 75 or 74 available moves he probably do not give too much importance to this difference while if he has only 1 or 2 available moves, the difference becomes important.

4. **Bias layer.** Additionally, we stack a layer filled with all 1s. This layer allows the network to learn a bias on the position of the performed move in the grid. It is possible that moves in a specific area of the grid are preferred to moves in other positions. For example moves in the lower part of the board could be preferred to moves in the higher part because they increase the probability of creating cascade effects and obtaining more points. This layer can be described as an overall "heatmap" of the performed moves. It add a bias representing the likelihood of choosing a move in a specific position on the grid independently of the content of the game board.

Note that except for the item layers, the other layers encode only a single value. As a consequence the layers are created by repeating the value in every cell of the 9x9 plane.

This allows to exploit the properties of the convolutional neural network also for this input information.

Moves: Each move is a swap between two candies or items. By enumerating the inner edges of the game grid we encode the moves with a scalar number. As showed in Figure 3.1, the moves are encoded enumerating first all the horizontal edges and then all the vertical ones. Since the cases where the direction of the move has an impact on the game are rare, we decided to do not distinguish between left-right or right-left moves and between top-down or bottom-up moves. This leads to a total number of 144 possible moves represented with integer values in the range [0, 143].

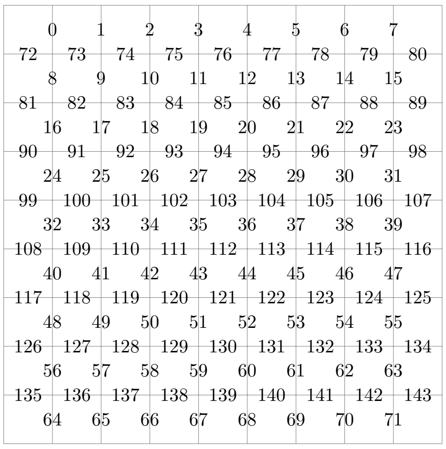


Figure 3.1. Moves encoding. Adapted from [20].

Player features: In Table 3.1 we list all the per-level player features that we use to explore the differences in the player strategies. We define as an attempt each trial to solve a level. An attempt is considered ended if the player completes all the objectives, ends all the available moves or decides to exit the level. If a player decides to use a booster that adds extra moves, the attempt ends when also all the extra moves are terminated. An attempt is considered ended with a success if the player is able to achieve all the objectives within the constraints of that specific level. Since in this work we do not focus on the subset of levels that have time requirements, the constraint is always a maximum number of moves the player can use to complete the level. As boosters, we consider all the possible types: pre-level, in-level and consolation boosters. Each extra move is considered as a single booster. For example, a plus five extra moves is counted as five boosters.

Table 3.1. Player features per level

Features
Number of attempts ended
Number of attempts ended with success
Number of boosters used

3.2 Player Modeling

The first stage aims to model players with different strategies. The goal of this stage is to generate multiple agents that simulate different strategies instead of having a single model that only simulates an average strategy learnt from all the players. Notice that the strategy learnt by the baseline approach is a combination of examples collected randomly from all the players. As a consequence it only represents an average strategy but it does not represent the strategy of an average player. Since for each player we have the performed moves in different states, a direct comparison between gameplay data cannot tell us how differently the players play. In the following, we describe the two different approaches we introduce for modeling various strategies while learning them directly from player data.

3.2.1 Clustering Players Approach

The idea is to cluster the players into different groups based on their performance in the game. A first objective with this approach is to discover if players with different performances in the game exhibit a different decision making tendency or strategy and if a CNN-based agent is able to capture it, approximating a policy function that shapes their actions in the game. This approach consist of three major stages:

1. Computing player metrics
2. Clustering players
3. Training of CNN-based agents

1. Computing player metrics: Using the collected player features, for each player we compute the success rate sr_i, on each level i, as follows:

$$sr_i = \frac{s_i}{a_i}, \tag{3.1}$$

where s_i is the total number of attempts ended with a success and a_i is the total number of attempts started by the player on level i. Using the per-level SR, we can represent each player with a one dimensional array containing all his SR ordered from level 1 to the last level played by each player. Due to privacy reasons we cannot expose the actual players' SR and as a consequence we use the standardized success rate ρ_i computed as:

$$\rho_i = \frac{sr_i - \mu_i}{\sigma_i}, \tag{3.2}$$

where μ_i is the average and σ_i is the standard deviation of the SR between all the players on level i. An alternative to describe the players, could be to use the maximum number of stars or the maximum score on each level. However, the number of stars can assume only three values while the maximum score contains several outliers. As a consequence, we decided to use the players' SR to more accurately describe the player performances on each level. Furthermore, in order to have a fair and complete comparison between players, in this approach we restrict our scope only to those players who finished all the training levels. In other words, we eliminate players that did not reach level 2500. This lead to a subset of $32,479$ players. This is also motivated by the fact that we are interested in predicting the perceived difficulty only on new levels and in order to play these levels, a player must have finished all the previous ones. We remind that we tracked only approximately 1% of the total number of active players.

2. Clustering players: Using k-means with euclidean distance we cluster the players' SR distributions to create k clusters of players with similar skill distributions over the training levels. We use k-means since it can work with data with high dimensionality and euclidean distance. Furthermore, k-means computes centroids that can give us a qualitative interpretation of what kind of player group each cluster represents. In order to decide the number of player clusters, represented by the parameter k, we use the elbow criterion.

3. Training of CNN-based agents: Coherently with the defined player clusters, we divide the state-action pair represented by the game boards and the associated performed moves. As a consequence, each cluster data set contains moves tracked from players with similar skill distributions. Then, on each cluster, we train a CNN that given the game board as input, predicts the performed move of the player. All the agents are trained for 10 epochs with the same network architecture and the same hyperparameters. For each agent we use 30 million state-action pairs randomly selected from each cluster data set. The test sets consist of 100,000 state-action pairs for each cluster. The training takes about 18 hours on a single machine with 6 CPUs and one Nvidia Tesla K80 GPU. Figure 3.2 summarize the various steps of this approach.

Finally, as a baseline for this approach, we train an identical CNN-based agent with the same amount of data and for the same amount of steps as each cluster agent. The gameplay data are randomly selected from data tracked from player that reached level 2500 without any distinction based on player features.

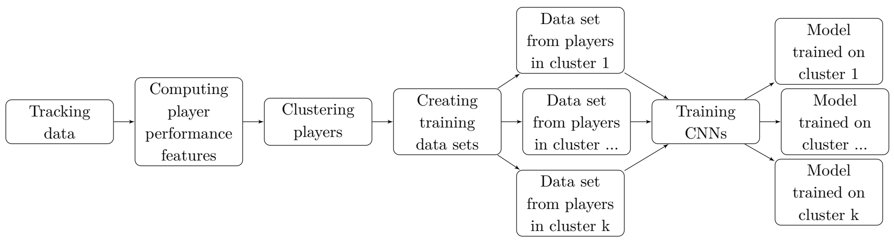

Tracking data
Computing player performance features
Clustering players
Creating training data sets
Data set from players in cluster 1
Data set from players in cluster ...
Data set from players in cluster k
Training CNNs
Model trained on cluster 1
Model trained on cluster ...
Model trained on cluster k

Figure 3.2. Clustering players flow: We track data from a subset of all active players. We then compute player performance features and we cluster the players into k clusters. Then, we divide the data set based on the defined clusters. The generated data sets are then used to train k different predictive models.

3.2.2 Clustering Simulated Strategies Approach

The idea is to let the CNN automatically discover the relationship between the player features and the performed moves since the neural network is able to automatically discover even complex relationships between input and output. Similarly to what Maddison *et al.* [48] did in their research, we provide the network with additional information regarding the player who performed the move. Compared to the previous approach, where we target one type of player at a time, it has the advantage of avoiding a substantial reduction of training data. This approach consists of four major stages:

1. Computing player metrics
2. Training the CNN-based agent
3. Predicting moves simulating different players
4. Clustering the simulated strategies

1. Computing player metrics: Since we leave to the CNN the task of learning the relationship between player features and predicted moves, this time we use three different dimensions to represent each player.
First of all, using the collected player features, we compute the standardized success rate ρ_i on each level i as defined in equation 3.2. Similarly, we computed the average amount of boosters b_i, used on each level i, as follows:

$$b_i = \frac{\text{boosters}_i}{a_i}, \tag{3.3}$$

where boosters$_i$ is the total number of boosters used by the player on level i and a_i is the total number of attempts started by the player on level i. Subsequently, we compute the standardized value β_i as:

$$\beta_i = \frac{b_i - \mu_i}{\sigma_i}, \tag{3.4}$$

where μ_i is the average and σ_i is the standard deviation of the amount of boosters b_i between all the players on level i. Finally, for each player, we compute the following three summary statistics:

- The standardized mean success rate $\bar{\rho}$ as:

  $$\bar{\rho} = \frac{1}{l} \sum_{i \in L} \rho_i, \tag{3.5}$$

  where L is the subset of levels played by the player during the tracking period and l is the number of levels in L.

- The standardized mean amount of booster $\bar{\beta}$, computed as:

  $$\bar{\beta} = \frac{1}{l} \sum_{i \in L} \beta_i, \tag{3.6}$$

  where L is the subset of levels played by the player during the tracking period and l is the number of levels in L.

- The number of levels played l during the tracking period.

These three player features can be joined together through a unique identifier that represents a specific player. For privacy concern, this unique identifier is anonymised in a way that is not possible to go back to the real player. We decided to use these three metrics because we believe that they can exhaustively represent the various types of players in the investigated game, e.g. skilled vs less skilled players or regular vs occasional players. Since we only take average values between all the levels played by a player, we do not anymore need to restrict our focus on players who completed all the $2,500$ training levels.

Furthermore, standardizing the SR, is not only necessary for privacy reasons but it also helps to take into account that different players may have played different levels. Computing the standard values includes the difficulty of each level in the player metric. However, in order to have meaningful summary statistics, we removed players who played less than 50 levels. This is motivated by the fact that the mean, if computed on few values, becomes too sensitive to outliers. Despite this, we still have tracked data from more than half a million players.

2. Training the CNN-based agent: We train a CNN-based agent that given the game board and the three player features as input, predicts the move performed by the corresponding player. Since players are not divided into groups, we have much more data compared to the previous approach. Therefore, the network is trained for 10 epochs using 125 million state-action pairs as training data. The training data are selected to have approximately 50,000 examples for each level in the range [1, 2500]. As a consequence, each level is represented with a similar amount of data. The training takes about 3 days on a single machine with 6 CPUs and one Nvidia Tesla K80 GPU. The test data consist of 100,000 examples.

3. Predicting moves simulating different players: The trained CNN acts as a move predictor sensitive to player features. It is able to generate different player moves for the same input game board only by changing the input player parameters. The player features are continuous values and, in theory, we can simulate an infinite number of players. Since our goal is to generate a reasonable amount of different agents, we select for each player features the 5th, 25th, 50th, 75th and 95th percentile and we create an agent for each possible combination of these parameters. These percentiles are selected to have an extensive representation of the real players. We decided to not use the extreme values because they might represent anomalous players.

Finally, each agent is created by cloning the trained CNN and fixing the input player features with a combination among those selected. This generates 125 different agents, each one simulating a different type of player.

4. Clustering the simulated strategies: The 125 generated agents can be an exhaustive representation of the different type of players in the *Candy Crush Saga* game. However, simulating gameplay of 125 agents for each level that needs to be tested is computationally expensive. Moreover, some of these agents simulate very similar strategies. In this step, we want to reduce the number of generated agents maintaining only a small number of agents while maximizing the differences in the simulated strategies. To select the agents, we create a validation data set with 10,000 state-action pairs and for each agent we predict the moves. We can then represent each agent with a one-dimensional array containing the predicted moves. Each array can be considered as an explicit representation of the strategy of each agent. Then, we use hierarchical clustering with hamming distance and complete-linkage function on the generated arrays, to group agents with similar strategies. Finally, we select a single agent for each cluster. Figure 3.3 summarizes the various steps of this approach.

As a baseline for this approach, we train a CNN-based agent for the same number of steps and using the exact same data but without the input player features. As a result, the baseline agent represents an average policy between all the players and simulates a single strategy. It is worth to remember that the objective of this thesis is not to directly compare the two experimented approaches, but to understand if each approach is better than its own baseline answering if player modeling can improve automatic playtesting.

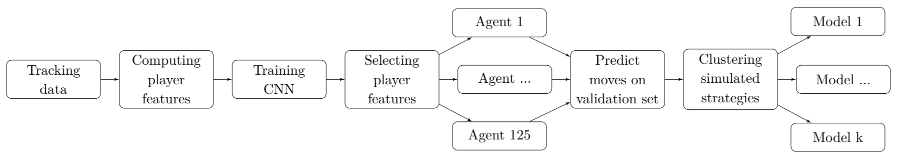

Tracking data
Computing player features
Training CNN
Selecting player features
Agent 1
Agent ...
Agent 125
Predict moves on validation set
Clustering simulated strategies
Model 1
Model ...
Model k

Figure 3.3. Clustering simulated strategies flow: We track data from a subset of all the active players. We then compute player features and we train a CNN-based agent giving as input the game board and the player features. Then, we predict moves on a validation set with different agents. Each agent shares the same network but it has a different player input feature combination. Finally, we cluster the simulated strategies and we select the agents maximizing the differences in their predictions.

3.3 Gameplay Simulation

Eisen [20] illustrated how a CNN-based agent can be used to predict human moves in the *Candy Crush Saga* game. Since his approach reached the best performance to date, we decided to use a similar network architecture for predicting player moves.

3.3.1 Player Moves Prediction

This stage is identical for both the experimented approaches since it does not depend on how we created the various agents. To simulate gameplay, for each agent, we use the following infrastructure. Several machines run the game engine, integrated with the agent interface, that takes care of simulating the attempts. At the same time, on a web server we run various replicas of the CNNs. We decided to separate these two components to allow the agent to use different approaches to select the moves, abstracting it from the predictions of the neural network. The agent interfaces communicate with the CNNs trough a component called load balancer. The main task of the load balancer is to redirect the requests received from the agent interfaces to the CNN with the lowest utilization. The requests contain the encoded game state obtained from the agent. The response is the predicted probability distribution over the 144 possible moves. When an attempt is finished, the results of the simulation are saved on a table using MySQL database.

For each agent we ran a simulation on both training and test levels. A simulation consists of 100 attempts on each level. This number is selected to run the simulation in a reasonable amount of time. Each attempt runs with a different seed that causes the game to generate different random content. However, to have a fair comparison between agents, the selected seeds are the same for each simulation. The simulation takes approximately 18 hours for each agent while running on a total of 26 virtual central processing unit (vCPU) cores and 24 GB of memory. The vCPU cores are divided between the various components as follow: 18 vCPU cores are used to run the bot interface, 2 vCPU cores are used by the load balancer while the remaining 6 vCPU cores are used to run the CNNs. This allocation of resources has demonstrated to work well in practice, in term of speed and utilization. All the computational resources are allocated using a cloud service provider. When the network is used as an agent to simulate gameplay, a greedy policy is used to select the moves. Since the output of the network is a probability distribution over the 144 available moves, the greedy policy selects the move with the highest predicted probability. Finally, from the results of the simulations we compute the agents' SR on each level. These metrics indicate how difficult a level is for each agent.

3.3.2 Convolutional Neural Network Architecture

When designing a CNN, several architectural choices need to be taken and various hyper-parameters need to be selected. In this research we rely on the choice made and discussed in [20] with only small changes to the input layer to consider the player features in the clustering simulated strategies approach. For completeness, in this section, we describe the network architecture used for predicting the moves given the states. The network, illustrated in Figure 3.4, is trained with Adam optimizer [96] and an initial learning rate of 0.0005.

Input Layer: The input of the network slightly differs between the two approaches we illustrate in our research. In the clustering players approach, the input is represented by 101 layers each one of size 9x9 to represent a game board as described in Section 3.1.2. To train the network we use mini-batch gradient descent with a batch size of 2048 examples since it showed to be a good compromise between accuracy and efficiency. This lead to an input of size [2048x9x9x101]. Differently, in the clustering simulated strategy approach, we add as input three player features leading to an input size of [2048x9x9x104].

Convolutional Layers: The network consists of 11 convolutional layers with 35 filters per layer and kernel size of 3x3. Since we do not want the translation invariance property of CNN architectures because we want to accurately determine which move to select, we do not use pooling layers. Furthermore, since the game board is only 9x9, we want the output size to be the same as the input size. As a consequence, we used zero-padding and

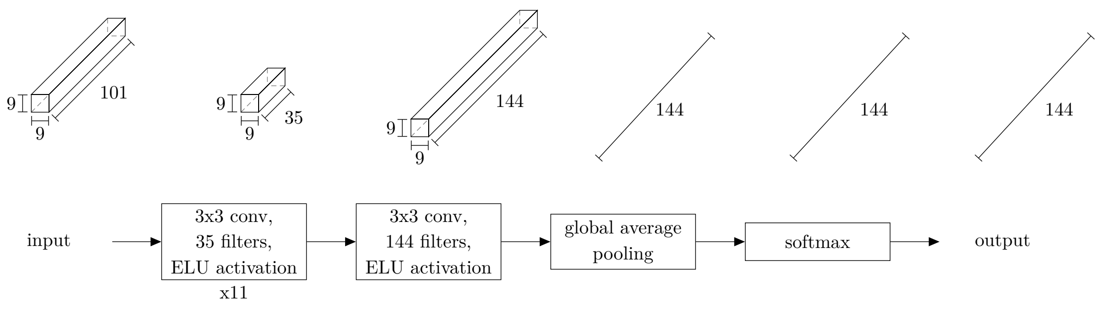

9
9
101
9
9
35
9
9
144
144
144
144
input
3x3 conv, 35 filters, ELU activation
x11
3x3 conv, 144 filters, ELU activation
global average pooling
softmax
output

Figure 3.4. Convolutional neural network architecture used in the clustering players approach, with data representations for each layer. In the clustering simulated strategies approach, the input layer changes from 101 to 104 input features. Image generated with code adapted from [20].

filters with stride of 1. Subsequently, a last convolutional layer with 144 filters is added to the network. The range $[0, 143]$ represents the encoding of the 144 possible moves that the network can predict. Each convolutional layer is followed by an exponential linear unit (ELU) activation function. The ELU function, introduced by Clevert *et al.* [97], is computed as follow:

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(e^x - 1), & \text{if } x \leq 0 \end{cases} \tag{3.7}$$

where the ELU hyperparameter α controls the saturation of the ELU function for negative values. As suggested in [97], we use a value of $\alpha = 1.0$. Compared to rectified linear unit (ReLU) function, the ELU function has also negative values, which allow the network to push the mean of each activation map closer to zero, reducing the *bias shift* typical of ReLU functions. Furthermore, as showed in [20], the ELU improved the accuracy of the model by 2.5% compared to the ReLU function.

Output Layer: The output layer is a GAP layer that transforms the 144 activation maps of the last convolutional layers into 144 scalars retaining only the average value for each map. Using a GAP layer also motivates the choice of encoding the information represented by scalar values into feature planes by replicating the number on all the cells of the 9x9 grid. Otherwise a fully connected layer would be necessary to include these features into the model. Finally, in order to have a probability distribution over the predicted moves, the *softmax* function is applied.

3.4 Players' SR Prediction

At this step we have the per-level agents' SR on both training and test levels. As demonstrated in [20], the agents' SR does not directly map to the SR of the players. However, they are correlated. In this stage we combine the agents' SR on test levels to better predict the players' SR compared to the baseline in each of the two approaches. In order to not expose the actual values of the players' SR for privacy reasons, we scaled the players' SR dividing each value by the difference between the maximum and the minimum value. As a consequence, the scaled players' success rate sr'_i on level i is defined as:

$$sr'_i = \frac{sr_i}{max(sr_i) - min(sr_i)}, \tag{3.8}$$

where sr_i is the actual players' success rate on level i, $max(sr_i)$ is the maximum success rate and $min(sr_i)$ is the minimum success rate between all the levels. This has the effect of scaling the values while maintaining the same distribution. Furthermore, since the distributions of SR for both the players and the agents have a positive skew, meaning that the mass of the distribution is concentrated on the left, we apply a log transformation on each variable. Nevertheless, applying the log transformation causes the values of the SR that were zero to become minus infinite. As a consequence, we decided to use the mean value to predict the players' SR when the agent's SR is zero. An alternative could be to use the $log(x+1)$ transformation where in our case x is the agent's SR. However, we prefer to handle the levels where the agent failed in a different way because when the agent's SR is zero does not necessary mean that the level is extremely difficult. This is also motivated by the fact that for extremely low values of the agents' SR there is no linear relationship with the players' SR. Since levels are designed and tested to be solvable by humans within a reasonable number of attempts, extremely low values of the players' SR do not exist. This explains why there is no linear relationship between extremely low values of the log of the players' and the log of the agents' SR. We excluded those levels where all the agents and the baseline failed since they would just introduce noise to the comparison of the approaches. Finally, we decided to not use a cross-validation approach for two reasons: first, because we want to produce the same scenario that would be used in practice where the existing levels are used to make predictions on new levels and second because we trained the agents on the training levels only and we let them play on new ones to reduce overfitting. In this stage we use two slightly different methods for the two experimented approaches.

3.4.1 Clustering Players Approach

Using the players' and agents' SR on training levels we fit a linear regression model for each agent, including the baseline. Then, we use the linear models to predict the players' SR on the test levels. An issue we need to take into account is that in *Candy Crush Saga*, levels might have been modified during time. Since the players' SR is computed from the players' attempts at the time they have played while the agents simulate the current version of the levels, we eliminate from our data set those levels that have drastically changed. We classify a modified level as drastically changed if the players' SR on it changes more than 50%. These levels represent 18% of the total number of levels and removing them leads to 1722 training levels and 388 test levels.

Each model predicts the players' SR of the specific group k that it represents. In order to compare our approach with the state-of-the-art, we combine the predictions of the agents on each test level i, computing the overall predicted success rate $\widehat{sr}_i$ between all the players using a weighted average as follow:

$$\widehat{sr}_i = \sum_{j=1}^{k} w_j \cdot \widehat{sr}_{i,j}, \tag{3.9}$$

where k is the number of defined agents, w_j is the percentage of players in cluster j and $\widehat{sr}_{i,j}$ is the success rate on level i of the agent trained with data from cluster j. The idea is that the predictions of each agents have a weight that depends on the percentage of players that the agent simulates. Finally, the predictions of this model are compared to the predictions of the linear regression model created from the baseline agent's SR.

3.4.2 Clustering Simulated Strategies Approach

Since we cannot precisely determine how many players each agent represents, in this approach we predict on each test level i, the overall predicted success rate $\widehat{sr}_i$ between all the players using a single linear regression model. The model takes as input all the k agents' success rates $\widehat{sr}_{i,j}$. The predictions of this model are then compared to the predictions of a linear regression model trained with the baseline agent's SR.

3.5 Evaluation Metrics

In this section we describe the metrics we use to evaluate both the move prediction performance of the CNN models as well as the SR estimation performance of the linear regression models.

3.5.1 Player Moves Predictions

To evaluate the performances of the CNN-based agents in predicting player moves we use the top-1 and top-3 prediction accuracy. The top-1 and top-3 prediction accuracy are the ratios of correct predictions to the total number of predicted actions. However, the top-1 accuracy considers a prediction as correct if the correct move coincides with the predicted one while the top-3 accuracy considers a prediction as correct if the right move appears in the top three predicted ones.

3.5.2 Linear Regression Models

To compare the baseline agent with our approach, we compute three different measures. The goal is to see how close the predictions of each approach are to the actual observed values compared to the baseline. The measures used in this research are: the mean absolute error (MAE), the mean squared error (MSE) and the adjusted R-squared (R^2_{adj}). The MAE describes the means of the absolute differences between the predicted and the observed players' SR on all the n test levels and it is defined as:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\widehat{sr}_i - sr_i| \tag{3.10}$$

The MSE measures the mean of the squared differences between the predicted and the observed players' SR on all the n test levels and it is defined as:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\widehat{sr}_i - sr_i)^2 \tag{3.11}$$

The MSE has the property of strongly penalizing large deviations from the observed values. Finally, the adjusted R-squared, explains the predictive power of the regression models considering the number of predictors. It decreases when a predictor improves the model less than what would be improved by chance. This is a desirable property since in the clustering simulated strategies approach, we compare a method that uses multiple predictors with the baseline approach that only uses a single predictor. The adjusted R-squared is defined as:

$$\mathrm{R}^2_{adj} = 1 - [\frac{(1-R^2)(n-1)}{n-k-1}], \tag{3.12}$$

where n is the number of data points, k is the number of predictors excluding the constant and R^2 equals the square of the Pearson correlation coefficient [98] between the observed and the predicted values.

3.6 Estimating Strategy Requirements

Finally, we aim to categorize levels based on the strategy required to solve them. Similarly to what Isaksen *et al.* [17] have done with a simplified version of *Tetris* and *Puzzle Bobble*. First, they trained an agent that reached super-human performances in the two simplified games. Then they generated a population of different agents that play with slightly different heuristics and finally they made the agents more human-like by adding, in different quantities, two types of human errors: strategy and dexterity errors. A strategy error means that a human player not always selects the optimal move, sometimes he makes a mistake and he selects a sub-optimal or random move. A dexterity error means that even if the player knows which move he want to perform, sometimes he is not able to correctly perform it due to pressure, time-constraints or other limitations. This approach for player modeling is based on the idea that models of players are obtained from a super-human agent by adding human errors to it. Using *Candy Crush Saga* as a test bed we aim to estimate the impact of human errors on some levels. However, the motor-skill required to correctly perform a move in *Candy Crush Saga* is very low and levels that require a lot of dexterity do not exist. As a consequence, we only focus on estimating the strategy requirement. Nevertheless, if the proposed approaches are applied to other games it may be valuable to estimate the dexterity requirement too. Since the variants of the games tested by Isaksen *et al.* are small enough to evaluate the full game tree, they used heuristics and search to train the policy of the agents and they modeled human errors using variants of Q-learning. On the contrary, we propose an approach that works with CNN-based agents and with more complex games. We model the strategy errors using a parameter ϵ. We modify the policy of the agents from a greedy one to an ϵ-greedy policy. Using an ϵ-greedy policy means that the agent selects greedy moves with probability $1 - \epsilon$, while with probability ϵ, the agent selects a random move with uniform distribution over the set of available moves. Selecting a greedy move means that we chose the move with the highest predicted probability from the CNN-based agent output. Selecting moves randomly instead of greedily represents the strategy error simulated by the agent. This approach is built on the assumption that the strategy learnt from player data is enough to solve the levels. To obtain a meaningful analysis we restrict our focus to those levels that are successfully solved by the agent when playing with no strategy errors, using the greedy policy. Then, by comparing the performances of the agents, we estimate the strategy required by each level. Starting from the previously developed agent that simulates an average strategy, we create a population of different agents by adding different amount of strategy errors. Finally, we randomly selected some levels to test this approach.

Furthermore, in their research, Isaksen *et al.* used an agent that is able to play with an optimal strategy when the strategy error is zero. Since the complexity of the game we focus on is much higher and the game is non-deterministic, that type of agent is not available. As a consequence, we consider the greedy policy as the optimal policy and the SR reached by the greedy agent as the maximum SR. We normalize the values dividing all the obtained SR by that maximum SR. By plotting the agent's SR against the amount of strategy used, represented by $1 - \epsilon$, we observe how the SR improves as we reduce the number of random moves made by the agent. Figure 3.5 shows the two extreme cases. On the left a level that does not require any strategy since independently from how many random moves are performed the level is solved. On the right a level that requires the optimal strategy because if any strategy error is made the SR is zero. These two extreme cases are not present in the game but are useful to give an idea of how to interpret the generated graphs.

Finally, to estimate the strategy required by a level, we compute the area above the line. This metric indicates how much strategy is required to complete the level. In the two extreme cases, the area above the line is zero when no strategy is required and it is one when the optimal strategy is needed.

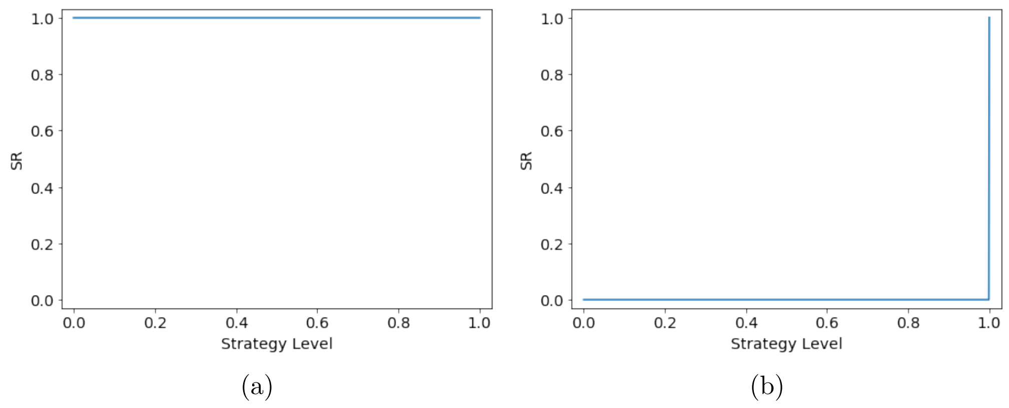


Figure 3.5. Agent's SR against strategy level for the two extreme cases of strategy requirements. Figure (a) shows a level that does not require any strategy while Figure (b) shows a level that requires 100% of strategy in order to be solved.

Chapter 4

Results

4.1 Preliminary Data Analysis

In this section we report the preliminary data analysis that we have done in order to better understand the problem and select the most appropriate methods. However, due to privacy reasons we cannot expose private information regarding the players or the game and as a consequence some of the values have been standardized while others have been completely hidden. The purpose of this section is not to illustrate the underling data but is to give an idea of how we approached the problem and also make it clearer some of the choices that we have done.

For each game match that we tracked, we collected 11 different features. The features collected regard the match itself and do not concern with personal information of the players, e.g. gender, age or country. Furthermore, also the identifiers used to link the matches have been anonymised in a way that it is not possible to go back to the real players. Subsequently, we expanded these features computing the following statistics:

– minimum

– maximum

– average

– standard deviation

– first quartile

– median

– third quartile

As a result we can describe each player with 77 features. Then, the successive step was to pre-process and clean the data to retain only relevant information. We removed duplicates or data that contained errors. Furthermore, we removed those features that had variance equal to zero meaning that the value was the same for all the players. For example, when we focused on the players that reached level 2500, the maximum number of stars for every player was 3. This means that the tracked players in at least one of the training levels have reached a 3 stars score. Since this features does not give any contribute to the player analysis we removed it. Then, we handled the missing values. We decided to use different techniques based on the type of missing value. In this research we used deletion, imputation with mean value and linear interpolation. Subsequently we standardized all the data computing the z-score as follows:

$$\text{z-score} = \frac{x - \mu}{\sigma}, \tag{4.1}$$

where x is the feature, μ is the mean and σ is the standard deviation of the considered feature x. After, we removed the outliers. We considered as outliers all the data that had a standard deviation that was higher than +10 or lower than -10. Finally, since most of the remained features were highly correlated we computed the correlation matrix and removed those features that had a correlation with any other feature that was higher than 0.9. As a result each player was described with 25 different features.

After pre-processing the data, the first analysis we did was the eigenvalue and eigenvector decomposition of the matrix containing the players and their features. Figure 4.1 illustrates the eigenvalue decomposition of the player features matrix. The eigenvalues are plotted in decreasing order. The figure shows that three eigenvalues have a value that is significantly higher than the others. To better visualize the implications of this

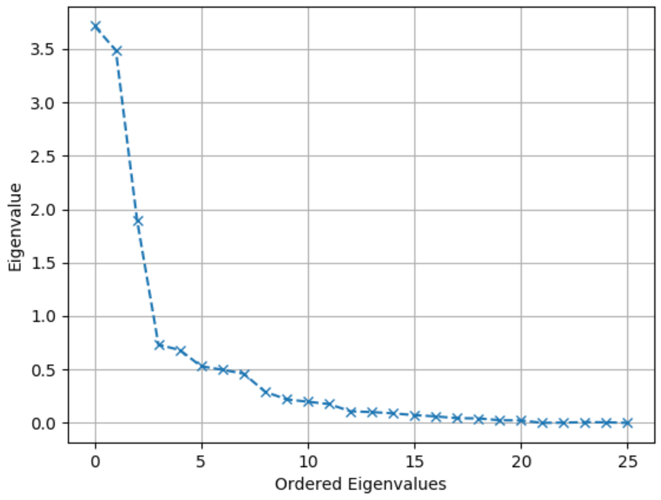


Figure 4.1. Eigenvalue decomposition of the player matrix. The x-axis represents the 25 different features ordered by their eigenvalues in decreasing order and the y-axis the corresponding eigenvalues.

analysis we illustrate in Figure 4.2 the individual explained variance of each feature with the histogram and the cumulative explained variance with the blue line. We observed

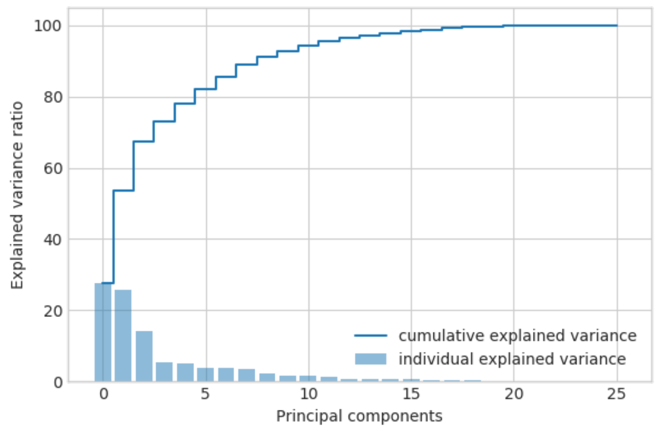


Figure 4.2. Individual explained variance of each feature illustrated with the histogram and cumulative explained variance illustrated by the blue line.

that only three player features are sufficient to explain almost 70% of the overall variance. As a consequence, we performed Principal Component Analysis (PCA) on our data using three principal components. We then computed the sum of the absolute values of each coefficients for the 25 player features and scaled each coefficient by the explained variance ratio of the corresponding component. By ordering the features based on the computed values we can observe which feature contributes the most if we want to reconstruct the data into the original space with 25 dimensions. Results showed that the average player's SR is the most important feature. Furthermore, we observed that the average amount of booster and the number of levels played are also important features to reconstruct the data. This motivates the choices that we made in the clustering simulated strategy approach where we decided to use these three features to describe the players. Furthermore we also experimented to perform a PCA with 10 components since in this way we are able to explain almost 93% of the overall variance. However as visible in Figure 4.3 we observe that the explained variance ratio significantly decreases after the third component. Furthermore, by computing again the importance of each feature we obtained similar results to the analysis with three principal components.

Another analysis that we performed was to understand how many clusters of players are present in the data. We ran the k-means algorithm on the cleaned player data with different number of clusters k and used the "elbow criterion" to analyze the result. Figure 4.4 illustrates that between four and eight are reasonable numbers of player clusters. This motivates our choice of generating six agents that simulate six different groups of players. Furthermore we performed an analysis of the levels in the game.

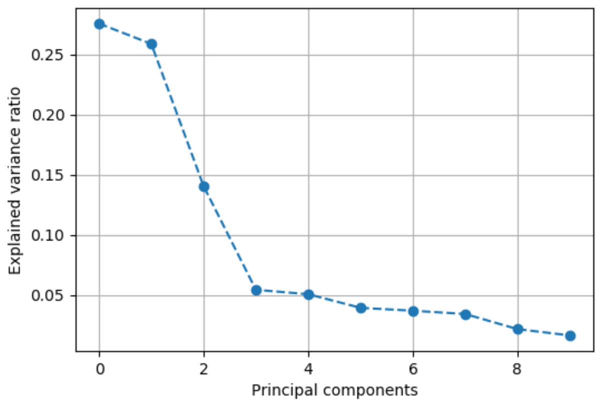


Figure 4.3. Explained variance ratio of the 10 components obtained with principal component analysis.

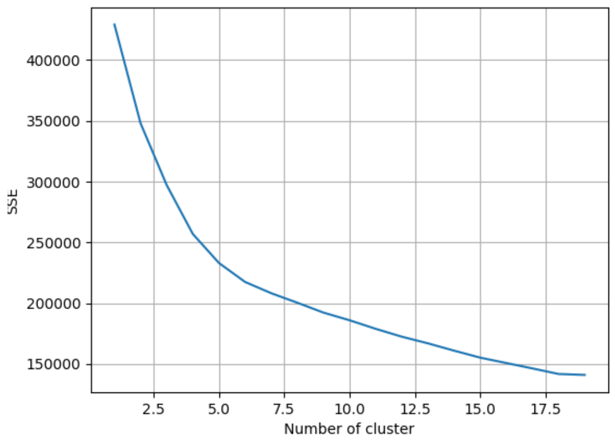


Figure 4.4. Elbow criterion to understand how many player cluster are present in the data describing the players with 25 features.

We observed the distribution of the average players' SR and their frequency. However, these data cannot be published. Finally, we performed an analysis of the distributions of the player features along the 2500 training levels. In the clustering player approach we decided to use the distribution of the standardized SR to describe each player. We also experimented with the distribution of the standardized amount of boosters and the distribution of the standardized maximum number of stars. In Figure 4.5 we illustrate the six generated centroids obtained by clustering the booster distributions.

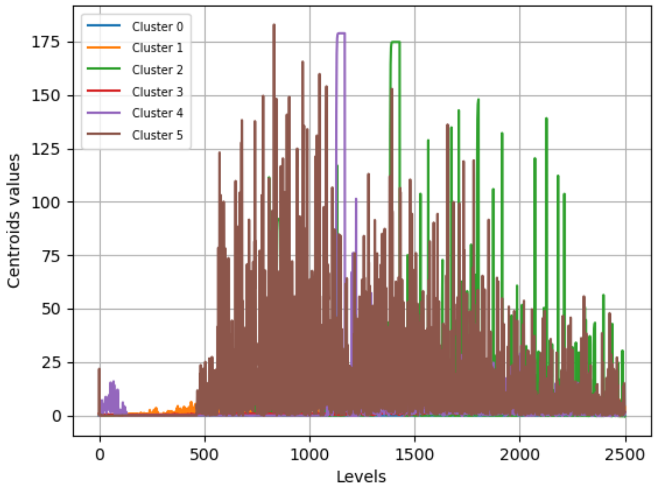


Figure 4.5. Centroids obtained by the k-means clustering algorithm on the standardized amount of booster distributions.

However, clustering by the standardized amount of booster generated very unbalanced clusters, some of them containing only a single data point representing an outlier. Furthermore, the generated clusters were not interpretable and as a consequence we decided to not continue into this direction. Clustering by the maximum number of stars generated more balanced clusters. However, since the number of stars can assume only three different values we decided to use the players' SR to represent the performances of the players in the game.

4.2 Player Modeling

We illustrate in the following section the results of the two experimented approaches regarding the modeling of the players and the creation of the various agents that simulate different strategies.

4.2.1 Clustering Players Approach

We represented each player with a one dimensional array containing all his $2,500$ standardized success rates ρ_i on the training levels. However, some players have missing SR for various reasons. The missing data points represent 5.16% of the total number of players' SR. In order to have comparable players' SR distributions, we reconstructed the missing values using linear interpolation. Figure 4.6 shows two examples of standardized players' SR distributions over the training levels. In order to select the number of player clusters k, we used the elbow criterion. By looking at the plot of the SSE against several numbers of k, as illustrated in Figure 4.7, we decided to group the players into six different clusters.

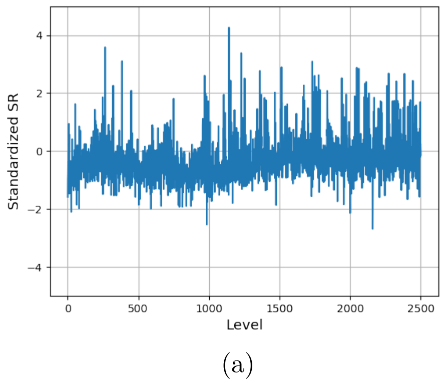


Figure 4.6. Two examples of standardized players' SR distributions over the levels in range [1, 2500].

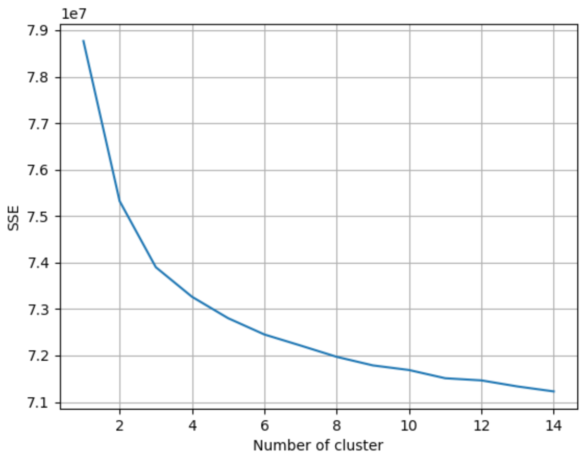


Figure 4.7. Elbow line used to select the number of clusters k. On the x-axis various numbers of k while on the y-axis the SSE of the corresponding generated clusters. The plot shows that reasonable values of k are between 3 and 8.

Subsequently, we applied k-means to the players' SR distributions. Figure 4.8 shows the centroids of the generated clusters. By looking at the centroids, we can qualitatively describe the six different types of players represented by each cluster. Players in cluster 1 (orange) show a consistently high standardized SR and the cluster can therefore represents very good players. On the contrary, players in cluster 4 (purple), show a very

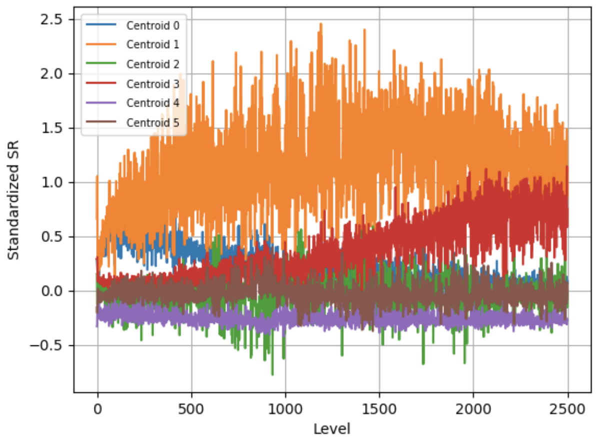


Figure 4.8. Centroids of the 6 generated clusters. The x-axis represents the training levels while on the y-axis the standardized SR. Centroid 0, in blue, represents players with decreasing standardized SR, centroid 1, in orange, represents players with high standardized SR, centroid 2, in green, represents players with average standardized SR, centroid 3, in red, represents players with increasing standardized SR, centroid 4, in purple, represents players with low standardized SR and finally, centroid 5, in brown, represents players with average standardized SR but with less variance compared to centroid 3. [This image is better viewed in colors].

low standardized SR distribution and the cluster can therefore represents less skilled players. Players in cluster 0 (blue) show a decreasing standardized SR and the cluster can therefore represents players that encounter difficulties in progressing the game. On the contrary, players in cluster 3 (red), show an increasing standardized SR and the cluster can therefore represents players that learn good strategies over time. Finally, players in cluster 2 (green) show a standardized SR distribution close to the average as well as players in cluster 5 (brown), but with slightly less variance over the levels. Since k-means generates clusters with different sizes, in Table 4.1 we reported the number of players in each cluster. Note that cluster 1 contains only 2% of the players, indicating the possibility that these players are simply outliers. To understand if the variability in the standardized players' SR distribution has an impact on the obtained clusters, we tried to cluster the levels into few buckets based on their difficulty and then represent each player with his average standardized SR distribution over the buckets. Since we obtained similar player clusters we decided to use the simplest approach avoiding to introduce a second clustering step on the levels.

Table 4.1. Player clusters

Cluster #	Color	# of Players	% of Players
0	blue	3460	10.65
1	orange	652	2.00
2	green	7042	21.68
3	red	2581	7.95
4	purple	8958	27.58
5	brown	9786	30.13

4.2.2 Clustering Simulated Strategies Approach

In this approach we decided to add other two player features: the standardized mean amount of booster $\bar{\beta}$ and the number of levels played l during the tracking period. Furthermore, instead of using the distribution of the SR along the levels, we aggregated the values to have a single measure $\bar{\rho}$, that represents the overall skill of the player. Then, we let the network detect the relationship between these features and the performed move. After training the CNN, we predicted on a validation set of 10,000 states with 125 different input player feature combinations. We generated all the possible combinations using the 5th, 25th, 50th, 75th and 95th percentiles of each player feature. The moves predicted by each agent are then considered as a sort of explicit representation of the simulated strategy of the agent. Clustering the simulated strategies we selected six agents that maximized the difference in the predicted moves. Results showed that the 25th and the 75th percentiles were never used by any of the obtained agents. This is reasonable since the player features are real numbers and extreme values lead to more diverse strategies. As a consequence we experimented repeating the clustering only with the 5th, 50th, and 95th percentiles of each player feature. Figure 4.9 shows the dendogram generated by hierarchical clustering. It shows that most of the agents have very similar predictions, however the six most different agents have a difference in the simulated strategies of at least 13%. This means that by changing the player features the network predicts differently. Table 4.2 shows the selected agents, their player features and a possible interpretation of the simulated types of players. The selected agents are the ones that on each of the six major clusters have a distance to their merged cluster that is the maximum compared to the other agents in the same cluster. As distance measure we used the hamming distance between the simulated strategies. The hamming distance represents the percentage of moves that the compared agents predict differently. We can observe that the two most diverse agents select different moves 31% of the time.

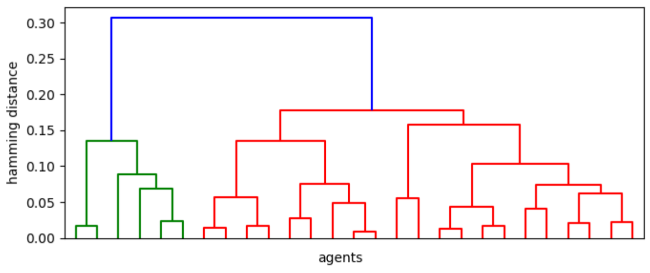


Figure 4.9. Dendogram illustrating the differences in the simulated strategies of 27 different agents. The x-axis represents the 27 different generated agents with different combinations of the standardized mean success rate $\bar{\rho}$, the standardized mean amount of boosters $\bar{\beta}$ and the number of different levels played l. The y-axis represents the percentage of moves that the agents predict differently.

Table 4.2. Agents generated by clustering simulated strategies

Agent	Skill ($\bar{\rho}$)	Levels played (l)	Boosters ($\bar{\beta}$)	Description
Agent 0	−0.76	241	−0.06	less skilled player
Agent 1	−0.76	69	1.51	less skilled occasional booster player
Agent 2	0.96	927	−0.37	skilled regular non-booster player
Agent 3	0.96	927	1.51	skilled regular booster player
Agent 4	0.05	241	1.51	booster player
Agent 5	0.96	69	−0.37	skilled occasional non-booster player

4.3 CNNs Training and Prediction

Except for the input size, we used the same network architecture for the two experimented approaches. Regarding the clustering players approach, Table 4.3 reports the top-1 accuracy, while Table 4.4 reports the top-3 accuracy of each agent. With bold we indicate the agent with the best accuracy on each data set. Each row represents a different data set. The first row represents the training accuracy of each agent on its own last mini-batch (consisting of 2048 examples). The remaining rows represent the test accuracy on different test data sets (each one consisting of 100,000 examples). By looking at the accuracy on the test data sets, we observed that in both the top-1 and top-3 accuracy, each agent better represents the players in its cluster compared to the baseline approach. This indicates that the CNN model is able to learn different policies by changing the input data used during training. However, we observed that there is only a small difference between the prediction accuracy of the agents and the baseline. This is explained by the fact that we ran experiments with thousands of different players. As a consequence, even if we divide the data, each cluster contains examples from many different players and the CNN learns the average strategy of the players on each cluster.

Combining data of such a large number of players reduces the differences in the strategies learnt. Furthermore, we noted that the baseline accuracy is different on each data set. This confirms that players in different clusters select moves with different strategies and the baseline agent is not specialized to predict any of them. Furthermore, also the agents show different accuracy. A possible explanation is that some players select moves that are more predictable than other players. For example, players in cluster 2, select moves that are more easily predictable by the CNN-based model than any other player cluster since the agent 2 performed better than any other agent in the top-1 accuracy. Nevertheless, all the agents showed a similar training progression and as an example, in Figure 4.10 we illustrate the training process of two of them. Finally, if we use the agents to predict on the same game boards, on average they disagree 30% of the times, meaning that approximately one third of the times they select different moves between each other. This confirms once again that we learned different player strategies. Note that the prediction accuracy of the model does not directly relate to their performances in the game. An agent can be very good in predicting the players that simulates but if the moves performed by that players are bad, the agent will perform poorly when playing the game. With a similar reasoning, even the improvement of the agents' accuracy compared to the baseline agent on each player cluster does not necessary mean that the agents are better than the baseline during gameplay. The tables only shows that each agents better predict the players that simulate compared to the baseline agent and that different player cluster use different strategies.

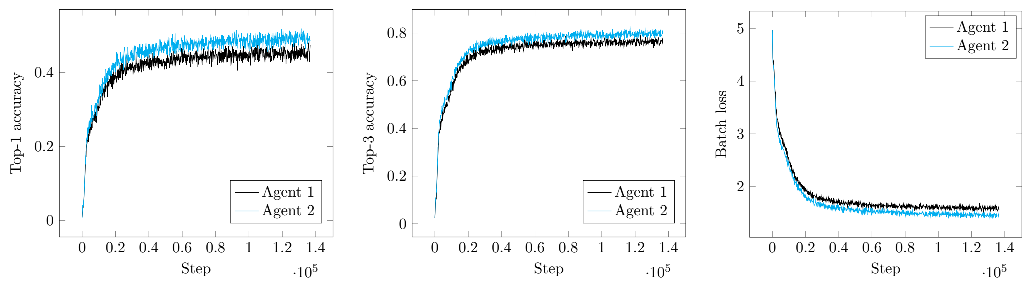
Regarding the clustering simulated strategies approach, Table 4.5 illustrates the top-1 and top-3 test and training accuracy. We reported the accuracy for both the network that uses the player features and the baseline. The maximum disagreement between the generated agents is 31%, meaning that the CNN has learnt to play differently based on the values of the player features. Finally, we observed that the test accuracy in the clustering player approach is higher than the test accuracy in the clustering simulated strategies approach. A possible explanation is that in the first approach we restricted

Table 4.3. Clustering players. Top-1 training and test accuracy

Data set	Agent 0	Agent 1	Agent 2	Agent 3	Agent 4	Agent 5	Baseline
Training data	**52.97**	47.46	48.77	46.39	49.10	51.78	49.74
Test cluster 0 data	**51.93**	-	-	-	-	-	51.16
Test cluster 1 data	-	**47.06**	-	-	-	-	46.02
Test cluster 2 data	-	-	**53.03**	-	-	-	51.97
Test cluster 3 data	-	-	-	**46.78**	-	-	45.54
Test cluster 4 data	-	-	-	-	**51.95**	-	51.85
Test cluster 5 data	-	-	-	-	-	**52.89**	52.49
Test baseline data	50.11	49.52	**50.56**	50.14	49.86	50.53	50.11

Table 4.4. Clustering players. Top-3 training and test accuracy

Data set	Agent 0	Agent 1	Agent 2	Agent 3	Agent 4	Agent 5	Baseline
Training data	82.35	77.15	80.40	77.35	81.24	**82.82**	81.01
Test cluster 0 data	**82.79**	-	-	-	-	-	82.46
Test cluster 1 data	-	**78.26**	-	-	-	-	77.50
Test cluster 2 data	-	-	**83.63**	-	-	-	82.99
Test cluster 3 data	-	-	-	**77.73**	-	-	76.74
Test cluster 4 data	-	-	-	-	**83.68**	-	83.45
Test cluster 5 data	-	-	-	-	-	**83.75**	83.56
Test baseline data	81.47	81.11	**81.96**	81.80	81.52	81.52	81.56



(a) (b) (c)

Figure 4.10. Visualization of the training process for agents 1 and 2. Figure (a) illustrates the top-1 training accuracy, Figure (b) shows the top-3 training accuracy while Figure (c) illustrates the batch loss.

our focus to players that finished all the training levels and the moves performed by these players are more predictable than the ones performed by random players.

4.4 Players' SR Prediction

Throughout this thesis, in order to not expose the actual values for privacy purposes, the players' SR has been scaled by the difference between the maximum and the minimum SR as described in Equation 3.8. Note that the agents' SR have not been scaled and that on both the scaled players' SR and the agents' SR we applied a log transformation to obtain less skewed distributions. Furthermore, we used linear regression models to predict the log of the scaled players' SR when the agents' SR is greater than zero and the mean of the log of the scaled players' SR on training levels as a prediction when the agents' SR is zero. For comprehensibility, in the rest of the thesis we will refer to the log of the scaled players' SR as simply the players' SR and we will refer to the log of the agents' SR as simply the agents' SR.

Table 4.5. Clustering simulated strategies. Top-1 and top-3 training and test accuracy of the agent trained with player features and the baseline

Data set	Accuracy	Player Features Agent	Baseline
Training data	Top-1	**46.52**	46.04
	Top-3	78.08	**78.29**
Test data	Top-1	**46.28**	46.11
	Top-3	**78.01**	77.82

4.4.1 Clustering Players Approach

In order to validate our models we performed various tests. First, we tested that there exists a statistical relationship in the training data between each agent's SR and the players' SR that each agent simulates. Since each agent uses only a single predictor, in Table 4.6 we reported its coefficient and the regression statistics. We can observe that the p-values are all lower than 5% meaning that is likely that changes in the predictor are related to changes in the response variable. In our case this means that changes in the agent's SR are likely to be related to changes in the players' SR. Note that each line in the table refers to a different linear model that correlates the agent's SR with its corresponding players' SR.

Table 4.6. Linear regression analysis between each agents' SR and its corresponding players' SR in the training data

Agent	coef.	std. err.	t-stat.	p-value
Baseline	0.696	0.006	120.84	0.00
Agent 0	0.618	0.005	127.08	0.00
Agent 1	0.317	0.004	71.55	0.00
Agent 2	0.668	0.006	105.54	0.00
Agent 3	0.528	0.006	87.40	0.00
Agent 4	0.800	0.006	124.90	0.00
Agent 5	0.689	0.006	124.64	0.00

Since we used linear regression models to predict the players' SR we also need to ensure that the following assumptions are valid:

1. Linearity of the relationship between dependent and independent variables
2. Homoscedasticity of the errors
3. Statistical independence of the errors
4. Normality of the error distribution

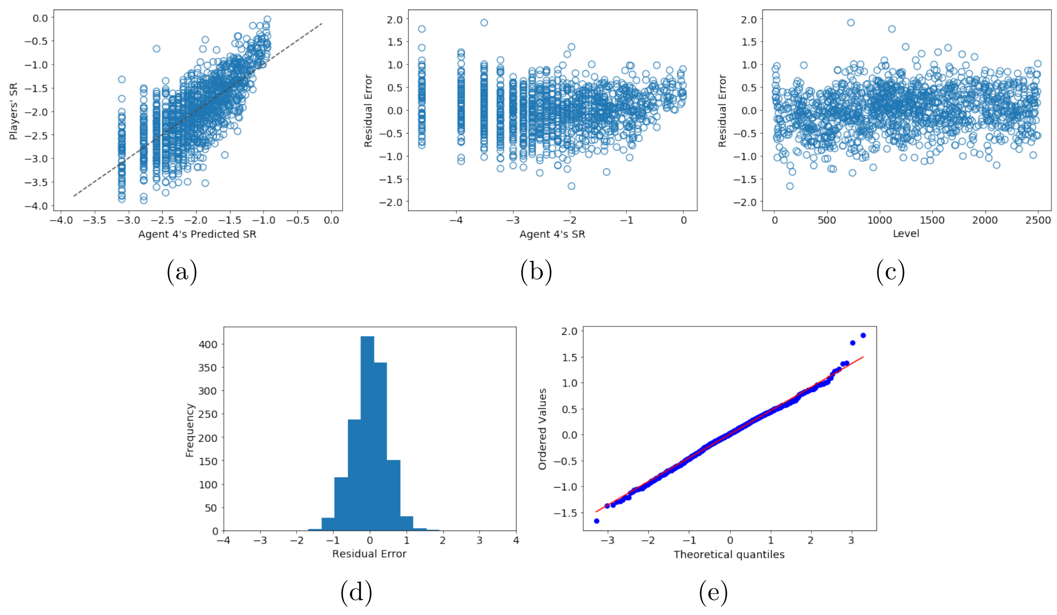


Figure 4.11. Plots validating the linear regression assumptions in the training data for agent 4. Figure (a) illustrates the linear relationship between the players' SR and the predicted players' SR, the black dashed line is the diagonal. Figure (b) shows the agent's SR against the residual errors while Figure (c) shows the levels against the residuals. Figure (d) shows the histogram of the residual errors and finally, Figure (e) shows the normal probability plot of the residuals.

As an example, we check these assumptions for the linear regression model trained with agent 4's SR. The same tests are performed with all the remaining linear models and the plots are illustrated in Appendix A.3.1. Note, that these analysis are performed on the training data before using the linear models for prediction. Plotting the agent 4's predicted SR against the players' SR, as illustrated in Figure 4.11a, we validated that exists a linear relationship between the dependent and the independent variables. Except for very high values of the agent's SR, the data points are symmetrically distributed around the diagonal line. This implies that there is no major violation of the linear relationship assumption (1). Figure 4.11b illustrates the agent's SR against the residual errors. The variance of the errors seems to be fairly constant along the independent variable, except for very high agent's SR where it seems to be lower. For completeness, we also checked that the variance of the errors is constant along the dependent variable and we obtained similar results. As a consequence, the variance of the errors does not significantly increase as a function neither of the predicted players' SR nor the agents' SR. However, the residual errors seem to be centered around zero only for values of the agent's SR between -1 and -4. For very low or very high agent's SR the errors are not centered around zero. More precisely, when the agent's SR is very low or very high, the residual error is more likely to be positive, meaning that we underestimate levels that are very difficult or very easy for the agent. However, since the data points that violate the homoscedasticity assumption (2) are only a small number compared to the total number of training data points, we concluded that there is only a minor violation of this assumption. In Figure 4.11b, we can also observe that consecutive errors are not correlated, therefore there is no violation of the statistical independence of the errors assumption (3). Figure 4.11c shows that the errors are equally distributed along the levels as well. Figure 4.11d illustrates the histogram of the residual errors. We can notice that the histogram seems to represent a normal distribution. Finally, we checked against violation of the normal distribution of the errors assumption (4) by looking at the normal probability plot [99] illustrated in Figure 4.11e. We can see that only a few points at both the tails of the theoretical quantiles do not follow the line. As a result, we can say that there is no violation of the normal distribution of the errors assumption.

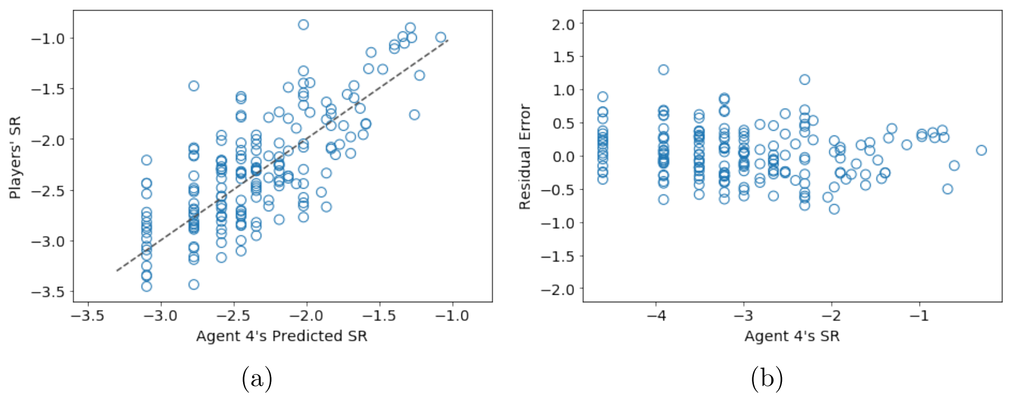


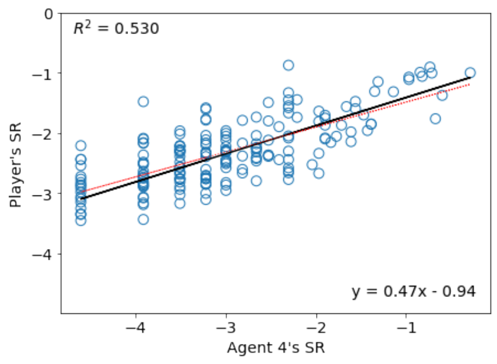
Figure 4.12. Plots confirming the linear regression assumptions in the test data for the agent 4. Figure (a) illustrates the linear relationship between the players' SR and the predicted players' SR, the black dashed line is the diagonal. Figure (b) shows the agent's SR against the residual errors.

Similar conclusions can be derived for the other linear models with only few exceptions. The linear models trained with agent 1's and agent 3's SR showed a small correlation between the levels and the residual errors. This is not surprising since the players that they simulate have a non-constant average SR in the training data. This implies that for these two specific agents, adding the levels as a feature of the linear model could improve the performances. Moreover, the linear model trained with agent 1's SR violates the normality of the error distribution assumption. However, this agent represent only 2% of the total number of players and we knew that it could represents outliers or anomalous players. Since agent 1 and 3 represent only 9% of the players in total, for practical purposes, a lower performance of the linear models for these agents is tolerable. Finally, similar plots for validating the assumptions with the baseline agent are reported in Appendix A.3.1. Subsequently, we use the fitted linear models to predict on test levels. In Figure 4.12 we observe that the assumption we validated in the training data for the linear models still hold in the test data. More specifically, Figure 4.12a shows that there is a linear relationship between the agent's and the players' SR in the test data while Figure 4.12b shows that the residual errors are well distributed, not correlated and with a fairly constant variance. In Figure 4.13, as an example, we illustrate the SR predictions of the players in cluster 4 for both the baseline and agent 4. The comparisons between the other agents and the baseline while predicting the remaining player clusters are reported in Appendix A.3.2. We observe that the linear model fitted with the agent 4's SR significantly outperforms the baseline in predicting the SR of players in cluster 4. Moreover, we observe that the baseline approach systematically overestimates these players. This is explained by the fact that cluster 4 represents players which have a SR distribution that is consistently under the average value. Similar reasoning can be done looking at the others agents' predictions compared to the baseline. Furthermore, Figure 4.13c shows that even if we train the baseline agent to predict the SR of players in cluster 4 instead of predicting the average SR between all the players, the prediction performances remain lower than those of agent 4. This confirms that the SR of players in cluster 4 are more correlated to the SR of agent 4 rather then the SR of the baseline agent. However, if we train the baseline agent to predict the SR of players in cluster 4 the difference with our approach in terms of SR predictions is small. This suggest that after clustering the players, using only the baseline agent to predict each player cluster SR could be a trade-off between computational requirements and prediction accuracy.

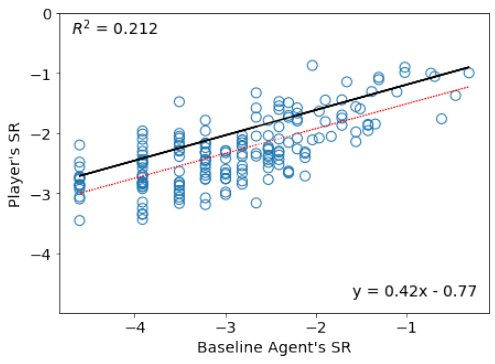
Table 4.7 reports the MAE, MSE and adjusted R-squared computed on each player cluster. Note that on each row the "agent" columns describe a different agent, the one trained with the corresponding player cluster. For example, we never use agent 1 to

Table 4.7. Analysis of the linear regression models while predicting on test data sets and comparison with the baseline agent

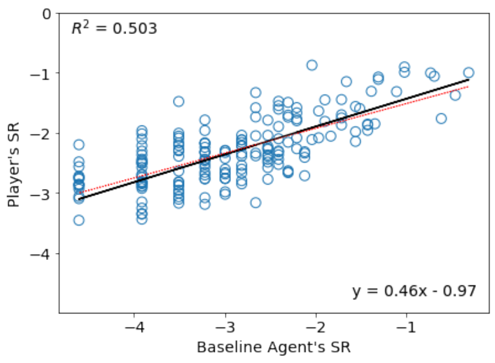
	MAE		MSE		Adj R-squared	
Players in	Agent	Baseline	Agent	Baseline	Agent	Baseline
Cluster 0	0.27	0.34	0.12	0.19	0.47	0.12
Cluster 1	0.34	0.85	0.18	0.87	−0.27	−5.26
Cluster 2	0.33	0.34	0.16	0.17	0.34	0.30
Cluster 3	0.40	0.91	0.22	0.94	−0.99	−7.75
Cluster 4	0.31	0.41	0.15	0.26	0.49	0.13
Cluster 5	0.30	0.31	0.14	0.16	0.50	0.44



(a)



(b)



(c)

Figure 4.13. Agent 4's and baseline's predictions for the success rate of players in cluster 4. Figure (a) illustrates the agent 4's predictions while Figures (b) and (c) show the predictions obtained from the baseline's SR. The baseline predictions in (b) are obtained training the baseline linear model to predict the average SR of the players. The baseline predictions in (c) are obtained training the baseline linear model to predict the SR of the players in cluster 4. In each graph, the black line shows the linear regression model while the red dashed line shows the ideal linear model. In the top left corner we illustrate the R^2 measure of the fitted regression model. In the bottom right corner of the graph we report the equation of the line obtained by the linear regression model.

predict the SR of players in different clusters from cluster 1. On the contrary, the baseline columns indicate always the same baseline agent used to predict the SR of players in all the defined clusters. We can observe that each linear model, compared to the baseline, better represents, in terms of SR predictions on test levels, the players that the agent simulates. Note that the adjusted R-squared is 1 in the ideal case where the line perfectly fits all the data points but it does not have a lower bound. This means

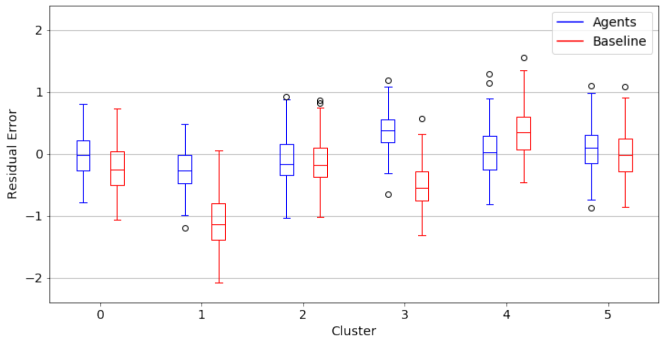


Figure 4.14. Boxplot comparing agent's and baseline's residual errors. For each player cluster illustrates the distribution of the residual errors of the agent representing the cluster, in blue on the left, and the baseline approach, in red on the right.

that as the linear model get worse, the adjusted R-squared value can become negative and it can decrease indefinitely. A boxplot illustrating the residual errors of the agents and the baseline on each player cluster is shown in Figure 4.14. It shows that the agents have an error closer to zero and a lower variance, confirming that they can better predict the players' SR that they simulate compared to the baseline.

Finally, in order to compare our approach with the baseline approach and answer our research question we combined the agents' predictions to obtain a single measure. The linear combination of the six predictions, as defined in 3.9, is based on the percentage of players that each agent represents. Table 4.8 reports the evaluation of our approach compared to the baseline. The clustering players approach reduced the prediction MAE from 0.30 to 0.26 with an improvement of 13% and reduced the prediction MSE from 0.13 to 0.10 with an improvement of 23%. In the table we also reported the 95% confidence interval of the mean. We ran a paired t-test to check that the mean difference between the absolute errors per each test level is different from zero. We decided to use a paired t-test because the two approaches are tested on the same levels and we want to remove the variability in the errors that has not been completely removed by the linear regression models due to the minor violation of the homoscedasticity assumption. The paired t-test increases the statistical power of our analysis because it removes the between-level variability. If a level is too difficult, the difference between the approaches does not have to be large even if the error is big. As a result, we obtained a two-tailed p-value of 0.04%. Since it is lower than 5% we rejected the hypothesis of no difference between the absolute errors of the two approaches. For completeness, we ran a two-sided t-test to check that the two approaches have a statistically different MAE. Since we obtained a p-value lower than 5%, we can reject the null hypothesis of identical average scores. Similar results were obtained for the MSE. As a consequence, we have strong evidence that the new approach outperforms the baseline approach. The MAE is the metric that for practical purposes we consider most important between the two. We prefer to have few wrong predictions and frequent accurate predictions rather than constantly fairly accurate predictions. This because human testing is still worth when concerning subjective or qualitative measurements like fun or player experience and if applied in conjunction with our approach, it can easily detect highly wrong predictions.

Table 4.8. Clustering players. Overall linear regression performance measures

Approach	MAE	MSE
Baseline	0.30 ± 0.03	0.13 ± 0.02
Clustering players	0.26 ± 0.03	0.10 ± 0.02

Predictions using the mean of the players' success rate

Regarding data points where the agents or the baseline have a zero SR, a prediction is formed using the mean of the players' SR where the corresponding agent's or baseline's SR are zero in the training data. This means that for each agent, the predictions in the test levels where the agent failed are computed by looking at the mean of the players' SR in the training levels where the corresponding agent failed as well. In Table 4.9 we reported both the performance in the linear part and the overall prediction performance. We can say that our approach reduced the MAE by 7% and the MSE by 4% when considering both the linear and the mean predictions. Furthermore, we reported the 95% confidence interval of the measurements. Note that the improvement of our approach is lower when considering the mean predictions as when considering only the linear part. Since the agents and the baseline failed in different levels, we need to consider that the number of levels predicted with the mean is different between each agent and the baseline. On average the six agents predict 62 levels using the mean while the baseline predicts 69 levels with the mean. We ran a paired t-test to check that the within pair difference obtained is larger than would be expected to have occurred by chance. Since we obtained a two-tailed p-value greater than 5% we cannot reject the null hypothesis when considering also the predictions realized with the mean SR.

Table 4.9. Clustering players. Regression performance measures

	MAE		MSE	
Approach	Linear part	Total	Linear part	Total
Baseline	0.30 ± 0.03	0.31 ± 0.02	0.13 ± 0.02	0.14 ± 0.02
Clustering players	0.26 ± 0.03	0.29 ± 0.02	0.10 ± 0.02	0.13 ± 0.02

Figure 4.15 illustrates the distribution of the SR for the failed levels of each agent as well as the mean used to predict in these cases. The players' SR in the failed levels tend to be lower compared to the succeeded levels. This is desirable since we expect levels where the agents failed to be difficult for the players as well. However, the variance is high and as a consequence the mean prediction performs poorly. Furthermore, since the failed levels represent on average 18% of the test levels, their impact on the regression performances is considerable. Finally, we can observe that the predicted mean in the baseline approach is closer to the true mean compared to other agents. This is due to the fact that the baseline represents a greater number of players and their average SR is less variable between training and test levels. On the contrary, the average SR on failed levels for some agents, especially agents 1, 2 and 3, changes a lot between training and test levels. This was expected by looking at the SR distribution in Figure 4.8 and it can explain why adding these levels reduces the performances of our approach.

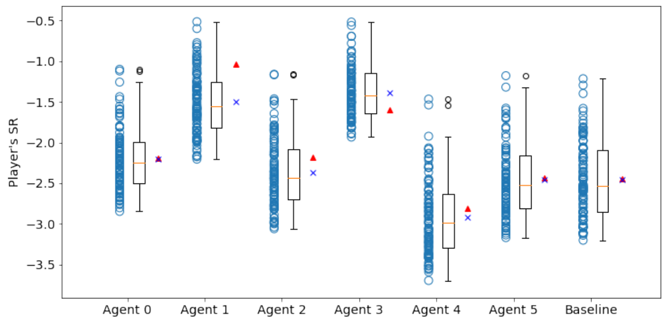


Figure 4.15. Data points and boxplots of levels predicted with the mean. The red triangles show the mean prediction of each agent while the blue crosses indicated the true mean.

4.4.2 Clustering Simulated Strategies Approach

In this approach, since we cannot accurately determine how many players each agent simulates, instead of combining the predictions of six different linear models, we use a single linear regression model with six predictors. Another difference with the previous approach is that we no longer need to eliminate levels that drastically changed since both the players' SR and the agents' SR are computed using the most recent and identical version of the levels.

As we did for the clustering players approach, we validated the four linear regression assumptions in the training data and we reported the plots in Appendix A.4.1. We can observe that the assumptions hold with only a minor violation of the homoscedasticity of the errors assumption. Subsequently, we used the linear model fitted with the six agents' SR and the one fitted with the baseline's SR to predict on test levels. Table

4.10 reports the evaluation of our approach compared to the baseline. The clustering simulated strategies approach reduced the prediction MAE from 0.30 to 0.26 with an improvement of 13%, the prediction MSE from 0.17 to 0.13 with an improvement of 24% and finally improved the adjusted R-squared from 0.51 to 0.62 with an improvement of 22%. We ran a paired t-test to check that the per-level mean difference of the absolute errors between the two approaches is different from zero with a significance level of 5%. We obtained a two-tailed p-value of 0.08% and rejected the null hypothesis concluding that the mean difference between the two approaches is statistically significantly different to zero. For completeness, we ran a two-sided t-test to check that the two approaches have a statistically different MAE. We obtained a two-tailed p-value lower than 5% and we rejected the null hypothesis of identical average scores. Furthermore, since in this approach we compared a linear model with six predictors with a linear model with a single predictor, the most important metric is the adjusted R-squared because it takes into account the number of predictors. More precisely, the adjusted R-squared decreases if we add a predictor that improves the model less than what would be improved by chance.

Table 4.10. Clustering simulated strategies. Overall regression performances

Approach	MAE	MSE	Adj R-squared
Baseline	0.30 ± 0.04	0.17 ± 0.06	0.51
Clustering simulated strategies	0.26 ± 0.03	0.13 ± 0.05	0.62

Predictions using the mean of the players' success rate

The data points where any of the agents or the baseline failed are predicted using the mean of the players' SR in the training levels where the corresponding agents or baseline failed as well. In Table 4.11 we reported the prediction performances in the linear part and the overall performances. The six agents predicted 193 levels using the mean since whenever one of the agents failed a mean prediction is applied, on the contrary, the baseline predicted only 101 levels using the mean players' SR. Considering all predictions, the clustering simulated strategies approach achieved a 8% lower MAE, a 16% lower MSE and a 19% higher adjusted R-squared than the baseline approach. However, we ran a paired t-test and we obtained a two-tailed p-value greater than 5%. As a consequence, we cannot reject the null hypothesis of mean difference identical to zero when considering also the predictions realized with the mean SR with a significant level of 0.05.

Note that the performance metrics reported in this thesis are computed on the scaled and log-transformed values. Reverting the SR predictions and computing the errors on the original SR values only led to different absolute values yet similar improvements between the proposed approaches and the baseline.

Table 4.11. Clustering simulated strategies. Regression performance measures

	MAE		MSE		Adj R-squared	
Approach	Linear part	Total	Linear part	Total	Linear	Total
Baseline	0.30 ± 0.04	0.35 ± 0.03	0.17 ± 0.06	0.22 ± 0.05	0.51	0.46
Clustering sim. str.	0.26 ± 0.03	0.32 ± 0.03	0.13 ± 0.05	0.18 ± 0.04	0.62	0.54

4.5 Estimating Strategy Requirements

We performed an analysis to categorize levels based on the strategy required to solve them. This approach is built on the assumption that the strategy learnt from player data is enough to solve the levels. Nevertheless, we observed that on average between agents, 82% of the tested levels are successfully solved. We created a population of 16 different agents, each one simulating a different amount of strategy error. By comparing the performances of the agents, we estimate the strategy required by each level. We randomly selected 20 levels to test this approach and we ordered the agents that use an ϵ-greedy policy from the one that plays completely random ($\epsilon = 1$) to the one that plays greedily ($\epsilon = 0$). By plotting the agent's SR against the amount of strategy used, represented by $1 - \epsilon$, we observed how the SR improves as we reduce the number of random moves made by the agent. Figure 4.16 shows for levels 154 and 155 the strategy level plotted against the agents' SR. The strategy level represents the amount of strategy used by the agent. As a consequence a strategy level of zero means that the agent uses a random policy while a strategy level of one means that the agent follows a greedy policy. Note that the agents' SR are normalized dividing them by the maximum value. This because in order to compare levels we assume that when the agents play with full strategy they reach the optimal performances ($SR = 1$). To estimate the strategy required by a level, we computed the area above the line. By computing this metric for the two levels in Figure 4.16 we obtained a strategy requirement of 0.72 for level 154 and a strategy requirement of 0.24 for level 155.

Finally, Figure 4.17 illustrates the scaled players' SR against the strategy required. We scaled the SR dividing by the difference between the maximum and the minimum SR to not show the actual values for privacy reasons. We observe that the strategy requirement is correlated with the players' SR. Although some levels have a similar SR, they have significantly different strategy requirements. This indicates that even if two levels have a similar difficulty, the concentration and focus required by players to solve them is different.

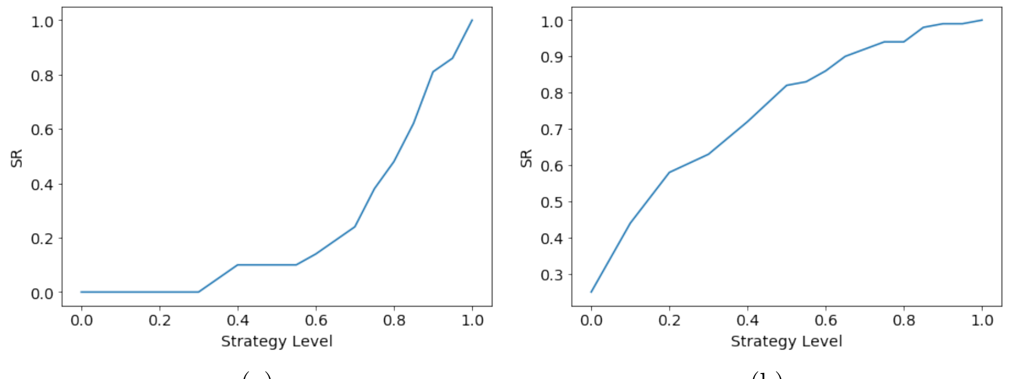


Figure 4.16. Two examples of strategy requirement plots for two random levels. Figure (a) shows a level (number 154) that has a high strategy requirement while (b) shows a level (number 155) that requires less strategy to be solved. The SR on the y-axis is normalized dividing it by the maximum value.

We found that *Jelly levels* have on average a higher strategy requirement compared to other types of levels even with a similar SR. However, this difference in the strategy requirement is present only on easy levels (with a high players' SR). This is explainable by the fact that in order to solve them, the player has to perform the moves above the jelly tiles and performing random moves in other positions reduces the possibilities to clear all the jellies and finishing the level. Regarding more difficult levels, the simple strategy of performing the moves above the jelly tiles is not enough and more complex reasoning is needed. This explains the fact that for difficult levels there is no a clear difference in the strategy requirements between different types of levels. Finally, we can hypothesize that levels that can be successfully solved also with random moves might be

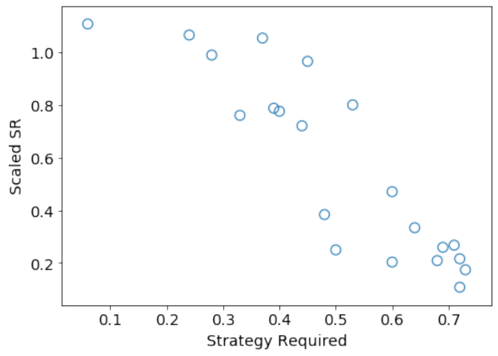


Figure 4.17. Players' SR against the strategy required for the 20 different tested levels.

perceived less challenging by some players or that levels that require too much strategy might be an obstacle to some other players. A deeper analysis is needed to verify these hypothesis. Furthermore, in the proposed analysis we only uses a single agent and modified it by adding different amount of human errors. In future work we will increase the diversity in the population of developed agents by adding human errors not only to one agent but to all the agents obtained with the two player modeling techniques proposed in this thesis.

Chapter 5

Discussion

5.1 Combining the Agents with a Single Regression Model

In the clustering player approach, in order to have a fair comparison with the baseline that uses only a single predictor, we decided to use six linear models and combine their predictions afterwards. However, we experimented also to use a single linear model that takes as input all the six agents' SR. We found that this model has a very similar MAE and MSE of our approach. However, looking at the p-value of each predictor we found that agent 2 has a p-value of 0.61, meaning that its predictions are not improving the model and they might be highly correlated to other agents' predictions. By removing this agent from our approach, we found that all the remaining predictors have a p-value that is lower than 5% while the performance of the approach remained unchanged. A similar result is obtained in the clustering simulated strategies approach. The lowest p-value between the six predictors was 0.08 and removing the corresponding agent did not alter the results. This is beneficial not only for simplicity, but also for sustainability, since each time a new game content needs to be tested we have to simulate gameplay with all the developed agents. By reducing the number of simulations we will reduce the computational resources utilized. Furthermore, by providing more accurate predictions, our approaches can reduce the number of iterations required by game designers to adjust a level. However, we need to consider that the computational requirements of our approach grow linearly with the number of simulated groups of players. From the sustainability point of view, a deeper analysis of the trade-off between number of iterations and number of simulations per iteration is necessary to answer if the proposed approach is more or less sustainable than the baseline approach. Finally, since the various simulations can be executed in parallel the time required to test a game content is the same as the baseline approach.

5.2 Comparison of the Two Approaches

Even if it is not the purpose of this thesis a comparison between the two approaches may be of interest. Nevertheless, we need to remind that the agents in the two approaches are trained on different data sets and predict on different test levels. By looking at the comparison of the two approaches with the baseline, we observe that they produce very similar improvements in the linear part. On the contrary, if we consider both the linear predictions and the mean predictions the clustering players approach only slightly improve the baseline yet being better in absolute numbers than the clustering simulated strategies approach. However, the diversity may be due to the different test levels considered in the two approaches. Note that in the clustering players approach we had to remove levels that drastically changed during time. Indeed, comparing the two approaches with the same test levels we obtained similar results. The clustering players approach favors description and interpretation, meaning that the various agents directly simulate a specific group of players. This means that if a particular behaviour is discovered by the simulation of an agent, it is possible to go back to the players that showed this behaviour. However, it requires that meaningful player clusters are created and more importantly, that players in different clusters use, on average, different strategies. On the contrary, the clustering simulated strategy approach works better when the relationship between player features and strategies is unknown. By looking at the player features of the selected agents, it is possible to qualitatively determine which player each generated agent represents. Furthermore, during training, this approach uses all the available data and should be preferred when gameplay data are limited. However, one of the major drawback of this approach is that the feature combinations tested to select the agents grow exponentially with the number of player features. A possible solution could be to randomly select few of them or use evolutionary algorithm to explore the feature space and select the agents. In this research we aimed to model different type of players and as a consequence we selected the agents that led to more diverse strategies, however, it may be of interest to select agents based on their performance in the game or other metrics.

5.3 Practical Implications

Accurately determining the difficulty of a level is crucial for game companies since the difficulty has an impact on the revenues, both directly, in terms of boosters bought by the players and indirectly, in terms of player satisfaction and entertainment. For example, if a too difficult level is included in the game, there is a risk that frustrated players may stop playing the game. Furthermore, being able to estimate the perceived difficulty of a specific group of players, can provide an indication of how many players could be affected by a too difficult level. Additionally, it allows game designer to create levels with a desired difficulty for specific players. As an example, a game designer may want to create a level that is easy for everyone while another game designer may want to develop a level that is easier for regular players than for occasional players. Furthermore, it is possible to allow the CNN-based agents to play extra moves after the maximum number of available moves is reached. Since the maximum number of moves is a parameter that game designers need to decide when creating a level, simulating longer gameplay allows to compute an estimate of how many moves to add if a level is considered too difficult for a specific type of players or globally too difficult. Furthermore, since players have the possibility to buy boosters that give them extra moves at the end of an attempt, we can estimate the potential impact of this type of boosters on different players.

The experiments in this thesis were performed using a cloud service provider. Thanks to the fact that the training of the agents and the simulation of the gameplay can be executed in parallel, the proposed approaches scale very well with an increase of computational power. Furthermore, not only the simulation of different agents were executed in parallel, but for each agent, several copies were executed, each one simulating several attempts of the same level. As a consequence, we experimented using 1,000 attempts per level and preliminary results showed a little improvement of the players' SR prediction. Finally, a deeper analysis on the agents' performance discovered that for levels that contain a special item called "Sugar Key", the agents' SR is constantly lower than the players SR. The "Sugar Key" are key-shaped candies, which can be matched like regular candies, however they cannot be matched with any special candy. The "Sugar Key" is always present with one or more "Sugar Chest". Every time a key is destroyed, a layer of lock on the chests is removed. Removing all the chests is usually a necessary condition to complete the level. As a consequence, the lower agents' performance can be explained by the fact that a different or deeper *strategic thinking* is required when this element is present on the game board, or it might be a lack of training data with these special items since they appear only on 8% of the levels. Also the "Sugar Key" can appear in six different colors and for each color it is considered as a different input feature for the CNN. A deeper analysis is necessary to better understand this issue.

5.4 Generalization of the Results

Despite the fact that we tested our approaches using the *Candy Crush Saga* game, we believe that the same approaches, with few variations, can be applied to other games as well. Using player features to differentiate gameplay simulations and better predict players' behaviour is a general concept that can apply to many games. Furthermore, the proposed approaches could have an even bigger impact on those games where the features of the players have a stronger relationship with the player strategies [15].

Drachen *et al.* [45] in their research, showed how the selection of the clustering algorithm can lead to different insights of the players population. In the clustering players approach, we used k-means to extract the general distribution of players' behaviour. By using others clustering algorithms, like Simplex Volume Maximization, we could identify and model players with extremes behaviours and use them to obtain different insights about the game. Furthermore, as illustrated by Holmgård *et al.* [16], the developed agents could be useful to characterize and classify new players by looking at which agent better represents the player decision making style. In this thesis we focused on the players' SR as a metric to evaluate new game content. However, the proposed approaches can be used to estimate other metrics. As an example, in *Candy Crush Saga*, we can use the developed agents to estimate the score distribution and automatically set the thresholds for reaching one, two or three stars on each level. These parameters are usually set and fine-tuned by game designers. Moreover, especially in other games, the proposed approaches could also work with completely different metrics.

Finally, the developed agents can be used for several purposes. Similarly to what Holmgård *et al.* [15] did in their research we can visualize the gameplay simulation to observe how different players interact with the game or we can use the agents to perform game balancing [10], providing each player with a slightly different level that matches their performances, skills and expectations.

5.5 Limitations

In this thesis, due to the properties of the experimented game, we used player features that are indirectly related to the player strategies. Defining, tracking and computing player features that directly relate to the performed moves, e.g. number of special candies created, could be useful to improve the developed player models. Moreover, in this work we did not consider the direction of the performed moves. Adding the possibility for the agents to chose between left-right or right-left moves and top-down or bottom-up moves can potentially increase the performances of the developed agents. Furthermore, we could also add the possibility for the agents to use in-level boosters by increasing their action space. Another possible limitation is that the presented approaches can only be used in games where the content evolves during time since they use data from existing game content to predict on future ones. Nevertheless, in most of the F2P games the game content is continuously added in order to engage and retain players. In addition, since we simulate gameplay using CNN-based agents, the experimented approaches work better with games that have states that can be represented with images or encoded in a grid-shape topology, games that have a discretizable action space and games that have the Markov property. Finally, a last limitation regards the models used to predict the players' SR. In this thesis, we used linear regression models. However, using more complex models could improve the prediction performances and a possible approach is discussed in the next section.

5.6 Future Work

5.6.1 Players' SR Prediction with Generalized Linear Models

To overcome the limitations of the linear regression models in the players' SR prediction stage we could try to use Generalized Linear Models (GLMs). GLMs are helpful, especially when the range of the dependent variable y is restricted and the variance is not the same along the predicted values but depends on the mean of y. A generalized linear model is built with a linear predictor and two functions: a link function describing how the mean of y depends on the predictor and a variance function describing how the variance depends on the mean. The general linear regression applied in this research is a particular case of GLM with identity link function and unit variance. In our case, since the predicted SR is a ratio and assumes values between 0 and 1, it could be better modeled by a binomial distribution. As a consequence a possible approach could be to use a GLM with *logit* as link function and variance function $V(\mu_i)$ defined as:

$$V(\mu_i) = \mu_i(1 - \mu_i), \tag{5.1}$$

where μ_i is the expected value of y. A preliminary experiment showed that a GLM with binomial distribution improves the MAE of both the proposed approaches. Furthermore, it could be worth to model the extra uncertainty that derives from the fact that the agents play deterministically and as a consequence the agent's SR is self-correlated. Meaning that there are some levels where the agent is always good or always bad while when we aggregate the players we average between different SR. More work in this direction could be done to improve the performances of the proposed approaches.

5.6.2 Further Applications

The benefits of having multiple agents simulating different strategies are not limited to better estimate the players' SR on new levels. A first extension is to compute different metrics while other applications are reported in this section and future work should deepen these aspects.

Clean the data

While tracking gameplay data from players we are aware that also "noise" in the data is tracked as well. As an example, sometimes players might select moves randomly, without focusing on the game and this data are tracked as well. Having multiple agents simulating different strategies could be beneficial to reduce this noise. Our approaches help to directly model the diversity in the player strategies. However, while modeling player strategies we inevitably include the tracked "noise" in the simulated behaviour. This is especially true for casual games that can be played without too much concentration. A possible approach to discover moves that are selected randomly, is to look at the time spent by the players to select each move and compare it with the average time spent. If a move is selected very fast, this could indicate that it is just picked randomly. Furthermore, some gameplay data are more useful to learn different player strategies than others. If all the agents predict the same move, this means that independently from which strategy they simulate the players more likely select that specific move. Results showed that 23% of the time all the developed agents predict the same move. It could indicate that these are states where a specific move is clearly preferred by the players. Maybe because there is only one legal move available or maybe because the predicted move leads to win the game. Since no difference is learnt from the CNNs in these states, removing them and re-training the networks could potentially increase the differences in the simulated strategies. Finally, we need to consider that we are simulating the strategies of groups of players and not individual ones. As a consequence it is possible that two different players, even if they belong to the same cluster, select two different moves in a given state. It is also possible that a player selects different moves even in the exact same state in two different moments. Note that in the experimented game it is hard to decide which move leads to the best outcome. By looking at the predicted probability of the CNN we can detect the gameplay data where the network is not sure on which move is the best one. As an example we can set a threshold and if the network predicts two or more moves with probabilities that differ less than the threshold we can assume that the network is not able to decide which one is preferred and a different strategy to select the move might be appropriate in this case. In order to simulate different strategies of the same group of players we can select from the distribution of the predicted moves using different policies instead of the greedy one. This means that for example, if a move is scored 0.2 by the CNN, it will be selected with probability of 20%. The *softmax* function applied by the output layer guarantees that the predicted scores can be interpreted as probabilities and sum up to one. However, previous experiments showed that selecting moves in a non-greedy manner, only lead to worse performances. Future work in these directions is needed to validate or reject our hypothesis and deeper understand both the players and the game.

Categorize levels

We showed in Section 4.5 how developing a population of agents with different skills can be used to categorize levels absed on their strategy requirement. Having different metrics to evaluate a level can be useful for game designers to test the levels and to improve the quality of the game. We can hypothesize that levels that can be successfully solved also with random moves might be perceived less challenging by some players or that levels that require too much strategy might be an obstacle to some other players. A deeper analysis is needed to verify these hypothesis. In their research, Isaksen *et al.* [17] also estimated the dexterity required by each level. However in *Candy Crush Saga* the dexterity requirements are very low on all the levels. Nevertheless, a slightly different ability, called observability, can be modeled and estimated on each level. The idea is that players, differently from the developed agents, do not always detect all the available moves on the game board. We can add this type of error to the agents in different quantities and then estimate the observability required by each level. The error can be modeled by randomly reducing the number of available moves that the agent can select. The moves will not be selected randomly as it is for the strategy error, instead they will be selected greedily but within a subset of all the available moves. As an example, an observability of 50% means that the agent only detects half of the available moves on each game board. As a consequence if the agent predicts a move that is not between the ones that are detected, the move is discarded and the subsequent predicted move is selected. This process is repeated until the predicted move is in the subset of the detected moves. Moreover, instead of randomly detecting the moves, different heuristics can be used to decide which moves are detected and which ones are not. As an example, we can create an agent that more often detect matches of red candies than blue candies or an agent that more often detect matches with T-shape rather than L-shape. Ideally, we could learn also these peculiarities from data.

Another application might be to classify each level based on the agreement between the agents in selecting the moves on game boards obtained from that level. Computing the agents' agreement on several game states obtained from a specific level allows us to categorize levels based on how often different agents agree. The idea is that levels where all the agents select the same moves are less fun to play since the player strategy has a minor impact on the gameplay. Finally, we can also categorize levels based on the predicted probabilities of the network [100]. For example, if the network on a specific level consistently predicts a move with high probability it might indicate that the possible choices for the player are limited and as a consequence the level is less entertaining, while on the contrary if the network predicts many moves with similar probabilities it might indicate a more fun and challenging level.

Chapter 6

Summary and Conclusions

6.1 Summary

We experimented if simulating strategies of various cohorts of players would produce agents' SR that can be used to better predict the players' SR on new generated game content. We proposed two different approaches to model different player strategies by directly learning them from real player data without expert knowledge. We validated our approaches using the *Candy Crush Saga* game as a test bed. We tracked several player metrics and gameplay data consisting of the game board and the performed move for levels in the range [1, 2945]. In the first approach, called clustering players, we divided the tracked state-action pairs based on the performances of the players that have been tracked. Then, we trained different CNN-based agents on each generated player cluster. On the contrary, the second approach, called clustering simulated strategies, is based on the idea of letting the network discover the relationship between player features and player strategy. We trained a CNN-based agent that received as input three different player features in addition to the game board. Then, during prediction, by changing the input features we were able to predict different moves and as a consequence to simulate different players. We tested both of the proposed approaches against the state-of-the-art method that simulates an average strategy learnt from a random sample of real players. The goal was to understand if simulating different player strategies could generate agents' SR that can be used to better predict players' SR on new levels. All the simulations where performed by running 100 attempts for each level and aggregating the results to compute the agent's SR. Then, we used linear regression models to map from the agents' SR to the players' SR. In the clustering players approach we fitted a different linear model for each agent and we combined the predictions based on the percentage of players that each agent simulates. In the clustering simulated strategies approach we used a single linear model that directly combines the various agents' SR. Then, we used the developed linear models to predict players' SR on test levels. This corresponds to how the approaches could be used in practice. The training will be done on all the available levels and the models will be used to predict on newly created levels. Finally, we compared the prediction performances of our approaches with the state-of-the-art approach. We found that both the approaches were able to improve the MAE by 13% and the MSE by approximately 23% in the linear regression part. Adding the mean predictions to have an estimate of the players' SR also for levels where the agents failed reduced the improvements. However, also in this case, the proposed approaches performed better then the baseline approach. Furthermore, we found that not all the developed agents were strictly necessary to predict the players' SR and reducing the number of agent maintained the same prediction accuracy while saving computational resources. We discussed how the clustering players approach favors interpretability and it can provide game designers with multiple difficulty measures, perceived by different type of players instead of only the average perceived difficulty. On the contrary, we discussed how the clustering simulated strategy approach can work without any assumption on the relationship between player features and strategies and how it allows to use all the available data during training. Finally, we discussed limitations and future work that could possibly improve the developed approaches.

6.2 Conclusions

We have been able to model different player strategies while directly learning them from real player data. Incorporating characteristics of the players into the model allowed us to improve the prediction accuracy of the CNN-based agents as well as the players' SR prediction of the linear models. Furthermore, using the proposed approaches, we have been able to provide game designers with level difficulty measures perceived by different cohorts of players helping them to iteratively balance game content and create the best possible player experience. Additionally, we illustrated how the developed agents can be used to estimate strategy requirements of new levels and proposed different ways to further extend the work of this thesis.

To answer our research question, results suggested that player modeling can improve the level difficulty estimation of a CNN-based agent simulating human gameplay. As a consequence, player modeling techniques can be used to improve automatic playtesting. In addition, the proposed approaches are general and can be extended to work with other games and different player features or to estimate different metrics.

Bibliography

[1] D. Takahashi*, Game industry growing faster than expected, up 10.7% to $116 billion 2017.* [Online]. Available: https://venturebeat.com/2017/11/28/newzoo-game-industry-growing-faster-than-expected-up-10-7-to-116-billion-2017 (visited on 01/31/2018).

[2] C. Hwong*, Leveling up your mobile game: Using audience measurement data to boost user acquisition and engagement.* [Online]. Available: http://www.vertoanalytics.com/report-leveling-mobile-game-using-audience-measurement-data-boost-user-acquisition-engagement/ (visited on 01/31/2018).

[3] K. Alha, E. Koskinen, J. Paavilainen, and J. Hamari, *Critical Acclaim and Commercial Success in Mobile Free-to-Play Games*, 1 Edition. Digital Games Research Association and Society for the Advancement of the Science of Digital Games, Aug. 1, 2016.

[4] K. Alha, E. Koskinen, J. Paavilainen, J. Hamari, and J. Kinnunen, "Free-to-play games: Professionals' perspectives", *Proceedings of nordic DiGRA*, 2014.

[5] T. Fields, *Mobile & Social Game Design: Monetization Methods and Mechanics*, 2 Edition. CRC Press, Jan. 22, 2014.

[6] I. Civelek, Y. Liu, and S. Marston, "Pricing of virtual goods and designing game challenge level for free-to-play mobile games in the presence of copycat competitors", in *Proceedings of the 51st Hawaii International Conference on System Sciences*, 2018.

[7] J. Hamari, N. Hanner, and J. Koivisto, "Service quality explains why people use freemium services but not if they go premium: An empirical study in free-to-play games", *International Journal of Information Management*, pp. 1449–1459, Feb. 2017.

[8] J. Hamari, K. Alha, S. Järvelä, J. M. Kivikangas, J. Koivisto, and J. Paavilainen, “Why do players buy in-game content? an empirical study on concrete purchase motivations”, *Computers in Human Behavior*, pp. 538–546, Mar. 2017.

[9] J. T. Alexander, J. Sear, and A. Oikonomou, “An investigation of the effects of game difficulty on player enjoyment”, *Entertainment Computing*, pp. 53–62, Feb. 2013.

[10] R. Hunicke, “The case for dynamic difficulty adjustment in games”, in *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ACM, 2005, pp. 429–433.

[11] A. Zook, E. Fruchter, and M. O. Riedl, “Automatic playtesting for game parameter tuning via active learning.”, in *FDG*, 2014.

[12] F. d. M. Silva, S. Lee, J. Togelius, and A. Nealen, “AI as evaluator: Search driven playtesting of modern board games”, in *Games@AAAI*, 2017, p. 8.

[13] G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*, 1 Edition. Springer, 2017.

[14] A. Isaksen, D. Gopstein, and A. Nealen, “Exploring game space using survival analysis.”, in *FDG*, 2015.

[15] C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis, “Evolving personas for player decision modeling”, in *2014 IEEE Conference on Computational Intelligence and Games*, Aug. 2014, pp. 1–8.

[16] C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis, “Evolving models of player decision making: Personas versus clones”, *Entertainment Computing*, pp. 95–104, Jul. 1, 2016.

[17] A. Isaksen, D. Wallace, A. Finkelstein, and A. Nealen, “Simulating strategy and dexterity for puzzle games”, in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, Aug. 2017, pp. 142–149.

[18] A. Drachen, A. Canossa, and G. N. Yannakakis, “Player modeling using self-organization in tomb raider: Underworld”, in *2009 IEEE Symposium on Computational Intelligence and Games*, Sep. 2009, pp. 1–8.

[19] N. v. Hoorn, J. Togelius, D. Wierstra, and J. Schmidhuber, “Robust player imitation using multiobjective evolution”, in *2009 IEEE Congress on Evolutionary Computation*, May 2009, pp. 652–659.

[20] P. Eisen, “Simulating human game play for level difficulty estimation with convolutional neural networks”, Master’s Thesis, KTH Royal Institue of Technology, 2017.

[21] E. R. Poromaa, “Crushing candy crush: Predicting human success rate in a mobile game using monte-carlo tree search”, Master’s Thesis, KTH Royal Institue of Technology, 2017.

[22] J. Ortega, N. Shaker, J. Togelius, and G. N. Yannakakis, “Imitating human playing styles in super mario bros”, *Entertainment Computing*, pp. 93–104, Apr. 1, 2013.

[23] D. H. Hlynur, “Predicting expert moves in the game of othello using fully convolutional neural networks”, Master’s Thesis, KTH Royal Institue of Technology, 2017.

[24] S. Purmonen, “Predicting game level difficulty using deep neural networks”, Master’s Thesis, KTH Royal Institue of Technology, 2017.

[25] Z. Chen and D. Yi, “The game imitation: Deep supervised convolutional networks for quick video game AI”, *arXiv:1702.05663 [cs]*, Feb. 18, 2017.

[26] C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis, “Generative agents for player decision modeling in games.”, in *FDG*, Citeseer, 2014.

[27] M. Rosenberg, N. Confessore, and C. Cadwalladr, “How trump consultants exploited the facebook data of millions”, *The New York Times*, Apr. 2, 2018.

[28] D. J. Solove, *The Digital Person: Technology and Privacy in the Information Age.* NYU Press, Sep. 1, 2006, 283 pp.

[29] M. Hildebrandt and S. Gutwirth, Eds., *Profiling the European citizen: cross-disciplinary perspectives*, Springer, 2008, 373 pp.

[30] European Union*, General data protection regulation.* [Online]. Available: http://eugdpr.org/eugdpr.org.html (visited on 02/06/2018).

[31] T. I. Oren, “Ethics in modeling and simulation (simethics)”, in *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)*, 2005.

[32] R. C.-J. Lee Dave, “Hawking: AI could end human race”, *BBC News*, Dec. 2, 2014.

[33] T. Walsh, “What if... we create human-level artificial intelligence?”, *New Scientist*, pp. 32–34, Nov. 19, 2016.

[34] N. Bostrom and E. Yudkowsky, “The ethics of artificial intelligence”, *The Cambridge handbook of artificial intelligence*, pp. 316–334, 2014.

[35] R. Kumar, “Selecting a study design”, in *Research Methodology: A Step-by-Step Guide for Beginners*, 3 Edition, SAGE Publications, Inc, 2011, chpt. 8.

[36] J. W. Creswell and J. D. Creswell, *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*, 5 Edition. SAGE Publications, Inc, Jan. 2, 2018.

[37] T. Walsh, “Candy crush is NP-hard”, *arXiv:1403.1911 [cs]*, Mar. 7, 2014.

[38] C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis, “Generative agents for player decision modeling in games”, in *Poster Proceedings of the 9th Conference on the Foundations of Digital Games. (2014)*, 2014.

[39] A. M. Smith, C. Lewis, K. Hullett, G. Smith, and A. Sullivan, “An inclusive taxonomy of player modeling”, *University of California, Santa Cruz, Tech. Rep. UCSC-SOE-11-13*, 2011.

[40] G. N. Yannakakis, P. Spronck, D. Loiacono, and E. André, “Player modeling”, in *Artificial and Computational Intelligence in Games*, ser. Dagstuhl Follow-Ups, S. M. Lucas, M. Mateas, M. Preuss, P. Spronck, and J. Togelius, Eds., 1 Edition, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013, pp. 45–59.

[41] A. Tychsen and A. Canossa, “Defining personas in games using metrics”, in *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, ACM, 2008, pp. 73–80.

[42] R. Bartle, “Hearts, clubs, diamonds, spades: Players who suit MUDs”, *Journal of MUD research*, p. 19, 1996.

[43] D. Thue, V. Bulitko, M. Spetch, and E. Wasylishen, “Interactive storytelling: A player modelling approach.”, in *AIIDE*, 2007, pp. 43–48.

[44] C. Bateman and R. Boon, *21st Century Game Design (Game Development Series)*, 1 Edition. Charles River Media, Inc., 2005.

[45] A. Drachen, R. Sifa, C. Bauckhage, and C. Thurau, "Guns, swords and data: Clustering of player behavior in computer games in the wild", in *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, Sep. 2012, pp. 163–170.

[46] I. Sutskever and V. Nair, "Mimicking go experts with convolutional neural networks", in *Artificial Neural Networks - ICANN 2008*, V. Kůrková, R. Neruda, and J. Koutník, Eds., Edition 1, Springer Berlin Heidelberg, 2008, pp. 101–110.

[47] D. Stern, R. Herbrich, and T. Graepel, "Bayesian pattern ranking for move prediction in the game of go", ACM Press, 2006, pp. 873–880.

[48] C. J. Maddison, A. Huang, I. Sutskever, and D. Silver, "Move evaluation in go using deep convolutional neural networks", *arXiv:1412.6564 [cs]*, Dec. 19, 2014.

[49] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey", *IEEE Transactions on Computational Intelligence and AI in Games*, pp. 172–186, Sep. 2011.

[50] G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation", *IEEE Transactions on Affective Computing*, pp. 147–161, Jul. 2011.

[51] M. Mateas and A. Stern, "Façade: An experiment in building a fully-realized interactive drama", in *Game developers conference*, 2003.

[52] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbäck, and G. N. Yannakakis, "Multiobjective exploration of the StarCraft map space", in *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, Aug. 2010, pp. 265–272.

[53] L. Cardamone, G. N. Yannakakis, J. Togelius, and P. L. Lanzi, "Evolving interesting maps for a first person shooter", in *Applications of Evolutionary Computation*, C. Di Chio, S. Cagnoni, C. Cotta, *et al.*, Eds., Springer Berlin Heidelberg, 2011, pp. 63–72.

[54] P. L. Lanzi, D. Loiacono, and R. Stucchi, "Evolving maps for match balancing in first person shooters", IEEE, Aug. 2014, pp. 1–8.

[55] G. Andrade, G. Ramalho, H. Santana, and V. Corruble, "Challenge-sensitive action selection: An application to game balancing", in *IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, Sep. 2005, pp. 194–200.

[56] Z. Chen, Y. Sun, M. S. El-nasr, and T.-H. D. Nguyen, “Player skill decomposition in multiplayer online battle arenas”, *arXiv:1702.06253 [cs]*, Feb. 20, 2017.

[57] N. Pobiedina, J. Neidhardt, M. d. C. C. Moreno, L. Grad-Gyenge, and H. Werthner, “On successful team formation: Statistical analysis of a multiplayer online game”, in *2013 IEEE 15th Conference on Business Informatics*, Jul. 2013, pp. 55–62.

[58] N. Pobiedina, J. Neidhardt, M. d. C. Calatrava Moreno, and H. Werthner, “Ranking factors of team success”, in *Proceedings of the 22nd International Conference on World Wide Web*, ACM, 2013, pp. 1185–1194.

[59] P. Yang, B. E. Harrison, and D. L. Roberts, “Identifying patterns in combat that are predictive of success in MOBA games.”, in *FDG*, 2014.

[60] T. Mahlmann, A. Drachen, J. Togelius, A. Canossa, and G. N. Yannakakis, “Predicting player behavior in tomb raider: Underworld”, in *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, Aug. 2010, pp. 178–185.

[61] A. Lobel, M. Gotsis, E. Reynolds, M. Annetta, R. C. Engels, and I. Granic, “Designing and utilizing biofeedback games for emotion regulation: The case of nevermind”, ACM Press, 2016, pp. 1945–1951.

[62] M. Pirovano, R. Mainetti, G. Baud-Bovy, P. L. Lanzi, and N. A. Borghese, “Self-adaptive games for rehabilitation at home”, IEEE, Sep. 2012, pp. 179–186.

[63] B. Gorman, C. Thurau, C. Bauckhage, and M. Humphrys, “Bayesian imitation of human behavior in interactive computer games”, in *18th International Conference on Pattern Recognition (ICPR’06)*, 2006, pp. 1244–1247.

[64] R. R. Wehbe, E. Lank, and L. E. Nacke, “Left them 4 dead: Perception of humans versus non-player character teammates in cooperative gameplay”, in *Proceedings of the 2017 Conference on Designing Interactive Systems*, ACM, 2017, pp. 403–415.

[65] P. Hingston, “A turing test for computer game bots”, *IEEE Transactions on Computational Intelligence and AI in Games*, pp. 169–186, Sep. 2009.

[66] J. Schrum, I. V. Karpov, and R. Miikkulainen, “UT2: Human-like behavior via neuroevolution of combat behavior and replay of human traces”, in *2011*

IEEE Conference on Computational Intelligence and Games (CIG'11), Aug. 2011, pp. 329–336.

[67] M. Polceanu, "MirrorBot: Using human-inspired mirroring behavior to pass a turing test", in *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, Aug. 2013, pp. 1–8.

[68] C. E. Shannon, "XXII. programming a computer for playing chess", *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, pp. 256–275, Mar. 1950.

[69] A. M. Turing, "Digital computers applied to games", *Faster than Thought*, 1953.

[70] A. L. Samuel, "Some studies in machine learning using the game of checkers", *IBM Journal of Research and Development*, pp. 210–229, Jul. 1959.

[71] J. Schaeffer, R. Lake, P. Lu, and M. Bryant, "CHINOOK the world man-machine checkers champion", *AI Magazine*, p. 21, Mar. 15, 1996.

[72] G. Tesauro, "Temporal difference learning and TD-gammon", *Communications of the ACM*, pp. 58–68, 1995.

[73] M. Buro, "The othello match of the year: Takeshi murakami vs. logistello", *ICGA Journal*, pp. 189–193, Jan. 1, 1997.

[74] M. Campbell, H. A. Joseph, and F. Hsu, "Deep blue", *Artificial Intelligence*, pp. 57–83, Jan. 1, 2002.

[75] S. Ontañón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game AI research and competition in StarCraft", *IEEE Transactions on Computational Intelligence and AI in Games*, pp. 293–311, Dec. 2013.

[76] G. Synnaeve, N. Nardelli, A. Auvolat, S. Chintala, T. Lacroix, Z. Lin, F. Richoux, and N. Usunier, "TorchCraft: A library for machine learning research on real-time strategy games", *arXiv:1611.00625 [cs]*, Nov. 1, 2016.

[77] N. Usunier, G. Synnaeve, Z. Lin, and S. Chintala, "Episodic exploration for deep deterministic policies: An application to StarCraft micromanagement tasks", *arXiv:1609.02993 [cs]*, Sep. 9, 2016.

[78] P. Peng, Y. Wen, Y. Yang, Q. Yuan, Z. Tang, H. Long, and J. Wang, "Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play StarCraft combat games", *arXiv:1703.10069 [cs]*, Mar. 29, 2017.

[79] D. Takahashi*, Candy crush saga: 2.73 billion downloads in five years and still counting.* [Online]. Available: https://venturebeat.com/2017/11/17/candy-crush-saga-2-73-billion-downloads-in-five-years-and-still-counting/ (visited on 02/06/2018).

[80] D. Silver, A. Huang, C. J. Maddison, *et al.*, "Mastering the game of go with deep neural networks and tree search", *Nature*, pp. 484–489, Jan. 2016.

[81] D. Silver, J. Schrittwieser, K. Simonyan, *et al.*, "Mastering the game of go without human knowledge", *Nature*, pp. 354–371, Oct. 18, 2017.

[82] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to Linear Regression Analysis.* John Wiley & Sons, Apr. 9, 2012, 680 pp.

[83] D. Stutz*,* Latex-resources: Collection of LaTeX resources and examples. [Online]. Available: https://github.com/davidstutz/latex-resources (visited on 04/22/2018).

[84] M. Lin, Q. Chen, and S. Yan, "Network in network", *arXiv:1312.4400 [cs]*, Dec. 16, 2013.

[85] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition", *Neural Computation*, pp. 541–551, Dec. 1, 1989.

[86] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position", *Biological Cybernetics*, pp. 193–202, Apr. 1, 1980.

[87] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[88] O. Russakovsky, J. Deng, H. Su, *et al.*, "ImageNet large scale visual recognition challenge", *International Journal of Computer Vision*, pp. 211–252, Dec. 2015.

[89] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.

[90] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[91] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, 1 Edition. The MIT Press, Nov. 18, 2016.

[92] B. S. Everitt, S. Landau, M. Leese, and D. Stahl, "Hierarchical clustering", in *Cluster Analysis, 5th Edition*, ser. Wiley Series in Probability and Statistics, 5 Edition, John Wiley & Sons, Jan. 7, 2011, pp. 71–110.

[93] S. Lloyd, "Least squares quantization in PCM", *IEEE Transactions on Information Theory*, pp. 129–137, Mar. 1982.

[94] S. T. Wierzchoń and M. A. Kłopotek, "Cluster quality versus choice of parameters", in *Modern Algorithms of Cluster Analysis*. Cham: Springer International Publishing, 2018, pp. 163–180.

[95] J. Wu, *Advances in K-means Clustering: A Data Mining Thinking*, 1 Edition, ser. Springer Theses. Springer-Verlag, 2012.

[96] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", *arXiv:1412.6980 [cs]*, Dec. 22, 2014.

[97] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)", *arXiv:1511.07289 [cs]*, Nov. 23, 2015.

[98] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient", in *Noise Reduction in Speech Processing*, ser. Springer Topics in Signal Processing, Springer, Berlin, Heidelberg, 2009, pp. 1–4.

[99] J. M. Chambers, W. S. Cleveland, B. Kleiner, and P. A. Tukey, "Assessing distributional assumptions about data", in *Graphical Methods for Data Analysis*, 1st ed., Chapman and Hall/Cole Publishing Company, Feb. 1, 1983, chpt. 6.

[100] M. Costa, “Probabilistic interpretation of feedforward network outputs, with relationship to statistical prediction of ordinal quantities”, *International Journal of Neural Systems*, pp. 627–637, Nov. 1996.

Appendix A

Appendix

A.1 List of Boosters

In the following table there are all the various boosters with their description, effect, usage condition and how they look like in the game. Each row represents one booster.

Table A.1. Full list and description of boosters

Name	Image	Type	Levels	Effect
Lollipop Hammer		In-level	All levels	Smash a candy (or a blockers) except chocolate spawner, ingredients, toffee tornado or sugar chest and destroy it.
Extra Moves +5		In-level, consolation	All except timed levels	Adds five additional extra moves.
Extra Moves +3		Pre-level	All except timed levels	Adds three additional extra moves.
Jelly Fish		Pre-level	Jelly levels	Spawn 3 jelly fish at random on the gameboard.
Colour Bomb		Pre-level	All levels	Start with one colour bomb on the gameboard.
Coconut Wheel		Pre-level	Ingredients levels	Spawn a coconut wheel on the gameboard.
Free Switch		In-level	All levels	Allows the player to switch 2 candies even if they do not match.
Striped and Wrapped		Pre-level	All levels	Start the game with a wrapped and a striped candy on the gameboard.
Bomb Cooler		In-level	All levels	Adds five extra moves to Candy Bombs. Available only if there are Candy Bombs on the gameboard.
Lucky Candy		Pre-level	Order levels	Spawn a lucky candy on the gameboard.
UFO		Pre-level	All levels	Spawn a UFO on the gameboard.
Striped Brush		In-level	All levels	Allows the player to convert any regular candy into a striped candy and to choose the directions of the stripes.
Party Popper		In-level	All levels	Clears the gameboard and adds special candies.

A.2 Supplement Information Regarding CNN Inputs

In the following table there are all the input features of the CNN. Each row represents one feature layer.

Table A.2. Full list of input features (1/2)

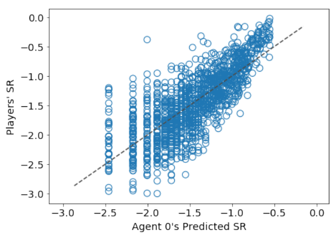
Layers
REGULAR_CANDY
PEPPER_CANDY
MYSTERY_CANDY
CHAMELEON_CANDY
CANDY_COLOR
CANDY_COLOR_RED
CANDY_COLOR_YELLOW
CANDY_COLOR_BLUE
CANDY_COLOR_GREEN
CANDY_COLOR_ORANGE
CANDY_COLOR_PURPLE
FISH
CANDY_STRIPED_LINE
CANDY_STRIPED_COLUMN
CANDY_WRAPPED
LUCKY_CANDY
VOID
EMPTY
LIGHT_1
LIGHT_2
CANDY_CANNON
STATIC_BLOCKER
FROSTING
LIQUORICE_LOCK
FUDGE
INGREDIENT_COLLECTOR
PORTAL
PORTAL_ENTER_POINT
PORTAL_EXIT_POINT
PORTAL_VISIBLE
PORTAL_VISIBLE_ENTER_POINT
PORTAL_VISIBLE_EXIT_POINT
LICORICE_SQUARE
MULTI_FROSTING_1
MULTI_FROSTING_2
MULTI_FROSTING_3
MULTI_FROSTING_4
MULTI_FROSTING_5
CHOCOLATE_SPAWNER
MARMELADE_LOCK
DIVINE_DROP
SUGAR_DROP
CANDY_CANNON_AMMO_CANDY
CANDY_CANNON_AMMO_INGREDIENT
CANDY_CANNON_AMMO_LIQUORICE_SQUARE
CANDY_CANNON_AMMO_PEPPER
CANDY_CANNON_AMMO_MULOCK_CANDY
CANDY_CANNON_AMMO_MYSTERY_CANDY
CAKE_BOMB
JELLY_FROG
MULOCK_1
MULOCK_2
MULOCK_3
MULOCK_4
MULOCK_5

Table A.3. Full list of input features (2/2)

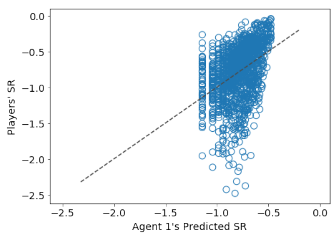
Layers
COCONUT_WHEEL
INGREDIENT
EXTRA_TIME
MULOCK_KEY
CHOCOLATE_FROG
POPCORN
UFO
EVIL_SPAWNER
FROGGER_EXIT
JELLY_COLOR_GREEN
JELLY_COLOR_RED
BOBBER
CANDY_CANNON_AMMO_CHAMELEON
CANDY_CANNON_AMMO_LUCKY
CANDY_CANNON_AMMO_TIME
CONVEYOR_BELT
CONVEYOR_BELT_UP
CONVEYOR_BELT_RIGHT
CONVEYOR_BELT_DOWN
CONVEYOR_BELT_LEFT
CONVEYOR_BELT_PORTAL
CONVEYOR_BELT_PORTAL_NONE
CONVEYOR_BELT_PORTAL_RED
CONVEYOR_BELT_PORTAL_BLUE
CONVEYOR_BELT_PORTAL_GREEN
ORDER_CANDY_COLOR_RED
ORDER_CANDY_COLOR_BLUE
ORDER_CANDY_COLOR_YELLOW
ORDER_CANDY_COLOR_ORANGE
ORDER_CANDY_COLOR_PURPLE
ORDER_CANDY_COLOR_GREEN
ORDER_CANDY_WRAPPED
ORDER_CANDY_STRIPED
ORDER_CANDY_COLOR
ORDER_CANDY_FUDGE
ORDER_CANDY_FROSTING
ORDER_CANDY_POPCORN
ORDER_LIQUORICE_SQUARE
ORDER_STRIPED_STRIPED
ORDER_STRIPED_WRAPPED
ORDER_STRIPED_CANDY_COLOR
ORDER_WRAPPED_WRAPPED
ORDER_CANDY_COLOR_CANDY_COLOR
ORDER_CANDY_COLOR_WRAPPED
BIAS_LAYER
MOVES_LEFT

A.3 Supplement Plots for the Clustering Players Approach

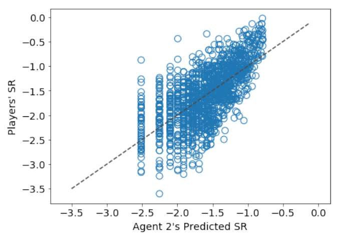
A.3.1 Assumption Validation on Training Levels Plots



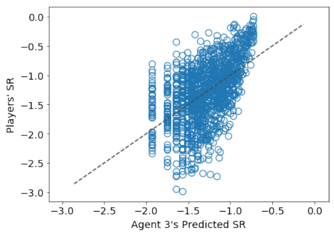
(a) Agent 0



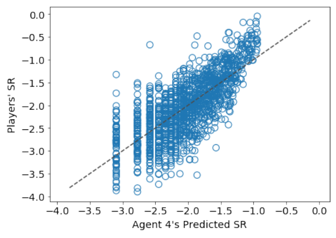
(b) Agent 1



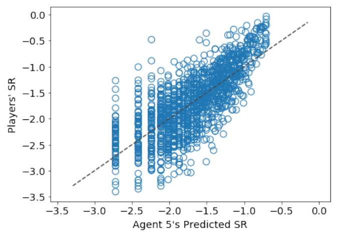
(c) Agent 2



(d) Agent 3

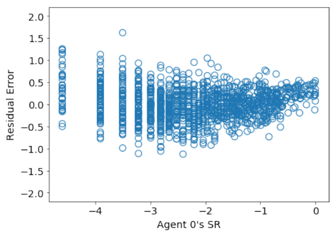


(e) Agent 4

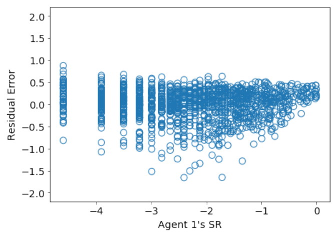


(f) Agent 5

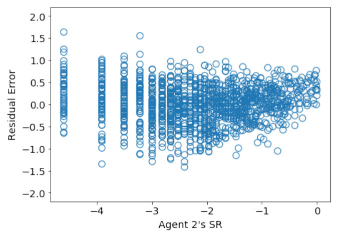
Figure A.1. Players' SR against predicted SR



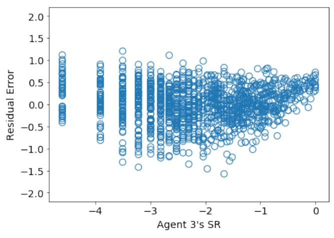
(a) Agent 0



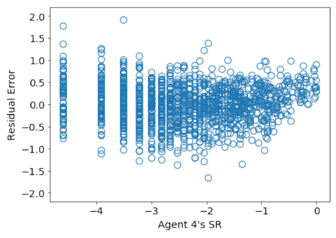
(b) Agent 1



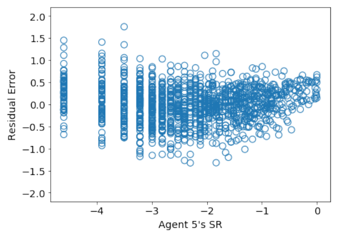
(c) Agent 2



(d) Agent 3

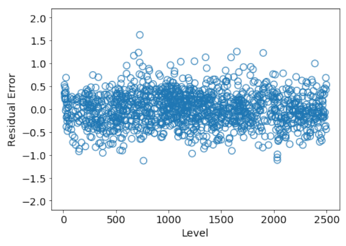


(e) Agent 4

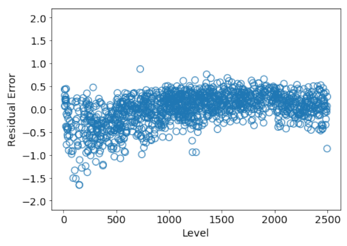


(f) Agent 5

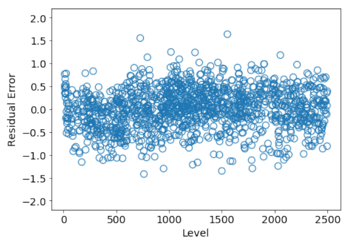
Figure A.2. Agent's SR against residuals



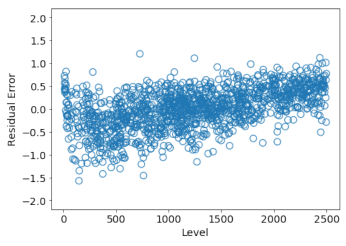
(a) Agent 0



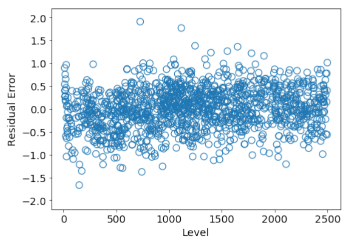
(b) Agent 1



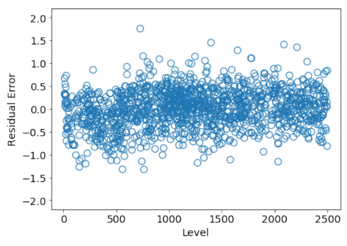
(c) Agent 2



(d) Agent 3

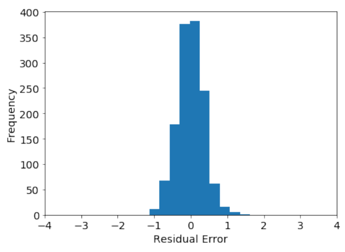


(e) Agent 4

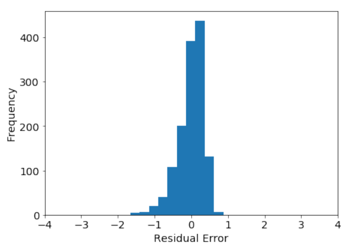


(f) Agent 5

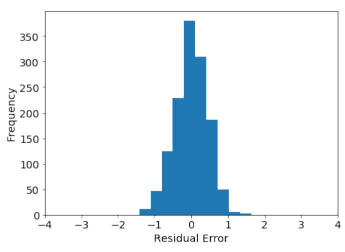
Figure A.3. Levels against residuals



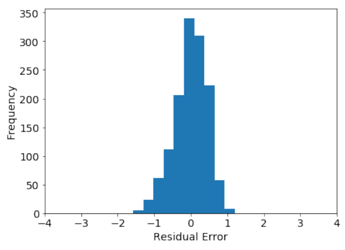
(a) Agent 0



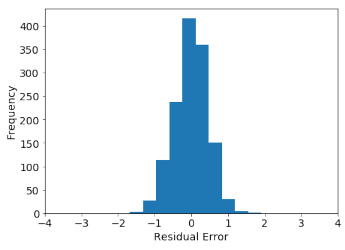
(b) Agent 1



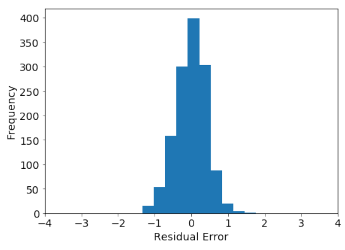
(c) Agent 2



(d) Agent 3

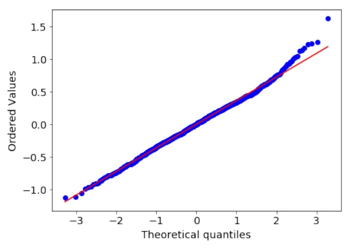


(e) Agent 4

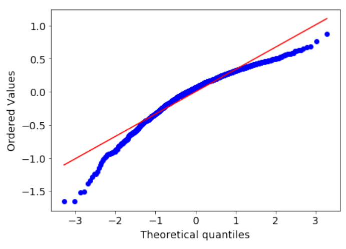


(f) Agent 5

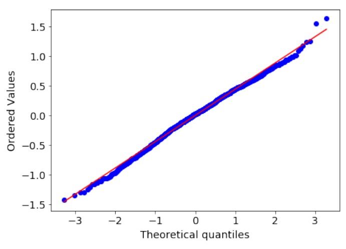
Figure A.4. Residual frequency



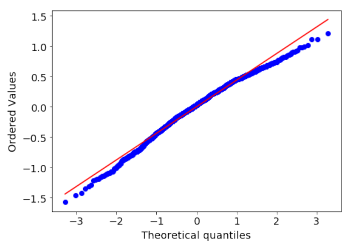
(a) Agent 0



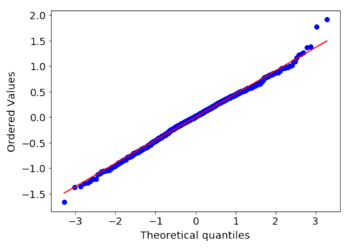
(b) Agent 1



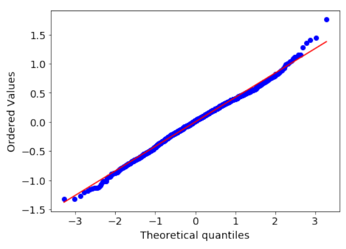
(c) Agent 2



(d) Agent 3

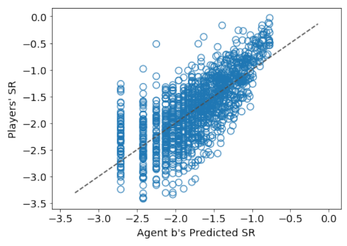


(e) Agent 4

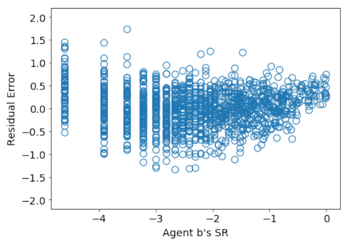


(f) Agent 5

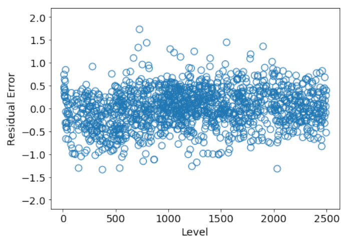
Figure A.5. Normal probability plots of residuals



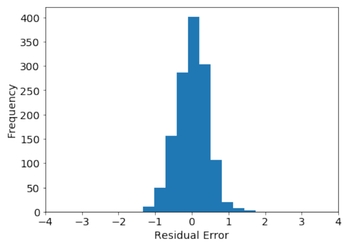
(a) Players' vs predicted SR



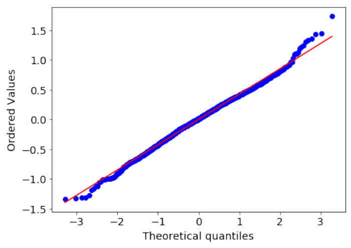
(b) Baseline's SR vs residuals



(c) Levels vs residuals



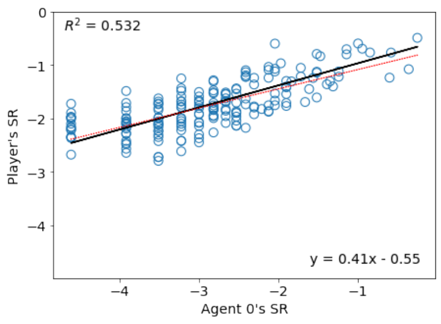
(d) Residual frequency



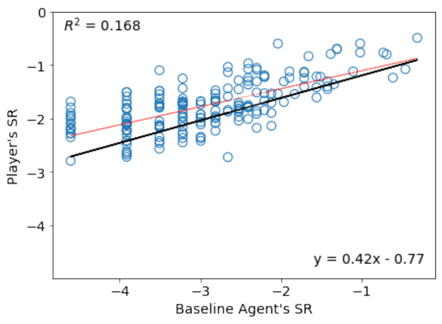
(e) Residual normal prob. plot

Figure A.6. Assumptions validation for the baseline

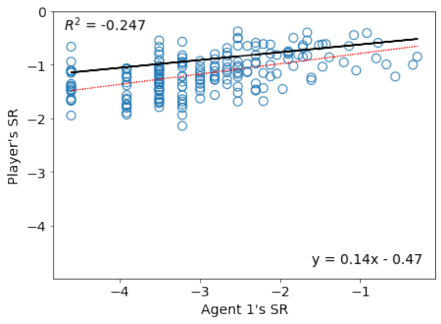
A.3.2 Agents' and Baseline's Predictions in the Player Clusters



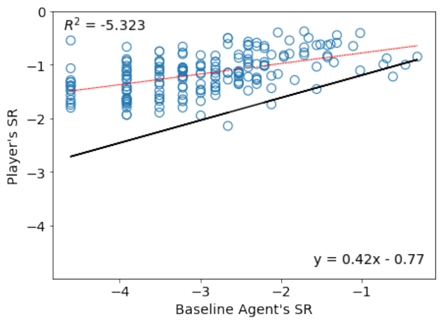
(a) Agent 0's predictions



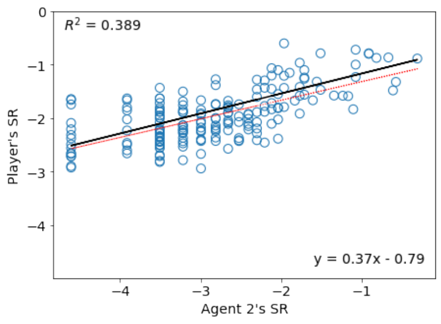
(b) Baseline's predictions for cluster 0



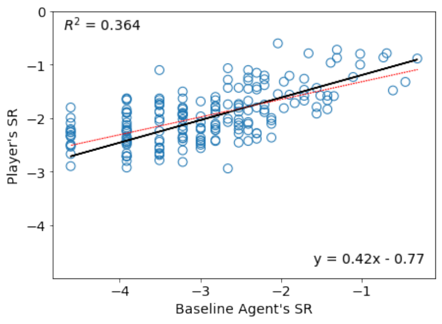
(c) Agent 1's predictions



(d) Baseline's predictions for cluster 1

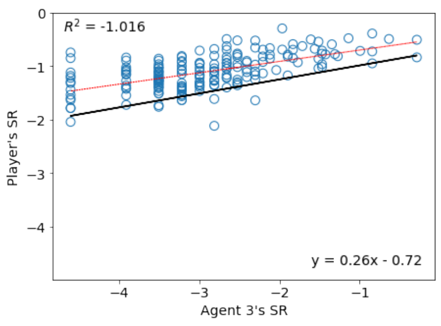


(e) Agent 2's predictions

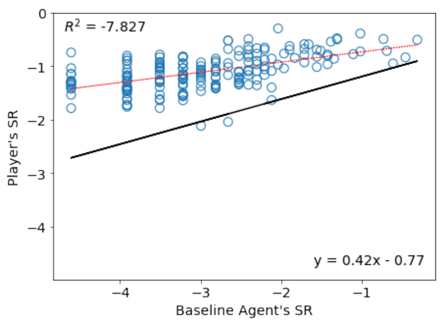


(f) Baseline's predictions for cluster 2

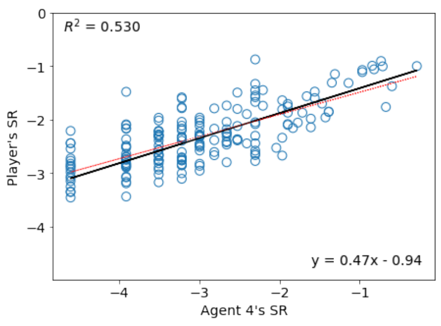
Figure A.7. Agents' vs baseline's predictions on each player cluster (1/2)



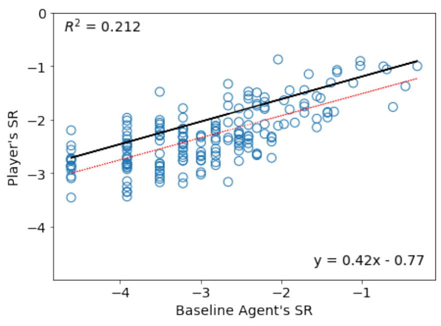
(a) Agent 3's predictions



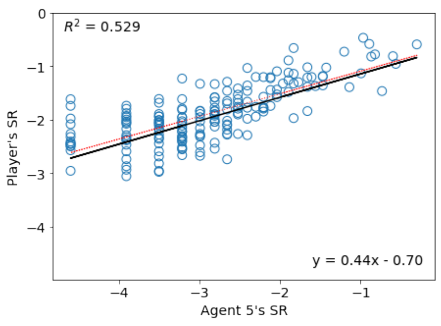
(b) Baseline's predictions for cluster 3



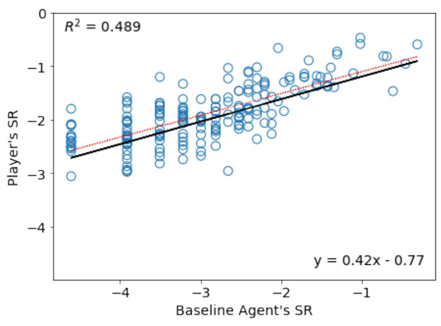
(c) Agent 4's predictions



(d) Baseline's predictions for cluster 4



(e) Agent 5's predictions

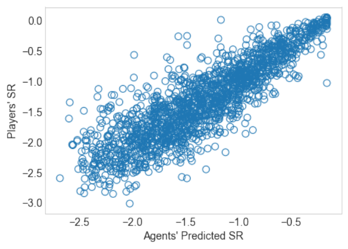


(f) Baseline's predictions for cluster 5

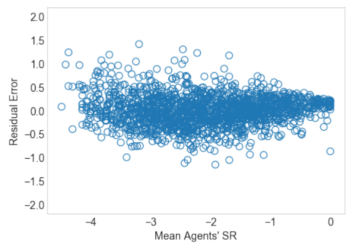
Figure A.8. Agents' vs baseline's predictions on each player cluster (2/2)

A.4 Supplement Plots for the Clustering Simulated Strategies Approach

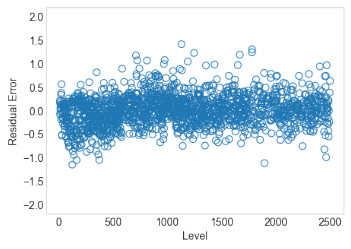
A.4.1 Assumptions Validation on Training Levels Plots



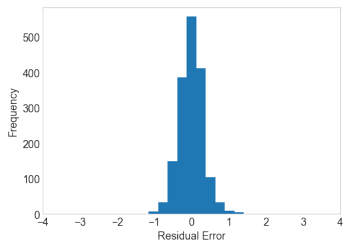
(a) Agents' vs predicted SR



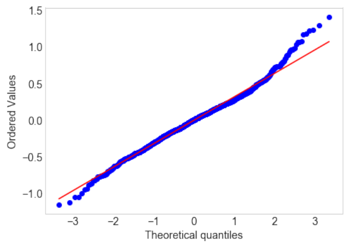
(b) Avg agents' SR vs residuals



(c) Levels vs residuals

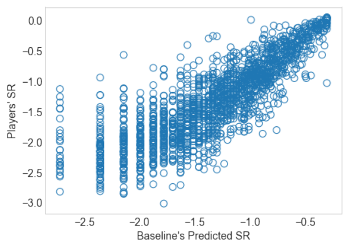


(d) Residual frequency

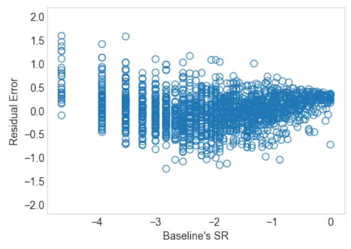


(e) Residual normal prob. plot

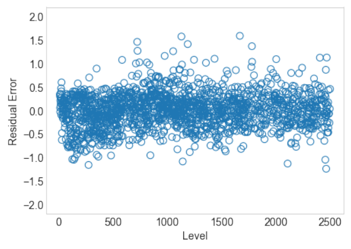
Figure A.9. Assumptions validation for the agents' linear model



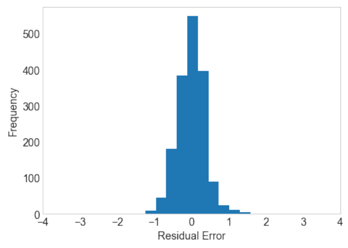
(a) Baseline's vs predicted SR



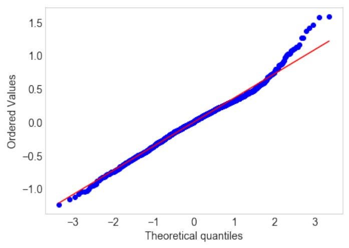
(b) Baseline's SR vs residuals



(c) Levels vs residuals



(d) Residual frequency



(e) Residuals normal prob. plot

Figure A.10. Assumptions validation for the baseline's linear model

Declaration

I hereby certify that I have written this thesis independently and I have only used the specified sources and resources indicated in the bibliography.

Milan, June 28, 2018

. . . .

Davide Anghileri