UNIVERSITÄT OSNABRÜCK

COGNITIVE SCIENCE

Challenge yourself:

A cognitive agent's deep reinforcement learning framework for adaptive, intelligent and interesting competition in video game opponents

M.Sc. Thesis in Cognitive Science at University of Osnabrück
Supervisor: Prof. Dr. Gordon Pipa
2nd Supervisor: Dr. Tarek R. Besold

Osnabrück, Germany
Johannes Pfau, 2016
Johannes.Pfau@gmx.net

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Johannes Pfau

Abstract

The following thesis will introduce the state of the art of Artificial Intelligence in video games, both from the perspective of academic research as well as from the industrial companies themselves, supported by an informative interview. Furthermore, having in mind that the implementation of intelligent opponents is hardly something to add on top, it depicts the development of these agents during the design process in the example of the trading card game *Korona: Tinker's Curse*. In a bottom-up fashion, features are introduced and complexity is increasing, while different learning approaches are benchmarked throughout the process. In the end, a deep reinforcement learning architecture is applied to the final version of the game and refined until it meets the advertised performance: a learning agent adapting to the player's choices and preferences with increasing challenge that reaches the performance of a highly complex, sophisticated heuristic finite state machine, which are the common tool for decision making otherwise. Finally, this implementation is wrapped up in a plugin, which brings an easy-to-use interface to developers interested in this technology. Further examples demonstrate the modularity and adaptability of the framework to other games and genres.

Key words

Artificial Intelligence in video games - Machine Learning
Reinforcement Learning - Deep Learning - Neural Networks

Digital attachment

This thesis is delivered with a CD containing all material used for it, i.e. a digital PDF version of itself, high resolution versions of all figures and graphs, Matlab scripts and the final C implementation. Note that some tables and images may be better perceptible digitally.

Acknowledgements

I would like to thank all the people that supported me in my work, motivated, inspired or accompanied me.

Thanks to my family, my girlfriend, my friends, my colleagues and fellow students.

Many thanks to the developers and companies that helped me with their inside knowledge of Artificial Intelligence usage in modern video games, thanks to

Daniel Doan from *Black Shell Media*,
Alen Ladavac and Damjan Mranuvac from *Croteam*,
Sandy Brand from *Crytek*,
Dave LeCompte from *Harebrained Schemes*,
David Szymczyk and Mikey Dowling from *Obsidian Entertainment*,
Carl Carenvall and Malin Enlund from *Paradox Development Studio*,
and all the anonymous submitters.

Thanks again for time and advice in setting up the whole thesis to my supervisors Gordon Pipa and Tarek Besold.

And special thanks for the everlasting physical support to my cat Misery.

Thank you.

Table of Contents

Chapter 1

The idea of Artificial Intelligence in video games

1.1 Introduction

Over the last 50 years, video games undoubtedly became a huge leisure time and pleasure factor in modern society, spanning nearly 2 billion consumers with a continually growing yearly revenue of almost $100 billion worldwide [52]. The community is no longer restricted to a specific subset of the society, but includes people from all age ranges, continents and backgrounds, due to the nowadays common distribution of computers, consoles, smartphones and tablets.

This opens the door for a lot of applications and fields of study, including Artificial Intelligence, which profits from many features the game industry can offer: Not only is the market that booming that big representatives could easily afford scientific research, but also is the community that big and spends that much time (on average over 6 hours a week [2]), that massive amounts of data can be collected, evaluated and learned. Above that, most of today's popular games operate on powerful machines anyway, since the graphical requirements often impose those - and/or require an active internet connection - so that enough computational power is provided, or the possibility for AI cloud computation [4] is opened up.

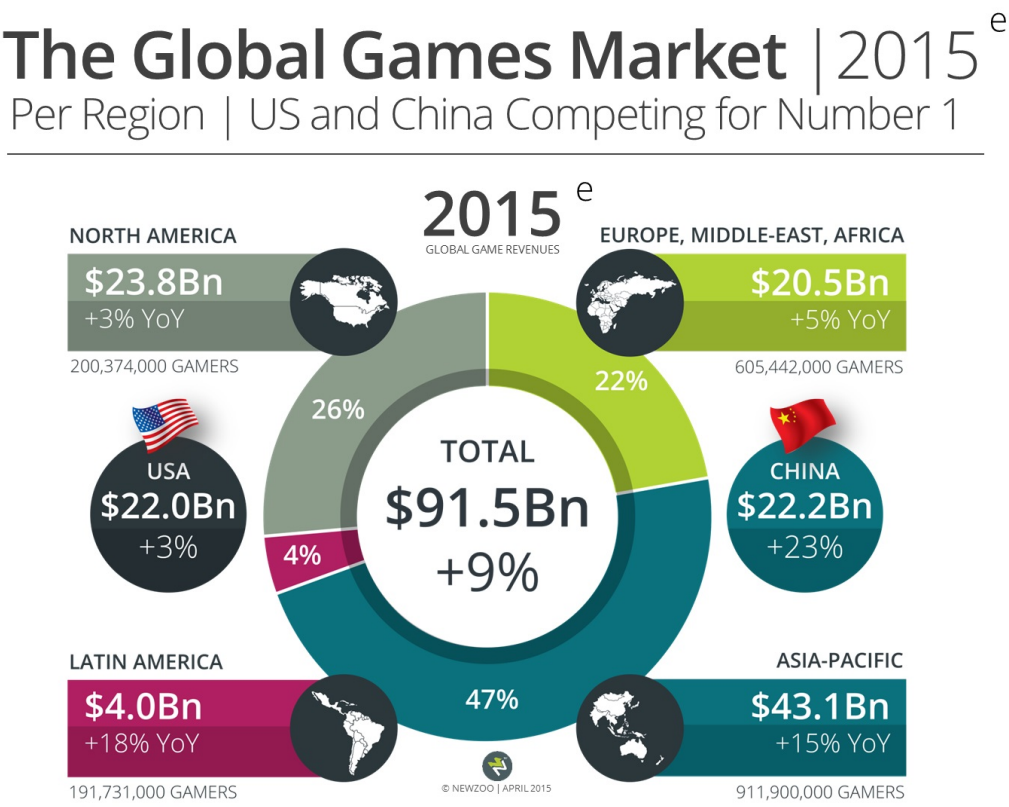


Figure 1.1: Approximated revenue of the worldwide video game industry, 2015 [52]

But not only the science of AI benefits from this giant experimental space, but also the other way around. In the most cases, video games somehow simulate parts of the real world, e.g. surviving an encounter in FPS games, steering a car in racing games or solving quests and puzzles in RPGs. These experiences are not only improved by refining the graphical quality of the environment, but also by the ("neither too hard nor too easy"[21]) challenge that the game itself or opponents present and the authenticity that these convey. Despite the vast number of approaches and potential applications that academic AI offers (mentioned in the next chapter), most video game companies seem to stick to "simple rule-based finite and fuzzy-state machines for nearly all their AI needs"[59]. For that reason, this thesis examines the possibilities of improvement that AI offers to the industry, interviews the actual developers about their conception of AI usage, advantages and disadvantages and finally describes the concept, design and implementation of a machine learning tool for more adaptive, challenging and interesting opponents in video games.

1.2 State of AI in modern video games

1.2.1 Academic approaches

Studying academic AI in video games is no new phenomenon, in fact there are many approaches on how to integrate it, to really improve the quality of the particular game. However, almost every published paper utilizes existing games to present the idea, without further notification (and not at all integration) from the side of the company - or sets up own, small game prototypes especially for the purpose of demonstration, which are far from being a real, acknowledged video game. For this reason, the proposed tool of this thesis will be directly applied to a game that is currently in development, provides a convenient benchmark environment and benefits greatly from the integration, before the tool is released and made publicly available as a plugin for game developers.

To grasp the scope of the history of academic AI in video games and furthermore set up a contrast to the industrial approaches in the next chapter, some of the most innovative and successful studies and implementations are listed in the following.

Many disciplines in the spectrum of academic AI have been researched, be it in the classical approach of planning [35] and search [30], decision trees [38], coordination graphs [27], dynamic scripting [48][24], self-organizing maps [18], pathfinding [62], context representation [3], cognitive architectures [12][55], reinforcement learning [26] [10] or artificial neural networks [23][28][29][15][34][14][5][32][56], where I draw the most inspiration from. Again, all of these approaches utilize existing games (traditional, like Chess [8], Go [58], 2x2 games [54][33], backgammon [49], Fox and Geese [32] or modern, like Quake II [5], Quake III [56], Halo 3 [51], Descent 3 [55], Unreal Tournament [12][25], Counter Strike [57], StarCraft [11][17], Neverwinter Nights [48], Baldurs Gate [47], Forza Motorsport [51], PacMan [19][34], Pong [18], Atari Games [14]) or come up with specifically designed prototypes (Virus [6], NERO [23], Build & Build [28], Cellz [29], Cutlass [39] and numerous others). Note that these studies do not restrict themselves to a specific genre, but incorporate traditional, First Person Shooters, Real-Time Strategy Games, Role Playing Games, Racing Games and Arcade. The raise of interestingness that academic AI offers has been studied several times [20][21][19], but, as Spronck et al. state, the academic kind of AI approaches have "*[...] a wealth of techniques that are waiting for implementation in actual commercial games, but [...] the games' publishers are reluctant to incorporate these techniques since they have no definitive opinion on the successes of a program that is outclassing human beings in strength and creativity , and [...] game AI has an entertainment factor that is*

too multifaceted to grasp in reasonable time" [48, p. 15]. To get an unbiased opinion on this matter, the following chapter concerns with the same topic from the viewpoint of the actual industrial video game developers.
Above that, many of the games that were purposely designed in this process have the questionable connotation that they could beg the question of the desired learning/game playing approach. Thus, in this thesis, I will focus on a game that is going to be published anyhow and adapt the AI to it. The immediate problem that usually arises with these adapted AI approaches is, both from the scientific viewpoint of generalization and due to the fact that the industry would want to re-use AI software once it is established, the AI shouldn't have to be too much adapted before it yields satisfying results. Thus, I will design a framework that operates on the same interface to the game that also a human player would have, where the input space is well observable and the relation to the actions will be learned in the same fashion as a human player would learn it. This ensures not only that we have a plug-in AI framework that suits many cases and applications, but also gives a cognitive notion on the artificial agent, since it is strictly focused on observation, state estimation and decision making or planning, all parts of a cognitive process.

1.2.2 Industrial approaches

The problem of judging video games about their implementation of AI algorithms is that the internal research and techniques are rarely published. The average gamer seems to be not interested in the internal, algorithmic structure of the game and the particular scripts, e.g. for opponent or agent behavior, are kept secret because of patent and privacy reasons. Above that, it is hard for the naked eye to tell whether simulated behavior stems from complex, intelligent reasoners or simple, but fuzzy decision trees. Thus, the only way to get valid inside knowledge about AI usage in industrial video game companies is to contact the particular developers of the various enterprises directly. This objective proved to be a challenging task, since the contact details of these developers are not publicly available, they often don't have the temporal resources to talk about these topics and again, certain companies conceal their algorithmic structures because of intellectual property reasons. Nevertheless, I tried to contact 105 of the most important and successful representatives of the industry (see: Documentation of industrial video game company correspondence, 9 responding contacts) several times, which led to satisfying results.

In order to get a comparable statement between companies or developers, they were asked to fill out a survey that followed the individual conversations. The access to the questionnaire was made available online [41], where the respondents filled out basic descriptive values, the information, which disciplines of AI they already use in their video games, the explicit algorithms, techniques or methods that were applied, personal statements and opinions about the value of AI in video games and whether they want to submit this survey anonymously (see: AI and Machine Learning in state-of-the-art video games: A Digital Interview).

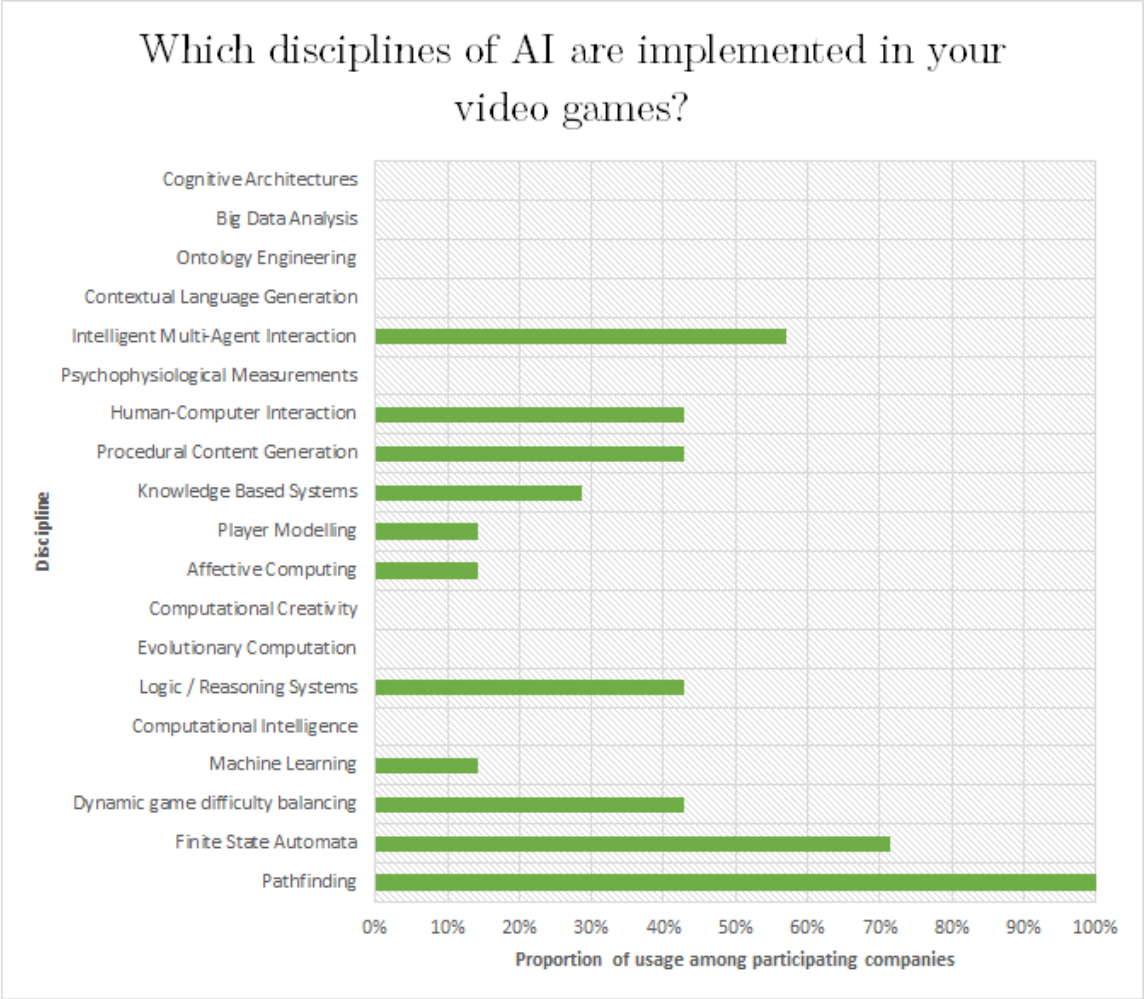


Figure 1.2: Proportional results of AI disciplines usage

Studying the results of the survey, the first thing that comes to mind is the successful integration of pathfinding in more or less every modern video game (Figure 1.2). Algorithms like A* [60] are cheap, reliable and compelling, which are all necessary conditions for consumer-used environments. Above that, compared to many of the other fields of AI, pathfinding is absolutely essential for video game opponents, to prevent totally idiosyncratic behavior, which led to a very early establishment in the industry. Another often used technique is shown in Finite State Automata, which are pretty robust and observable, but do not possess any higher level capability of reasoning. Developers state that they use

them for *"Movement state machines, etc."*, *"Character action sequences and combat"* or *"a lot of tasks not considered AI, like managing states of User Interface widgets"*, which shows that they fulfill predictable tasks but are far from academic AI. Dynamic game difficulty balancing is rather roughly applied with heuristics like *"[opponents] will start to miss more after managing to hit the player too rapidly"*, which holds also for the reasoning systems in modern games, which are mostly frugal decision makers about movement (*"e. g. to find out what a good position to shoot from will be, considering things like line-of-fire, distance to target, minimal distance from current position, closeness to allies, etc"*, *"Most of our AI is still reactive, but we have systems that 'sample' positions in the world for things like: get good attack posion, cover spot, etc"*. Knowledge bases for NPCs are elementary, but common, by stating things they know versus things they dont know, for example in *"computer player's knowledge of the game state (where other units are on the map)"*. Procedural Content Generation has found it's place in the game industry, not least because of games that are completely centered around it, like Minecraft, Spore, No Man's Sky, etc. - but also in regular games that are not completely focused on PCG, mostly for *"Worldbuilding"* or *"[generating] in-game content, like making trees at design time"*. Finally, Multi-Agent interaction is a discipline that can improve game quality in a thoroughly manner, which is why many companies try to come up with good solutions, e.g. *"NPCs can decide to perform a complex attack together"*, *"One AI charges a player, while the team members give covering fire"*, albeit drawing on FSA for these decisions. The reasons for this are straightforward and shared amongst many companies: *"So far, our AI systems are mostly reactive and driven by behavior trees that get signals from events that happen in the world.The reason for this is that we need to model explicit rules in their behaviors to make the AI readable and 'fun' for the player.Also, we need to do this using limiting CPU bandwidth and in a way that these systems are debuggable".*

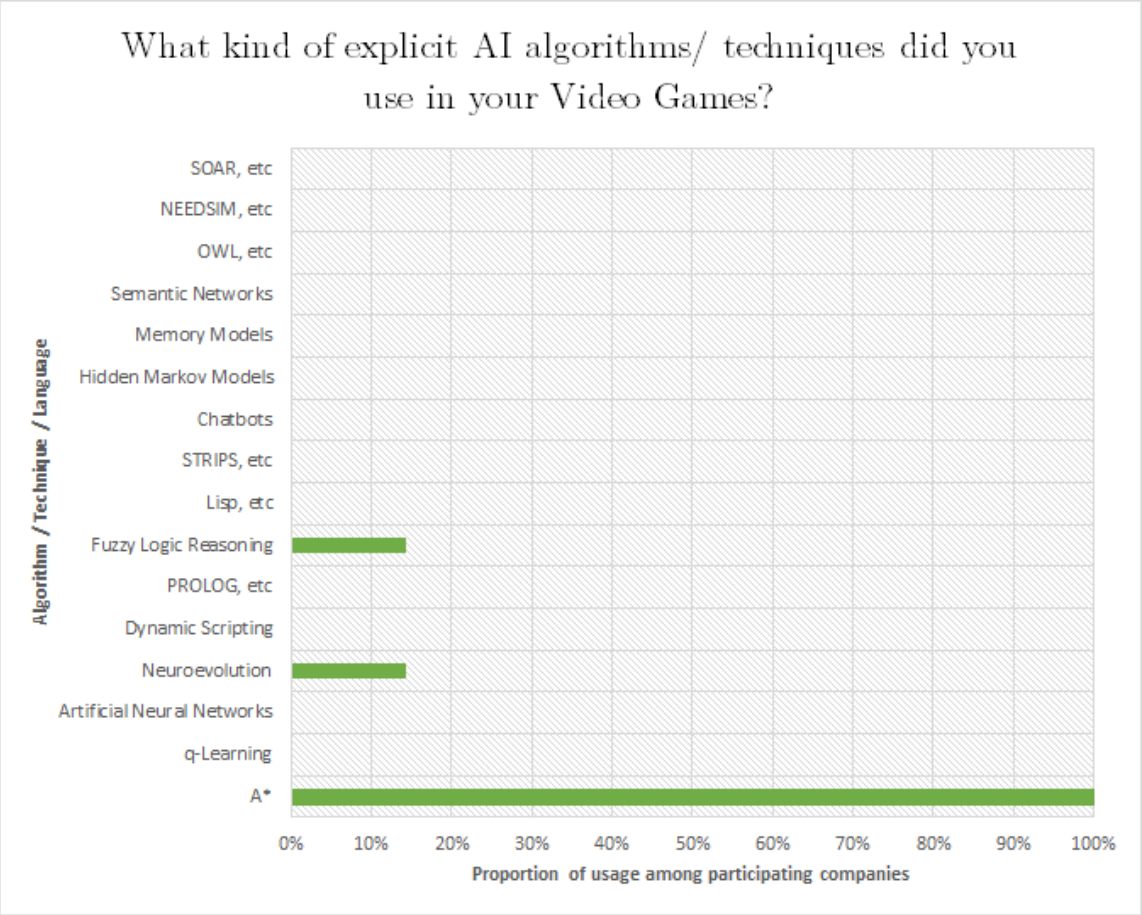


Figure 1.3: Proportional results of AI algorithm usage

When it comes to the algorithms that are popular and frequent in academic AI (Figure 1.3), the game industry refuses the usage in the most cases, again with the exception of the A* pathfinding algorithm invented in 1968. Very few other techniques are applied, e.g. some kind of Neuroevolution for *"Dynamic difficulties "* or fuzzy state machines to *"Making AI agents less predictable"*. The phenomenon of the contrast that many companies claim to integrate AI in their games but aren't using algorithms and methods stemming from actual academic Artificial Intelligence research may arise due to the definition of AI itself, which is in the scientific world *"The theory and development of computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages"* [9] and in the video game context *"[...] artificial intelligence consists of emulating the behavior of other players or*

the entities (that is, all the elements of the game that can act or be acted upon-from players to missiles to health pickups) they represent. The key concept is that the behavior is simulated. In other words, AI for games is more artificial and less intelligence. The system can be as simple as a rules-based system or as complex as a system designed to challenge a player as the commander of an opposing army [31]".

Figure 1.4: Results of the personal opinion of academic AI

Concerning the personal opinion about academic AI in industrial video games (Figure 1.4), developers were asked to state their assessment about 4 questions on a 5-level Likert scale, ranging from strong disagreement to strong agreement. They agreed that academic AI is quite interesting (on avg. 3.9/5), not only for science or developers, but also for the players and definitively makes the environment more believable (on avg. 4.3/5), which reflects the desire on the implementation and usage of AI, also from the industrial perspective. However, when it comes to the effort of implementation, developers answered pretty conservative (on avg. 2.9/5), with a clear message in the last question, that the state-of-the-art-industry goes a totally different path than academically developed AI (on avg. 2.6/5).

In the end, the state of the art in video game companies can be best expressed from the insiders themselves:

"What we call "AI" in games is vastly different than what's used in academia, or in business/engineering/apps/...Due to specific requirements like suspension of disbelief, games need a tighter control of possible outcomes and cannot afford the situation to be wildly misinterpreted. (E.g. as when speech recognition blurbs something completely out of context.)For those reasons, usage of "real AI" algorithms is very limited in games, especially in first-person games like what we usually make. A (mostly for path finding, only very rarely for other decisions) is the only exception, because the results are very contained. Using decision trees, goal oriented action planning, and similar is found in some games, but we still largely rely on hand-tuned conditions controlled by hard-coded ifs, state machines etc. If you care more about "plausibility" than "intelligence", experience shows that hand-tuned solutions go a long way further than emergent ones.Also, consider the fact that performance budget is severely limited especially if there's a large number of actors. E.g we once experimented with a very elaborate goal-oriented action planning algorithm heuristic for gun-fight tactics (choosing cover, targets, ...) where things like e.g. flanking were emergent results of the simple base logic resting on data like cover positions, precision estimation, etc... The results were impressive, but way too expensive. And could still produce unexpected results in some cases. When you consider that most games in that genre do away with prescripted actions for each possible scene, saving an order of magnitude on performance - and guaranteeing no unexpected behavior, you realize that there's still a long way to go for "real AI" in games.(Note: This all, of course, applies to first-person "realistic" games like we make. Different genres, especially e.g. strategy games, chess games, etc, do use what we'd call "real AI". But in massive shooters, it's not worth it.) "*

- Alen Ladavac, Croteam

"In order to make AI a noticeable feature where towns are full of interacting NPCs or where enemies are executing complex strategy, a company has to dedicate probably a dozen or more programmers/designers for over a year to set it all up, which is very expensive. Also, the more complex the AI, the more bugs that are created which reduces the polish of the game. We would love to have awesome villager AI with life like daily routines, but it's just too cost prohibitive."

- David Szymczyk, Obsidian Entertainment

"As game AI is focused on creating entertainment rather than primarily solve problems (which academic AI typically does), and usually has much stricter constraints on performance than academic AI, it is often faster to custom make solutions rather than use academic approaches. It also appears to be largely cheaper to produce a solution that fits the game and is 'correct enough' than actually implement a method that produces a correct result.I think for most game AI developers, the interest in using academically developed AI goes as far as it can improve specifics in AI behaviour reliably and within budget (both development resources as well as CPU and memory)."

- Carl Carenvall, Paradox Development Studio

"I think there are some opportunities to do more 'advanced' AI in video games, but, it probably means that these games needs to be build and designed 'around' these systems to make them really shine."

- Sandy Brand, Crytek

In other words, the desire for elaborate, complex and fascinating AI usage in video games is more than present, be it for challenging opponents, believable NPC behavior or the generation of content that no human designer could foresee. The main obstacles in this endeavor are agreed on amongst all developers: plausibility and performance.

1.3 Interesting content

Having the successes and failures of the academic research as well as the desires and necessities of the industrial viewpoint in mind, this thesis will focus on using AI, especially machine learning, to improve the quality of the challenge that simulated opponents offer to the player, or in other words: generate interesting content in the matter of game mechanics. Nowadays, a large proportion of popular modern video games rely on on-line mode to provide satisfying opponents, since human players inherently present interesting challenges. However, this only shadows the problem that a completely satisfying opponent for off-line games is simply non-existent yet - and a lot of games consist of many NPCs that can't be replaced by human players. The interestingness of such a simulated opponent stems not from intelligence, planning or mastering the particular game alone, just as little as from an opponent that would be defeated every time, but in balance and adaptivity. That means, it has to be "neither too hard nor too easy" [21] for the human player to win, but with continuing playing and learning of the game, the opponent also has to evolve to keep up with the human player. The ideal opponent is neither a flawless game player with superhuman abilities (as in powerful chess solvers like Deep Thought [16] and followers) nor a predictable, dull agent with idiosyncratic behavior which immediately differs from human-like behavior (like most NPCs). If no incentive to learn the game, overcome personal weaknesses and former playing mistakes is offered, a game is just as frustrating as an extremely hard game that is eventually impossible to beat. The first step in the direction of the balance between game challenge and player skill is usually the introduction of difficulty levels, but these go rarely beyond "easy, medium and hard", are heuristically tailored to the complete set of players and are either fixed, or no incentive is given for the player to change the difficulty after learning in the progress of the game. Commonly, games get more and more complex the longer they are played and thus require more skill to stay successful, but these increases in difficulty also lack adaptability and are set up pretty roughly.
Assuming that the ideal and most interesting opponent offers a game setup that is equally likely to win as it is to lose, so in the long run a win/lose ratio of 50% if the player doesn't come up with new insights or reinvents himself, then the ideal and most interesting opponent is neither a crowd of NPCs that express their challenge by mass (as e.g. in most FPS/RPGs), nor an overwhelmingly superior opponent that is only defeatable by the investment of much time, effort or the contribution of many other players (as e.g. in most MMORPGs/online games), but exactly one person:

You have to *challenge yourself*.

This is by far no easy behavior to accomplish, since mimicking each and every turn and action, and therefore mirroring the whole play of a game, would come out as predictable as primitive, hard-coded behavior rules. To get a satisfying reflection of the player's skill level, an intelligent opponent would start where the player start and evolves within his learning process. People are pretty fast in adapting to new situations and finding heuristics for how they can manipulate the environment for their benefit [22, p. 727], given a description of it and a known set of actions. This process follows a certain learning curve [61, p. 302] and once a strategy is found that proves itself as satisfying (but not optimal), the incentive of optimizing it drops. The functionality of this process can be adequately modeled with classic Reinforcement Learning, which consists of the observation of the reward-relation between given environmental conditions (states) and chosen actions, which can be learned and optimized. Based on behaviorist psychology and established in neuroscientific explanations [36, p. 3], Reinforcement Learning cannot only describe human reasoning, but gives a grounded account on the cognitive plausibility of the agent designed in this thesis. The addressed environment and states in the game used for the first benchmark of the algorithm are established by the total amount of important observable information at each point in time during the game, where the set of actions is exactly described by the possible moves available to the player (see: Complete state space description of Korona: Tinker's Curse at a single point of time). To emphasize the generalizability of the framework, these kind of environmental observations and possible actions are present in each and every video game, only differing in the magnitude.
Classic Reinforcement Learning has it's weaknesses though: In such a multi-dimensional setup, it takes an exponentially growing number of iterations to fully capture the spectrum of each constellation of game states throughout the whole game, which slows the learning rate down to an infeasible extent. Also, to fully grasp the spectrum of human cognitive abilities, game playing also requires predictive power, context integration, opponent estimation, temporal comprehension and the ability to develop a strategy from all these evaluations that maximizes the own utility, so in this case, that wins the game. Thus, this thesis will deal with different machine learning approaches and their combinations to approximate a realistic, dead even opponent.

Chapter 2

Korona: Tinker's Curse

2.1 Complete game description

The game *Korona: Tinker's Curse* is an integrated game in Nevermind Creations' adventure *Korona*, in which my work is mainly grounded in the implementation of game mechanics and development. In *Korona*, the player solves puzzles and quests, experiences a storyline and (among other things) is able to get access to cards that can be used to challenge various enemies in the game of *Tinker's Curse*. This itself is a purely logical, turn-based trading card game with changing opponents and without temporal constraints, consisting of 85 different cards, which are seperated into *Energies* and *Units* from the 7 distinct colors *red, yellow, green, blue, violet, black* and *white* (see: List of all cards in *Korona: Tinker's Curse*). At the beginning of each game, both of the 2 players start with a *Deck*, so a selection of 40 cards from the entire pool, where duplicate usage is legal and usual. At that point, each player has to pay attention that every *Unit* requires a certain amount of *Energy* of the same color on the board to play.

Example: The *Wachposten* ***(ID:37)***(Figure 2.1a) is a regular red *Unit* without effects, whose costs (upper right corner) require at least 2 red *Energies*(Figure 2.1b) on the board, before it can be played. Its *Attack* value is 1, the *Defense* value 3 (lower right corner).

(a) Sample Unit (b) Red Energy

Figure 2.1: Example: Playing a *Unit*

Thus, the occurence of *Units* without the corresponding *Energies* in the deck makes no sense (by now). The cards of each color include 1 specific *Energy* card, 9 different *Unit* cards and 2 different *Legendary Unit* cards, which are only allowed once per deck. In general, all cards can be distinguished by means of the following criteria:

Name		
Color		(red, yellow, green, blue, violet, black or white)
Type		(Unit or Energy)
if *Unit*:		
	Cost	(upper right corner, ranging 1-7)
	Attack	(1st value in lower right corner, ranging 0-8)
	Defense	(2nd value in lower right corner, ranging 0-8)
	Effect	(textbox)

The game starts with 40 cards in the deck of both players, where 5 are drawn in the beginning, making up the *Hand Cards*, so the set of cards that players can effectively access.

Areas

The game board is divided into 5 separate areas on both sides (Figure 2.2):

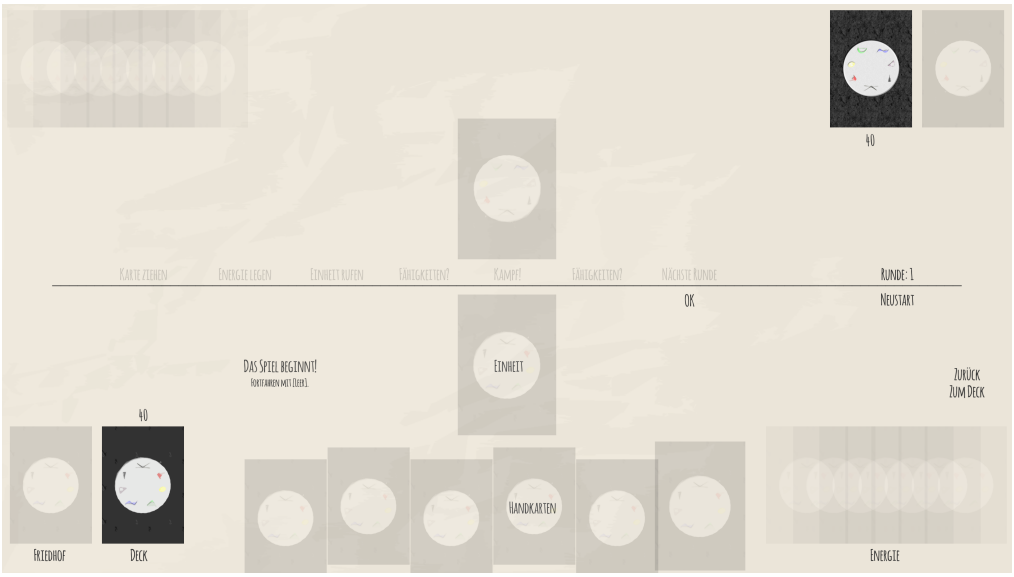


Figure 2.2: The board of *Korona: Tinker's Curse* before start

Deck:	The hidden stack consisting of all cards with which a player competes.
Hand Cards:	In the beginning of the game, each player draws 5 from the deck, which become the first hand cards. These are visible to the owner, but not to the opponent. In later phases, *Energies* and *Units* can be played from the hand. At the beginning of each turn apart from the first one, each player draws a card, however the limit of cards in hand amounts 6, so if a player would draw a card that would exceed this limit, he has to discard a card to the *Graveyard* before.
Graveyard:	A visible stack containing all cards that were discarded or destroyed.
Energies:	In this area, all energies will be placed that are necessary to play *Units*.
Unit:	One player is only able to control one unit at a time.

Figure 2.3: The phases of *Korona: Tinker's Curse*

Phases

Korona: Tinker's Curse consists of symmetrically proceeding phases (Figure 2.3), which are arranged in the same order each turn and continue if both players agree or aren't able to make an action anyhow.

Phase 1: *Draw card*: Each player draws a card from his *Deck* and adds it to his *Hand Cards*. If a player is not able to draw a card, he loses the game. If both players aren't able to do this, the game ends in a draw.

Phase 2: *Play energy*: If a player has *Energy* cards on his hand, this is the phase in which these can be played, but no more than 1 per turn.

Phase 3: *Play unit*: If a player has *Unit* cards on his hand, this is the phase in which these can be played, provided he has enough *Energy* of the respective color on the board that can cover the costs. If a player already has a *Unit* in this phase, he is able to replace it, i.e. place it into the *Graveyard* to make room for another *Unit*.

Phase 4: *1st ability phase*: If a player has a *Unit* on the field that has an *active ability*, this is the phase in which it can be activated, if the respective requirements are met.

Phase 5: *Battle*: In this phase the actual battle takes place. If no *Units* are existent on both sides of the board, this phase is skipped. If only one player controls a *Unit*, it deals damage to its opponent in the amount of its *Attack* value. The damaged player moves then that many cards from his *Deck* to his *Graveyard*. If both players have a *Unit*, they deal damage in the amount of their *Attack* values to each other - a *Unit* that receives damage which matches or exceeds its *Defense* value, dies and is sent to the *Graveyard*. The possible outcomes in this *Unit* vs *Unit* battle are thus: One *Unit* dies (e.g. 2/2 vs 1/1), both *Units* die (e.g. 1/1 vs 1/1) or no *Unit* dies (e.g. 1/2 vs 1/2).

Phase 6: *2nd ability phase*: If a player has a *Unit* on the field that has a *active ability*, this is the phase in which it can be activated, if the respective requirements are met.

Phase 7: *Next turn*: If both players agree to the last phase, a new turn begins, starting again with phase 1.

The game ends when one or both players lose due to the inability of drawing cards in phase 1 or when an effect triggered from a *Unit* causes a player to win.

2.2 Basic heuristic game opponent

Even though the procedure of the complete game seems easy to grasp (and by means of only a few previously defined heuristics a solid basic strategy can be composed), the actual playing of the game shows the immense complexity arising from the 78 *Unit* cards, already after a few matches. In total, these lead to 3081 unique *Unit*-vs-*Unit* combinations that can't be recognized and covered trivially in beforehand, but the main idea of the game is that the players have to discover the strengths and weaknesses of cards and decks while playing. However, to depict the way that industrially designed game opponents usually run through, a first, functional, simple heuristic game opponent is shown in Figure 2.4, which sets up the basis for more complex models.

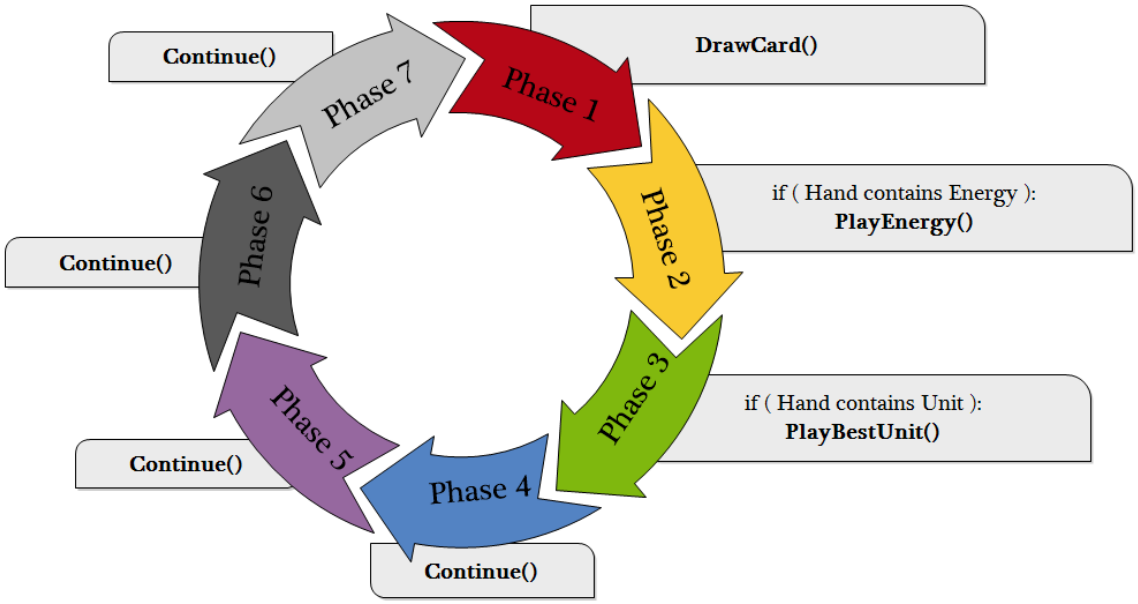


Figure 2.4: Basic heuristic model

In pseudocode, this opponent could be described as a Finite State Automaton that does the following:

if(phase == 1)	DrawCard()
if(phase == 2 AND Hand contains Energy)	PlayEnergy()
if(phase == 3 AND Hand contains Unit)	PlayUnit()
else	Continue()

In order to show that this very primitive model is sufficient, and in the long run to improve it piecewise, as it is common in FSA behavior tuning, the next subsections deal with the particular phases.

Phase 2: Energy phase

Already in phase 2, the opportunity of improvement is given. Admittedly, the strategy of merely playing each possible *Energy* from the hand seems to be a good first idea, since later *Units* possibly require much *Energy*. At second glance however, this naive *Energy* optimization blocks a number of possible moves or advantages. E.g., certain *Units* can make use of cards in the *Hand* of its owner, but no longer if they are already played.

Examples:	
(ID:15) Thvindazar	receives a bonus in both *Attack* and *Defense* for each card in the owners' hand.
(ID:22) Rastloser	can be brought back from the graveyard, if its owner discards 2 cards from his hand.
(ID:31) Mönch	gets one entire round of invincibility, if its owner discards 2 cards from his hand.
(ID:42) Gebrandmarkter,	(ID:43)Revolutionär or (ID:44)Weltbrandler under the control of the opponent are able to destroy energies on the board. If these *Units* cannot be blocked anyhow, there are cases in which it makes more sense to keep *Energies* on the hand instead of wasting them. The recognition of these cases however requires an understanding of context, predictive power and estimation capabilities - or summarized, intelligence.

Furthermore, the case of having several *Energies* of different colors in the deck can occur. Thus, the naive playing of the first available *Energy* on hand could delay the deployment of a differently colored *Unit*. After all, there is no trivial solution to this decision, since the maximization of *Energy* of one color would lead to the first mentioned dilemma, the strategy to balance the several colors however is similarly not always optimal, e.g. if the player already has an expensive *Unit* in hand, but his choice prolongs the building up of the *Energy* of that color. The question, whether and which *Energy* to play is therefore highly contextual and requires an estimation of what can be drawn in the later course of the game and what action would yield the most useful outcome to be answered intelligently. Finally, the costs for the most expensive *Units*, which are the *Legendary Units*, are always 7. Up to this point, it seems to be not optimal to have more than 7 *Energy* of one color on the board, if these are only used for playing *Units* (this changes if the deck contains *Units* that are capable of consuming *Energy* to activate certain effects).

Phase 3: Unit phase

The most difficult question undoubtedly appears in the third phase, which depicts the actual core challenge of the whole game: Which is the "best" *Unit* to play? Of these that are available on the hand, surely heuristics can be defined, such as:

Offensive choice:	Choose the *Unit* with the highest *Attack* value.
Defensive choice:	Choose the *Unit* with the highest *Defense* value.
Expensive choice:	Choose the *Unit* with the highest *Energy* cost.

As may be imagined, none of these heuristics truly dominates the others. Whether it is optimal to play an offensive or defensive unit is in the most cases situational and especially influenced by the constellation of the whole deck. E.g., in a pure blue deck that focuses on direct damage through *Units* like (ID:66)Aeronaut, (ID:67)Wolkenwanderer or (ID:68)Gewitterhetzer, the offensive choice is a viable heuristic, since most *Units* are not tailored to survive a potential battle with the opposing *Unit*, but possess the effect that they deal damage directly to the opponent, regardless if there is another *Unit* or not. On the other hand, a possible deck idea would be to delay the whole game by preventing damage and battles, until a card is drawn that is able to win the game in another way (e.g. (ID:13)Geldsack, which lets the player win if enough *Energy* is available). This variant would obviously prefer defensive *Units* and would be more successful in incorporating the defensive heuristic. After all, if a player plays such an extremely designed deck, one ought to think that the most expensive choice always is a good clue, since normally costs go along with utility, i.e. more expensive offensive *Units* are stronger than less expensive offensive

ones, and more expensive defensive *Units* are bulkier than less expensive defensive ones. Additionaly, this variant prefers in each case *Legendary Units*, whose values are in the most cases not significantly better than regular *Units*, but possess effects that make a great difference. Thus, for the first approach of a heuristic model, the most expensive choice seems to be the best choice, even if it displays inherent weaknesses: it cannot distinguish between offensive and defensive *Units* and neglects so the whole situational context, so that *Units* are often "wasted", since their application is mismatched.

Example:

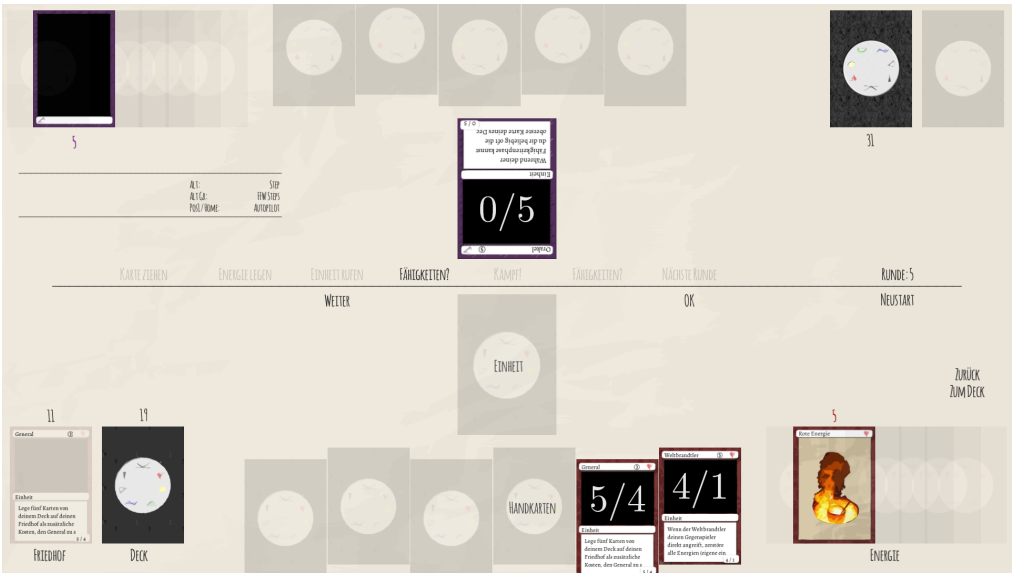


Figure 2.5: Choice dilemma for expensive choice heuristic

In Figure 2.5, one of many bad choices of the most expensive heuristic is portrayed. The opponent controls a (0/5) *Unit* which constitutes a solid defense. The player's only choices are between a (ID:41)General(5/4, cost: 3) and a (ID:44)Weltbrandler(4/1, cost: 5). In order to break the defense of the enemy, the only suitable choice would be the offensive one (ID:41) - the most expensive heuristic however would choose (ID:44), although the effect that explains the higher costs can't be triggered, since the opposing *Unit* is too defensive. Additionally, the most expensive heuristic bargains away each potential trump as soon at it is playable. In many cases however, it is highly advantageous to hold back better *Units* and sacrifice weaker ones to save the former for the appropriate situation.

Example:

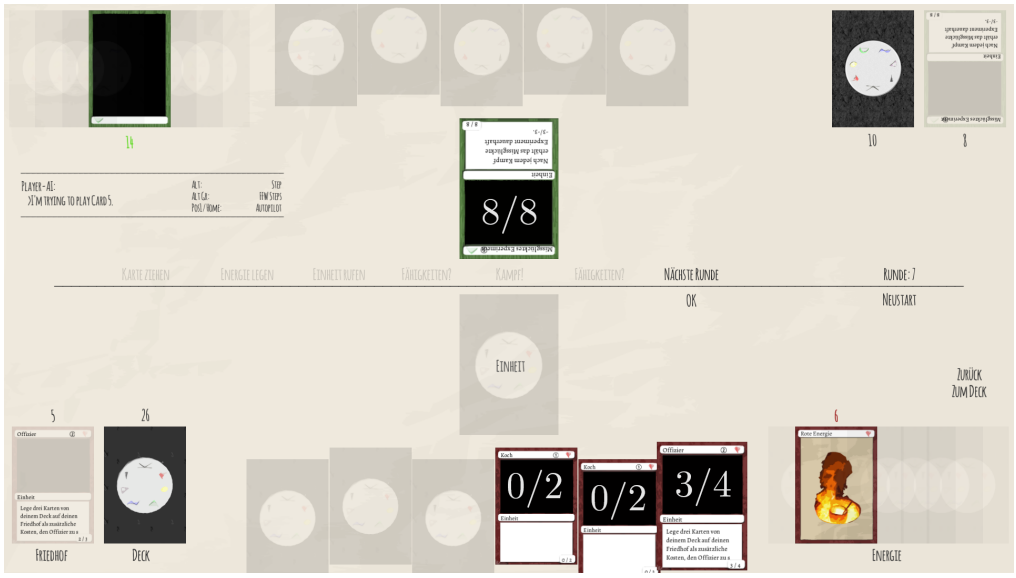


Figure 2.6: Choice dilemma for expensive choice heuristic

On the opponent side is a (ID:62)Missglücktes Experiment, a (8/8) *Unit* with the effect of permanently losing 3 points of both *Attack* and *Defense* after each battle phase. The hand of the player consists of two times a (ID:36)Koch(0/2, cost: 1) and a (ID:40)Offizier(3/4, cost:2). All the mentioned heuristics, the offensive, defensive and the most expensive choice strategy would use (ID:40) to fight the unbeatable opponent, although every *Unit* is inferior to it. An intelligent strategy would choose the cheap (ID:36) to face the threat, not only because fewer possible utility is lost, but also because the opposing *Unit* shrinks to mere *Attack* and *Defense* values of (2/2) after two battles, which can then be easily dominated with the leftover (ID:40).
Clearly, the last example only turns around the posing of the question, since if choosing the "best" *Unit* is a non-trivial task, then finding the "worst" *Unit* is likewise ambiguous. At this point it becomes clear how important contextual integration contributes to each step of the player's decision-making process. Also, the last paragraph depicts the strategy of "sacrificing the weakest" in desperate situations, which is used in the next chapter for more complex heuristics and should be learned both from human and artificially intelligent players in order to succeed in the long run.

Phase 4 and 6: Abilities

Both ability phases are intended for activating certain abilities of *Units*. Since most of these are completely different in concept and purpose (see: List of all cards in *Korona: Tinker's Curse*), it makes little sense to write general heuristics that activate effects, only because the opportunity exist to do so. Especially activated abilities impose often various costs, e.g. the sacrifice of own hand cards, *Energies*, *Units* or cards from the own deck. Activating these abilities in the wrong context, situation or amount not only shows idiosyncratic behavior, but also bears substantial disadvantages, where the non-activation only ends in missed advantages. To avoid these misactivations, a case differentiation for each single *Unit* that has an activated effect would have to be listed and coupled to a context estimation, where a simple and naive approach to a heuristic opponent is not enough.

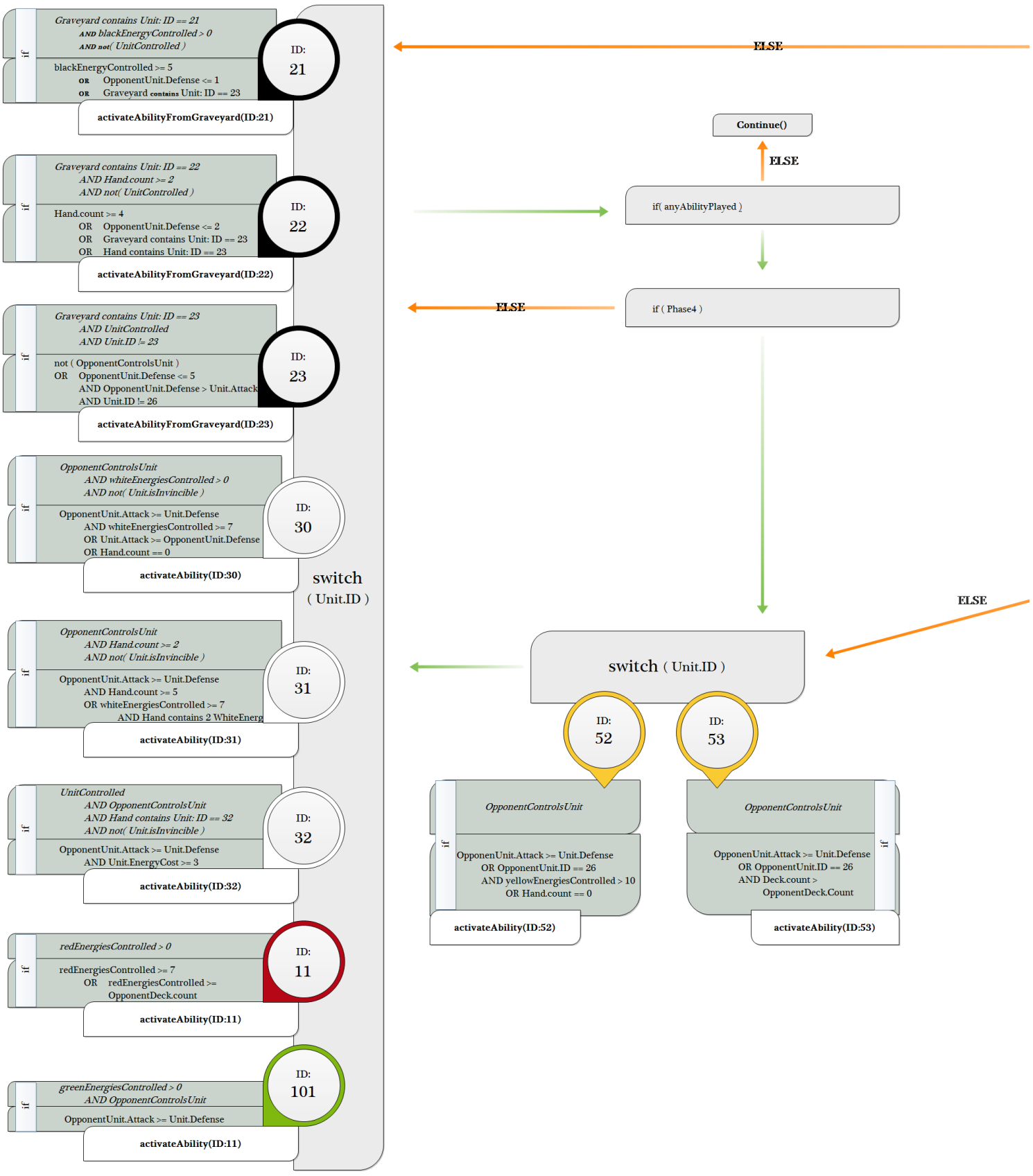
2.3 Advanced heuristic game opponent

As mentioned in the last subsection, the usual improvement of basic heuristic behavior FSA consists of manually hard-coded refinements that try to cover more and more cases. Having the challenges and dilemmas of the whole previous section in mind and assuming expert knowledge of the particular game with all of its situations is given, optimality criteria can be specified that state, which *Unit* or ability is beneficial in which situation (and beyond). The advantages are obvious: Once a complete, comprehensive heuristic strategy is defined, the FSA delivers a computationally cheap and debuggable answer for each turn and phase to the action selection problem. However, just at the moment of actual implementation of the whole model, or during the empirical evaluation of its weaknesses at the latest, the disadvantages become evident:

- The programming effort is incredibly fussy and expensive
- The strategy is rigid and invariable, if new game features, in this example *Units* or abilities are introduced in updates or expansions, new guidelines and heuristics must be set up manually, possibly even for combinations with other cards
- The strategy is always on its most performant status (a completely inexperienced player has to challenge the very same opponent as a professional one. This easily leads to the perception of too high difficulty, if incentives and progress are not experienced quickly)
- In return, the strategy is not adaptive (a professional player won't see a challenge in this opponent once he passes the performance of it)
- The strategy is predictable (the longer a player faces this opponent, the more obvious it becomes that the naively most optimal choice is made each and every time. This predictability is exploitable by the player who can come up with surprise attack moves, but exactly these interesting moves are absent in these rigid models, what proves the short-term nature of the challenge or interestingness again)

- The strategy has no temporal context. It is able to include all cards that are present in the game into its decisions, but opposite to a human player, it lacks the estimation capabilities about the further course of game, the open chances and the preceding turns.

For reasons of demonstrations and comparability, Figure 2.7 (full resolution [40]) illustrates such a comprehensive heuristic model that includes expert knowledge about every *Unit* and ability and provides a benchmark base for the later introduced intelligent game opponents.


if
Graveyard contains Unit: ID == 21
AND blackEnergyControlled > 0
AND not(UnitControlled)
blackEnergyControlled >= 5
OR OpponentUnit.Defense <= 1
OR Graveyard contains Unit: ID == 23
ID: 21
activateAbilityFromGraveyard(ID:21)
if
Graveyard contains Unit: ID == 22
AND Hand.count >= 2
AND not(UnitControlled)
Hand.count >= 4
OR OpponentUnit.Defense <= 2
OR Graveyard contains Unit: ID == 23
OR Hand contains Unit: ID == 23
ID: 22
activateAbilityFromGraveyard(ID:22)
if
Graveyard contains Unit: ID == 23
AND UnitControlled
AND Unit.ID != 23
not (OpponentControlsUnit)
OR OpponentUnit.Defense <= 5
AND OpponentUnit.Defense > Unit.Attack
AND Unit.ID != 26
ID: 23
activateAbilityFromGraveyard(ID:23)
if
OpponentControlsUnit
AND whiteEnergiesControlled > 0
AND not(Unit.isInvincible)
OpponentUnit.Attack >= Unit.Defense
AND whiteEnergiesControlled >= 7
OR Unit.Attack >= OpponentUnit.Defense
OR Hand.count == 0
ID: 30
activateAbility(ID:30)
switch
(Unit.ID)
if
OpponentControlsUnit
AND Hand.count >= 2
AND not(Unit.isInvincible)
OpponentUnit.Attack >= Unit.Defense
AND Hand.count >= 5
OR whiteEnergiesControlled >= 7
AND Hand contains 2 WhiteEnerg
ID: 31
activateAbility(ID:31)
if
UnitControlled
AND OpponentControlsUnit
AND Hand contains Unit: ID == 32
AND not(Unit.isInvincible)
OpponentUnit.Attack >= Unit.Defense
AND Unit.EnergyCost >= 3
ID: 32
activateAbility(ID:32)
if
redEnergiesControlled > 0
redEnergiesControlled >= 7
OR redEnergiesControlled >=
OpponentDeck.count
ID: 11
activateAbility(ID:11)
if
greenEnergiesControlled > 0
AND OpponentControlsUnit
OpponentUnit.Attack >= Unit.Defense
ID: 101
activateAbility(ID:11)
ELSE
Continue()
ELSE
if(anyAbilityPlayed)
ELSE
if (Phase4)
ELSE
switch (Unit.ID)
ID: 52
ID: 53
if
OpponentControlsUnit
OpponenUnit.Attack >= Unit.Defense
OR OpponentUnit.ID == 26
AND yellowEnergiesControlled > 10
OR Hand.count == 0
activateAbility(ID:52)
if
OpponentControlsUnit
OpponenUnit.Attack >= Unit.Defense
OR OpponentUnit.ID == 26
AND Deck.count >
OpponentDeck.Count
activateAbility(ID:53)

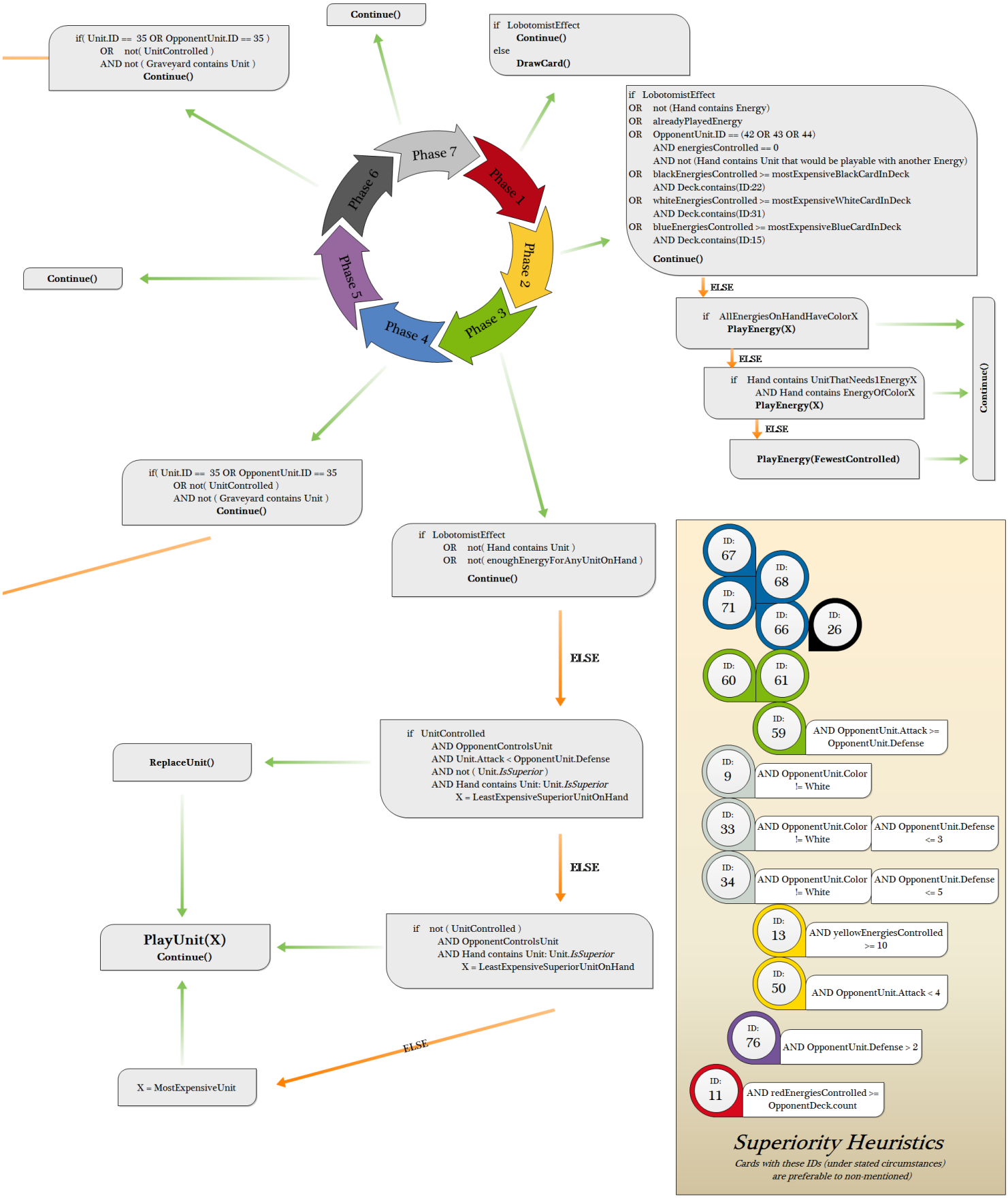


Figure 2.7: Advanced heuristic model

2.3.1 Extensive documentation

In the following, the realization of the advanced heuristic model (see: Pseudocode: Advanced heuristic model) is explained thoroughly. It is still based on the first basic model, so phases are skipped or continued if no useful choice can be made and the magnitude of *Energies* and *Units* is preferably maximized, but the performance was improved through many manually hard-coded constraints and heuristics stemming from human conceptions about the utility of *Units* and abilities which were improved and refined in a high number of test runs. In e.g. the playing of *Energies*, every insight of the last section is applied: For the cases in which red, *Energy*-destroying *Units* could attack unhindered, the model refuses to play these *Energies*, just as when the deck contains cards that can make more use of them if they are not played. Where the base of this phase is rather heuristic, the concrete case differentiations are really dependent on hard-coded constraints made up for each and every card. This becomes even more distinct in the phases 3,4 and 6, where not only a legality condition is set up for each card (in order to prevent idiosyncratic tries), but also a criterion that decides on the utility of the *Unit's* or ability's use.
The basic heuristic to defeat a given *Unit* of the opponent is "Choose the cheapest *Unit* that is able to defeat the opponent's *Unit*, if this is not possible, sacrifice the very cheapest", which seems to be slim and suitable for many situations, but ignores a lot of *Units* with special intended purposes which have to be included via hard constraints. Thus, the selection of the *Unit* in phase 3 is highly overloaded with incredibly specific individual cases that have to be stated to not miss the *Unit's* advantages. E.g., 7 *Units* are generally preferred over *Units* that are not stated in the "superiority heuristics list" and will be automatically replaced if they aren't able to defeat the opposing *Unit*. Most of the other individual cases are explained easily if the effect of the particular card is known, e.g. (ID:33) which is a 1/1 *Unit* that becomes (3/1) if the opponent controls a non-white *Unit*. If no contextual knowledge is present in such a strategy model, this effect has to be hard-coded to render the behavior not completely naive and narrow-minded. In the previous case, as well as for the rest of the *Unit* constraints, the purpose is reflected and easily recognizable in the constraints (e.g. some *Units* use the fact that the opposing *Unit* cannot defeat it and activates then some passive effects, or other, for humans trivially identifiable knowledge that cannot be calculated naively from the information of the *Unit's* values).
The problem of activating abilities is imaginably more difficult. Again, both legality and utility constraints are shown, to avoid attempts of activating abilities when it's not legal due to the rules of the game and to integrate clues in which situations these abilities are actually useful. The former are rather trivial cases, e.g. "In order to activate the effect of (ID:21) this card has to be in the *Graveyard*, the owner has to control at least 1 *Black Energy*, but no *Unit*". The effect allows the revival of the (1/1) *Unit* at the cost of sac-

rificing one *Black Energy*. This may have use in many cases, but is by far not optimal in every situation in that it is legal (e.g. one could play a more optimal *Unit* from the Hand without sacrificing *Energy*). Thus, utility constraints define situations in which the activation of this effect is appropriate. In this case it would be "if *enough black Energies* are present", "if the opposing *Unit* can be defeated by this" or "if the *Graveyard* contains (ID:23)". In the first case, the immediate question arises, how much *Energy* is *enough* to allow painless sacrifices, which is most of the time dependent on the situation and thus hard to cover in a hard-coded constraint. The third case however is a very tricky one, since here the application of the combination of effects of different *Units* comes into play. In this example, it could be very useful to sacrifice a *Black Energy* to revive the (1/1) *Unit* and replace it directly after it by activating the ability of (ID:23), which brings it back from the *Graveyard* to play and represents one of the most offensive *Units* of the black color with the values of (5/1). This constraint requires and implies higher-level game knowledge about tactics and mechanics that most human players only learn while playing and are not contained implicitly in a single card. The advanced heuristic opponent shows that it is not impossible to simulate these tactics with the simplest, hard-coded algorithms, however it should now become apparent that by increasing the number of *Units* and abilities these combinations between the various cards becomes exponentially bigger and do not stop at combination with only 2 cards.
The further requirements and constraints in the ability phases show likewise heuristics that seem arbitrary in the first place, but can improve the performance of the model from an empirical viewpoint. However, these are only constructed by the mere intuition and gut instinct of human programmers which makes them never perfect in performance, nor easily adaptable or extensible, nor trivial to assess and set up, nor in any way scientifically or game theoretically interesting. Still, since the invention of video games, digital opponents are either built on these loose structures or by exploiting the internal informations of the game that are not accessible by the player (e.g. the usual FPS opponent knows the position of all players in the game, but is then artificially worsened in the common "easy, medium, hard" steps to provide a beatable challenge). The following chapters thus will concern intensively with the outpacing of these mentioned paradigms.

Chapter 3

Feature-by-feature machine learning and analysis

To grasp the complexity of the whole card game, it is reasonable to begin with the modeling of the fundamental building blocks that will be extended and combined in later steps. This approach shows not only the general feasibility, but also the development of performance, strengths and weaknesses of different models in various situations and after all, this process demonstrates how successful game play can be developed even in an environment that is highly complex and difficult.

Choosing this bottom-up procedure, this section deals with the core of the game: the confrontation of two *Units* on the board. Winning this battle means deciding the whole turn for one player in most of the cases, which yields advantages for the later turns - thus, many won battles lead to the winning of the whole game. So, the basis for every intelligent system should at least contain the ability to master this confrontation. Again, to start at the most simplified level, certain game aspects like uncertainty, chance or resource scarcity are excluded in the first setups.

3.1 Setup 1: Learning Unit relation

- No uncertainty (Opponent's *Unit* is known to the agent)
- No chance (The agent can use every existing *Unit* of the set at all times)
- No resource scarcity (The agent can play every *Unit* unlimitedly)
- No costs (No unit is more expensive than another)
- Reduced card space (The set of cards is restricted to 10)
- Fixed decks (The opponent's *Unit* options do not differ from the agent's)

This setup restricts the set of *Units* to the following:

ID:	0	1	2	3	4	5	6	7	8	9
Values:	0/0	0/1	1/0	1/1	1/2	2/1	2/2	2/3	3/2	3/3

Figure 3.1 demonstrates the strictly deterministic outcome of each *Unit*-vs-*Unit* pairing, where IDs in y display the opponent's *Unit* and IDs in x the possible choices for the agent's *Unit*. + stands for defeating the opposing *Unit*, - for losing the battle and # for a tie.

ID:	0	1	2	3	4	5	6	7	8	9
0	#	+	#	+	+	+	+	+	+	+
1	-	#	#	+	+	+	+	+	+	+
2	#	#	#	#	+	#	+	+	+	+
3	-	-	#	#	+	#	+	+	+	+
4	-	-	-	-	#	#	+	+	+	+
5	-	-	#	#	#	#	#	+	#	+
6	-	-	-	-	-	#	#	+	#	+
7	-	-	-	-	-	-	-	#	#	+
8	-	-	-	-	-	-	#	#	#	#
9	-	-	-	-	-	-	-	-	#	#

Figure 3.1: *Unit*-vs-*Unit* outcome in setup 1

Thus, Figure 3.1 shows that for the first 8 cards, there is (at least) one strategy to win. Example: Opponent chooses (ID:6)(2/2). To win, counter this with (ID:7)(2/3) or (ID:9)(3/3).
For the last two cards, no more than a tie can be achieved (since (3/2) and (3/3) are not beatable in this set without defeating the other *Unit* as well). Assuming the opponent is choosing his *Unit* randomly, the optimal win probability is at 80% on average, which constitutes the upper bound for the following simulations.
For comparison, the lower bound is considered to stem from a random choice player. The win probability of this player is at 34% on average, calculated by the average of the addition of all row probabilities (E.g., (ID:0) can be beat 8 times out of 10, so the row probability is 0.8. In total, ((0.8+0.6+0.5+0.5+0.4+0.2+0.2+0.1)/10) results in 0.34).
These two boundaries are thus simulated by the first two players. An agent that serves the purposes of this thesis, to explore the structure of the game in an intelligent way, thus should start at the performance of a clueless (random) player and develop itself consecutively, like a human player would. Since no prediction is required in the first setup, the sole task here is to learn a mapping between two dimensions: the opposing *Unit's* ID (known in this setup) and the own *Unit* ID (with free choice in it, since no chance or resource scarcity is present). These two dimensions result in a target value, i.e. the (estimated) reward, in this case a value that represents winning (+1), losing (-1) or tieing (0). This environment is very suitable for the application of Reinforcement Learning, where I contrast various variants in the following to describe how certain parameters and decision strategies have influences on the win ratio.

3.1.1 Variant 1: Naive Reinforcement Learning

Yellow graph, 79.94% win after n=5000

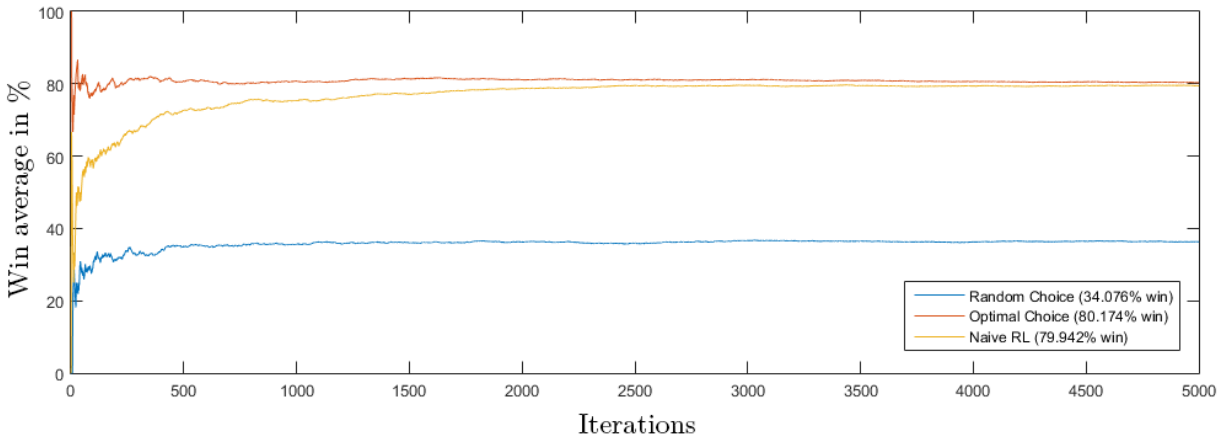


Figure 3.2: Win average for variant 1 over 5000 iterations

In the first variant, a probabilistic decision making strategy is chosen. That means, a 10x10 RL-map is initialized with values between 0 and 1, which constitute the decision probabilities in each iteration and are getting updated constantly in the actual battle iterations. Figure 3.3 demonstrates the initial probability distribution, ranging from low (indicated blue) to high (indicated yellow) probabilities. E.g., if the opponent *Unit's* ID is 4, the agent would most likely choose (ID:6) to counter it, and most improbably (ID:4).

In this fashion, classic Reinforcement Learning without punishment is applied to learn the various probabilities. In each battle iteration (in this case, 5000 times), the opponent chooses a *Unit* from ID 0 to 9 at random. The RL agent then decides his alternative on the basis of the (learned) map, probabilistically. Subsequently, both *Units* are confronted and the battle outcome (win, lose, tie) is evaluated. In the case of winning, a positive reward (+1) is added directly to the map, before the score is normalized again to represent the actual decision probability (negative rewards are left out in this variant). In the course of the 5000 iterations, the average winning curve approximates to 80%(Figure 3.2), which is the absolute optimum in this setup, the agent thus really converges to optimality after a certain time of learning.

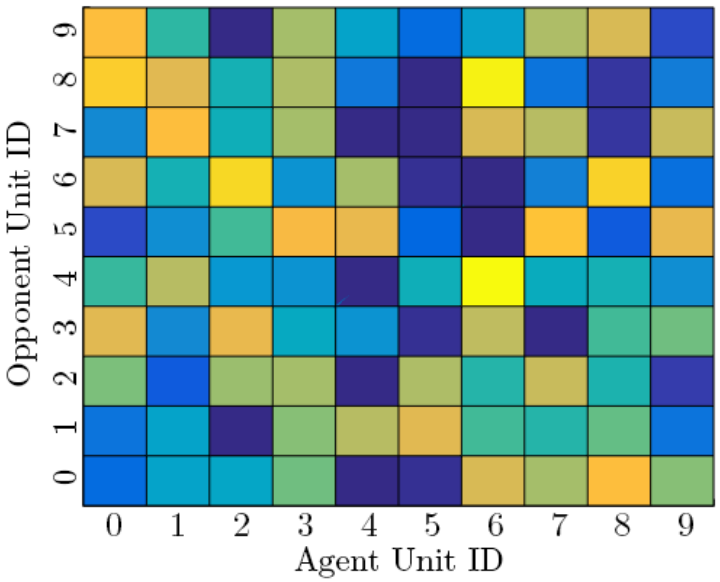


Figure 3.3: Randomly initialized RL map

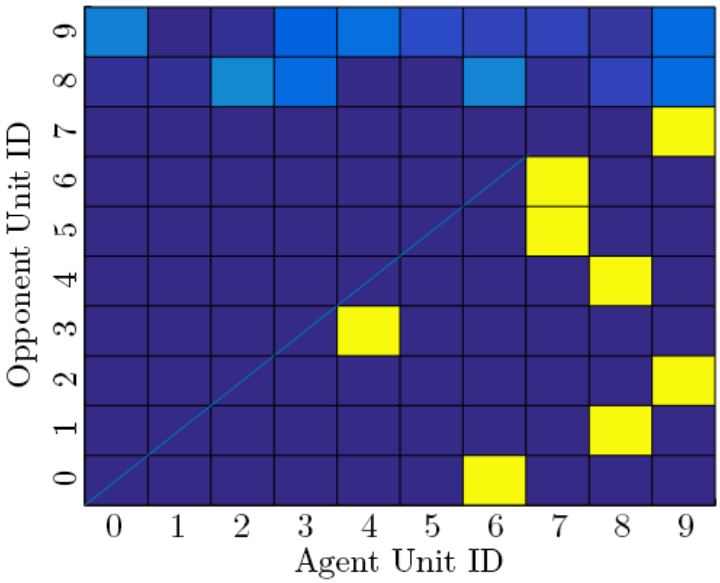


Figure 3.4: Variant 1 RL Map after 5000 iterations

Figure 3.4 displays the progress after 5000 iterations. The decision probabilities have shifted to the right in the most cases, as a higher ID correlates with a higher probability to win in this setup. Since for ID 8 and 9 only ties are possible, which weren't included in the first variant and thus no positive reward was added, the top two rows remain untrained. The choice of the favored alternatives (yellow) seems to be arbitrary, but this setup requires simply *some* winning candidate, i.e. to win a battle against a (ID:0), (ID:1,3-9) would be possible, but (ID:9) would yield no advantage over (ID:1) and vice versa. Thus, the agent stacks with the first winning alternative in the most cases, with a certain variance because of the probabilistic nature.

3.1.2 Variant 2: Discrete naive Reinforcement Learning

Light blue graph, 80.42% win after n=5000

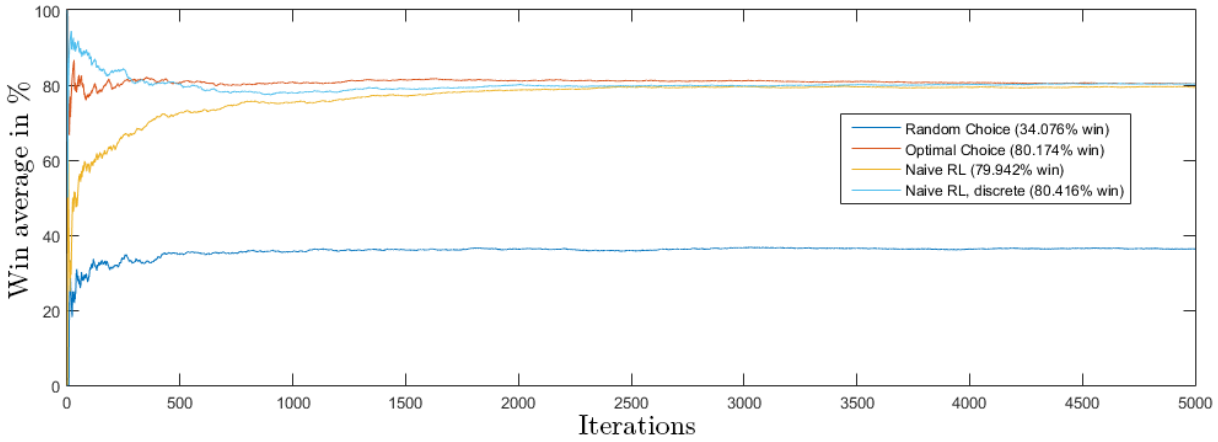


Figure 3.5: Win average for variant 2 over 5000 iterations

In contrast to the former *probabilistic* RL which needs a certain amount of iterations to smoothen the variance, a *discrete* approach of RL is immediately more performant. The only difference in these is the decision strategy in the end, where the first variant weighs the different alternatives in the amount of its estimated reward and this variant goes always for the highest estimated reward which lets it learn faster for this simple setup. This leads to the total exclusion of all other alternatives if a winning candidate is found, which can lead to a radical result like shown in Figure 3.6.

For (ID:8-9) obviously no winning candidates can be found either, but this player chooses strictly the variant that is superior in the most cases for every opponent (ID:9),

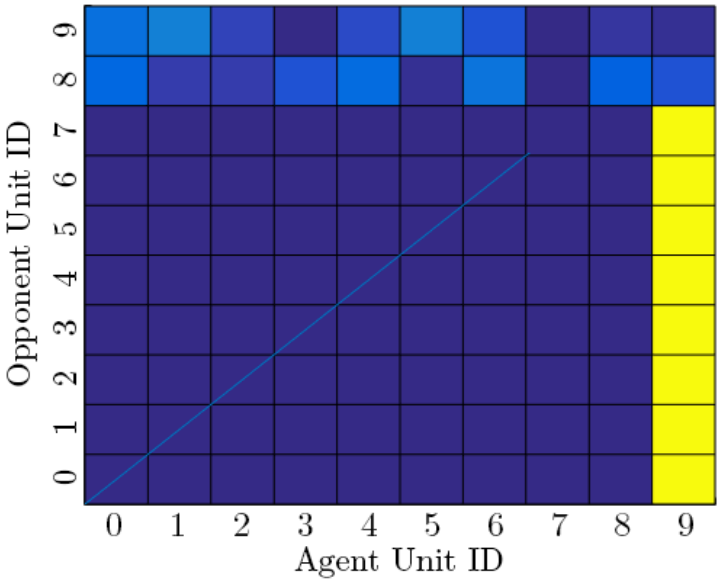


Figure 3.6: Variant 2 RL Map after 5000 iterations

which is a viable winning strategy in this setup, but could cost valuable resources in later setups. Despite the fact that it can't beat the 80% that variant 1 also reached, it gets there in a much faster way, which is important to have in mind for performance issues, but not the kind of learning that was desired in the first place when describing the framework of an adaptively learning agent.

3.1.3 Further RL Variants

To benchmark various variants of Reinforcement Learning, the following insights are gathered by altering the possible parameters of the map, learning and decision process.
- Increasing the winning reward numerically (**Purple, 75.8%**) can lead to a faster learning process in the probabilistic variant, since it will focus faster onto the relevant winning candidates. However, it takes a significantly longer time to find this winning candidate if it is initialized with a very low value in the beginning.
- Decreasing the winning reward numerically (**Green, 71.4%**) leads to a decent visualization (Video: Visualization: Slow Reinforcement Learning [46]) at cost of learning speed, thus the approximation to optimality is slowed down. However, this ensures that variants are handled carefully, so that winning candidates are rarely missed.
- The implementation of punishment via negative reward at losses (**Dark red, 78.2%**

probabilistic, **Dark blue, 79.9%** discrete) leads to the preference of tie-alternative to loss cases, which is not essential for this setup, but a conclusive factor for further purposes and intelligent behavior. Note that this alteration slightly slowed down the learning process in the tested trial.

3.1.4 Neural Network

(**Black graph, 80% win after n=5000**)
Even though the training of an artificial neural network for such a simple setup requires a significantly higher computation time in contrast to filling a RL map, it is important to mention this alternative early, since the more complex the tasks and setups become, the more game context has to be learned and interpreted, where neural networks are able to achieve this very performantly. Where the presented RL approach becomes lost in complexity the higher the dimensionality becomes, deep learning can break down this dimensions into hierarchical representations modeled by a number of hidden layers. Thus, also the development of a NN player (and its combination with RL) will be documented from the simplest to the most complex problem.

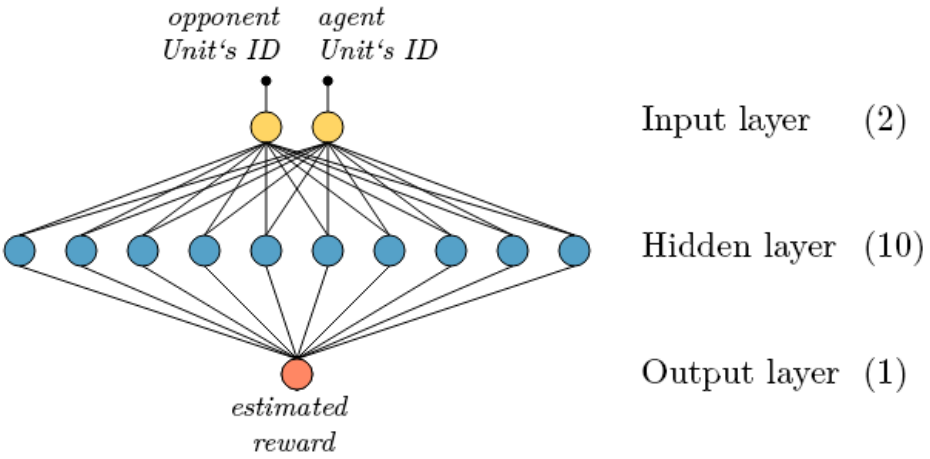


Figure 3.7: 2-10-1 Neural Network setup

For the purpose of analyzing the core features in this section, MATLABs Neural Network Toolbox [1] is used, where for the later actual in-game usage and the development of the *Challenge Yourself*-plugin a particular implementation is written to evade the computational bottlenecks of programming language wrappers or tunnels. In this simple setup, a 2-10-1 network (Figure 3.7) is sufficient to learn the classification of *Unit*-vs-*Unit* pairs into target values (estimated rewards). That means, the input layer of the NN consists of 2 neurons (the opponent *Unit's* ID and a possible *Unit* choice against it), from which the one-dimensional output of the reward (trained with -1, 0 or +1) is approximated through a hidden layer of 10 neurons. The network thus delivers, assuming the opponent *Unit's* ID, a reward for each of the possible agent *Unit's* IDs, which leads to 10 values. From these (like in the discrete RL case) the highest expected reward is chosen in the actual selection process, what results in a convincing learn speed and also approximates the optimal win average, as seen on the total performance comparison (Figure 3.8).

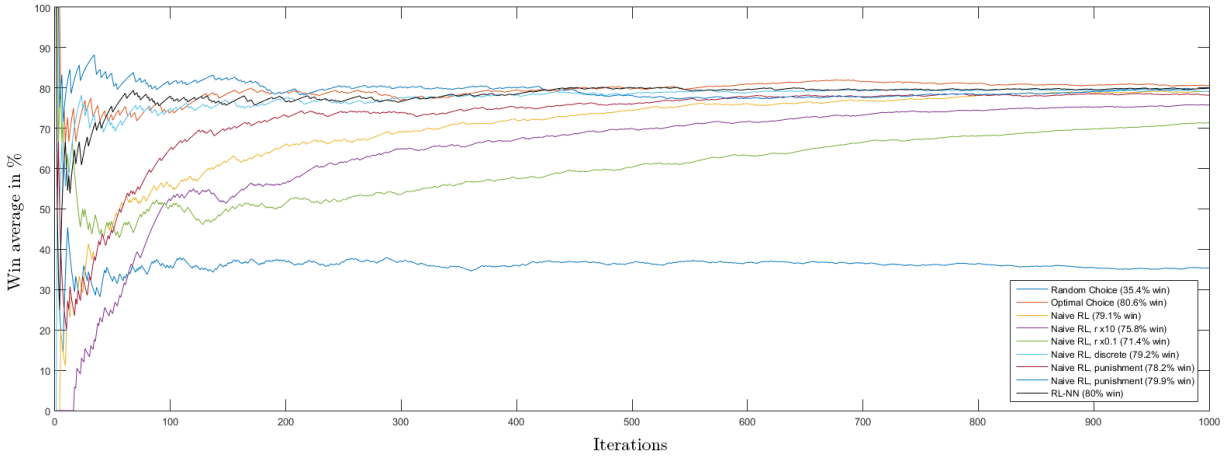


Figure 3.8: Win average for all variants over 5000 iterations

3.2 Setup 2: Without Replacement

As in the previous setup, following assumptions stay to demonstrate the modularly advancing progress:
- No uncertainty (Opponent's *Unit* is known to the agent)
- No chance (The agent can use every existing *Unit* of the set at all times)
- No costs (No unit is more expensive than another)
- Reduced card space (The set of cards is restricted to 10)
- Fixed decks (The opponent's *Unit* options do not differ from the agent's)

Thus, the stepwise complication is made up through:
- Resource scarcity (Each *Unit* can only be played once, until all cards have been selected)

This problem describes the classical "without replacement" scheme and is already now no longer trivial, since the total context of previous turns has to be included in the decision of a *Unit*. Strategies like in Figure 3.6 are not working anymore, since the prominent *Unit* (ID:9) can only be used once. Above that, the optimal win average changes, since one option disappears, as seen in Figure 3.9.

ID:	0	1	2	3	4	5	6	7	8	9
5	-	-	#	#	#	#	#	+	#	+
6	-	-	-	-	-	#	#	+	#	+
7	-	-	-	-	-	-	-	#	#	+

Figure 3.9: Extract of Figure 3.1

(ID:7)(2/3) can only be dominated by (ID:9)(3/3). For the winning against (ID:6)(2/2) and (ID:5)(2/1) however, a *Unit* is required that has a higher *Defense* value than 2, which are only (ID:7)(2/3) and (ID:9)(3/3). If (ID:9) is already used to counter an opposing (ID:7), only the own (ID:7) remains to beat either (ID:5) or (ID:6), but not both. Thus, in this setup, one winning option vanishes, defining 70% as the new maximum win average, as the optimal choice player proves empirically (Figure 3.10).

3.2.1 Variant 1: Discrete naive Reinforcement Learning

(**Yellow graph, 60.5% win after n=5000**)

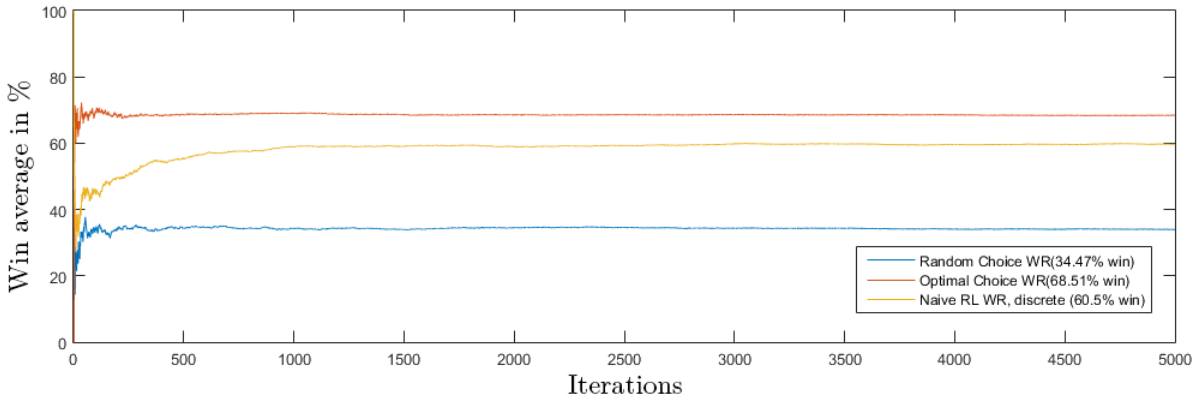
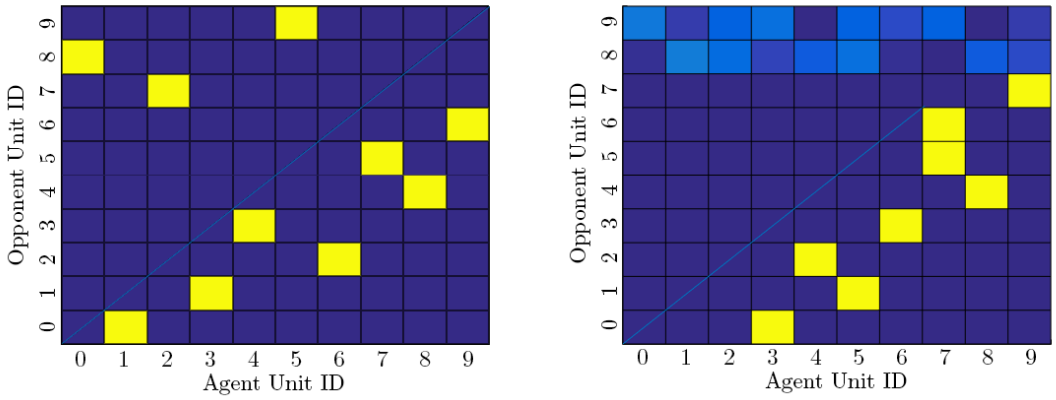


Figure 3.10: Win average for variant 1 over 5000 iterations

Beginning with the version that reached the optimal win strategy in the previous setup with the lowest temporal and computational effort (the discrete RL, reward multiplicator = 1, without punishment), this player reaches an average winning ratio of 50-60% that never converges to the optimal average (Figure 3.10).
Comparing the learned strategy to the optimal one (Figure 3.11), the latter is willing to lose three iterations and thus sacrifices weak *Units* against strong opposing *Units* that are not beatable anyway ((ID:7-9)). Thereby, 7 battles remain which do not require double usage of *Unit* candidates and thus can be won deterministically by following a simple algorithm:
- Opposing *Unit* is presented
- Starting at (ID:0)(if available), test if this *Unit* would win
- If yes, choose it (e.g.: (OpponentID:0), smallest winning *Unit* is (ID:1))
- Else, test next available ID
- If not even the greatest ID can win, choose smallest available ID and sacrifice it (e.g.: (OpponentID:8), smallest sacrifice is (ID:0))



(a) Optimal (b) Variant 1

Figure 3.11: RL maps of Setup 2

When observing the learning behavior of variant 1 on the basis of the RL map over time, the weakness becomes obvious: Even though some winning candidates are learned correctly (this, which are not blocking opportunities for other *Units*), the naive RL player also tries to learn these which cause these conflicts. Once such a candidate is learned, it is automatically excluded in all other cases in which it could be needed.

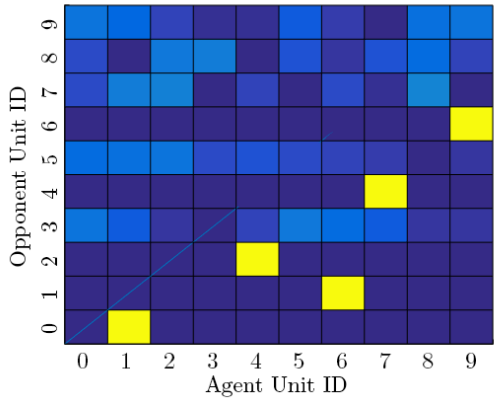


Figure 3.12: RL map of Setup 2, variant 1 in progress

As displayed in Figure 3.12, a opposing (ID:0) is accidentally countered optimal (in terms of resource scarcity) by the usage of (ID:1), the next choice of (ID:6)(2/2) against (ID:1)(0/1) might win also, but gives away the opportunity to use (ID:6) for stronger opponents. E.g., against (ID:3), at least (ID:4) (which is in this case already used against (ID:2)), (ID:6) or higher is required. Since all alternatives are unavailable for this opponent, row 3 remains blue, i.e. no candidate can be found that fits into that strategy. The same holds for (ID:7) which is not beatable if (ID:9) is already used, that in this case counters the opponent's (ID:6). Concluding, even if the discrete player yielded convincing results very quickly in the first setup, the integration into the total context is missing, which will be included in the further variants in various ways.

3.2.2 Variant 2: Reinforcement Learning with heuristics

(**Purple graph, 69.3% win after n=1000**)

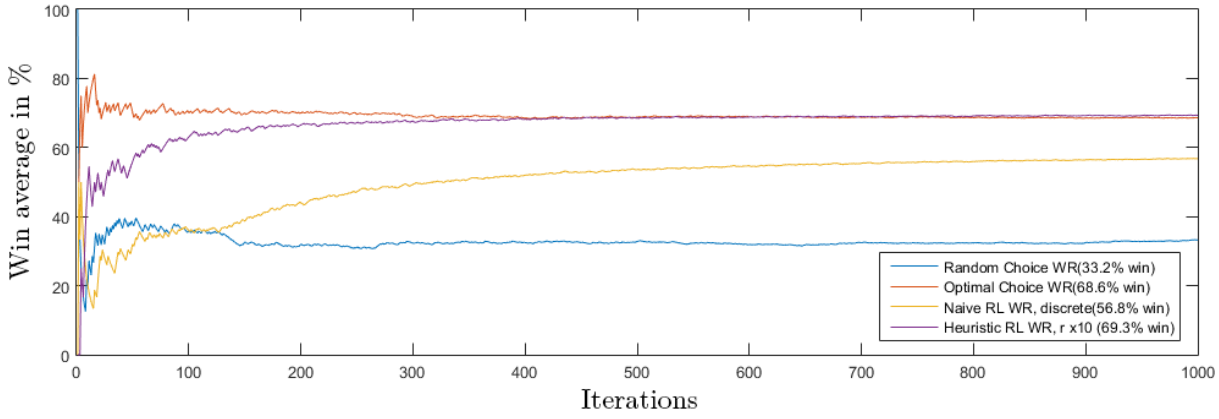
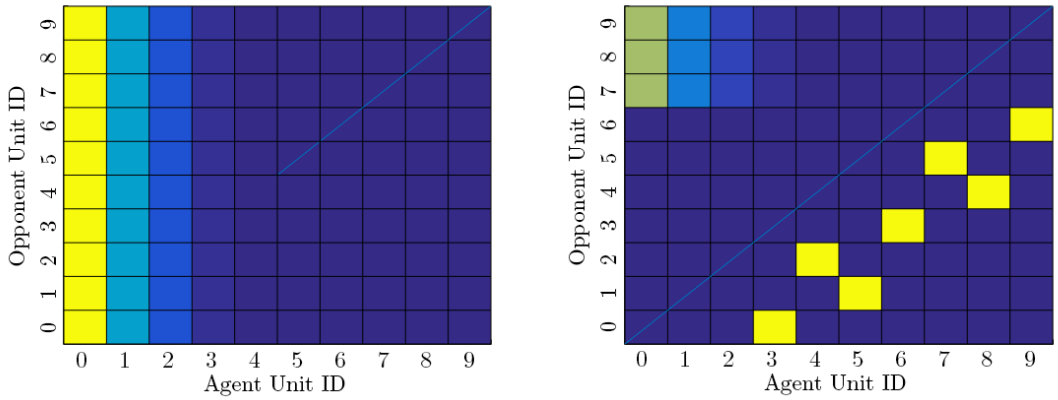


Figure 3.13: Win average for variant 1 over 5000 iterations

A first approach to the problem of conflicting alternatives would be to improve the former completely random choice by giving it a hint about the value of *Units*. The usage of those heuristics is common in scientific Artificial Intelligence [37], but comes with the flavor of tuning the learner with external, human knowledge about the game. For the purpose of this demonstrational setup, this fact doesn't harm the reasoning process, however on the long run this thesis aims to develop truly intelligent and interesting opponents with the elimination of conventional strategies that succeed only because of the external knowledge of their programmers without a learning process of the game.



(a) Initialization

(b) Learning after n=1000

Figure 3.14: RL maps of Setup 2, variant 2: Heuristic RL

Instead of randomly initialized RL maps, this player is provided with an initial map (Figure 3.14(a)) that generally prefers *Units* with smaller IDs over *Units* with bigger ones. This does not contain the actual winning strategy (since the relation between ID and *Unit* superiority is not linear), but improves the learning process significantly, because the agent is not getting stuck in big alternatives (e.g. (ID:9) vs (ID:0)). As the learning curve in Figure 3.13 demonstrates, the result is more than convincing, within 100 iterations the win average stabilizes onto the optimal level, since an optimal candidate is found for every winnable case, where for the non-winnable cases the high probabilities still remain in sacrificable low IDs.

3.2.3 Variant 3: Temporal Difference Learning

A substantially more interesting approach to this problem that (in comparison to the heuristic variant) is not dependent on external hints is depicted in Temporal Difference Learning. In this variant of Reinforcement Learning the reward is not distributed immediately, but rather after a sequence of iterations. This allows the contextual information about all 10 choices made in the process onto the particular candidate fields on the map. The difference to the former classical RL approach thus lies in the training, which changes from:

- Given the opponent *Unit's* (ID:y), choose (ID:x) and evaluate
- If x wins, record positive reward r to (x/y) on the RL map
- If y wins, record negative reward -r to (x/y) on the RL map

to:

- Given the opponent *Unit's* (ID:y), choose (ID:x) and evaluate
- If x wins, add positive reward r to r_cumulated
- If y wins, subtract negative reward -r from r_cumulated
- Save (x/y) and r_cumulated temporally
- After 10 iterations, record r_cumulated to all saved (x/y) on the RL map

In theory, r_cumulated constitutes the reward in the total context of 10 iterations which turns out higher the less conflicting candidates were chosen. In the end, the learner tries to maximize r_cumulated to reach an optimal arrangement of *Unit* alternatives.

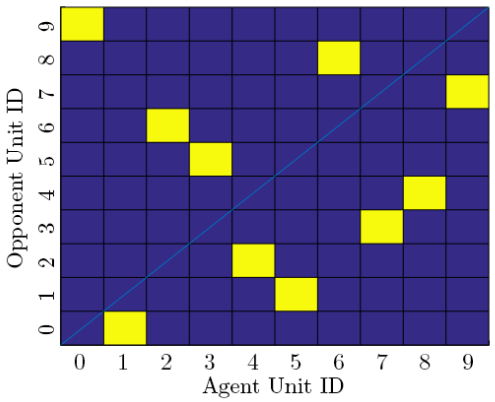


Figure 3.15: RL map of Setup 2, variant 3

In practice however, this implementation of TDL is in fact able to learn all 7 optimal applications of *Units*, but not consistently, i.e. there are cases in which the map is trained to constellations like shown in Figure 3.15, where the algorithm ends up in a local minimum, i.e. a "good" but not optimal solution is found, but the reward for this constellation is too high to search for further alternatives. Nevertheless, TDL can be combined with other methods, heuristics oder implemented in other machine learning approaches, thus it is kept in mind for its ability to model temporal successions.

3.2.4 Variant 4: TDL-NN

Combining the power to integrate a large context with the ability to add a time dependency to the reward leads to the hybrid of the Temporal Difference Learning-Neural Network. In this approach, the specific superiority relations amongst the *Units* are assessed with the NN (as proven in chapter 3.1.4) with the addition that it can model the temporal problem of the new setup.

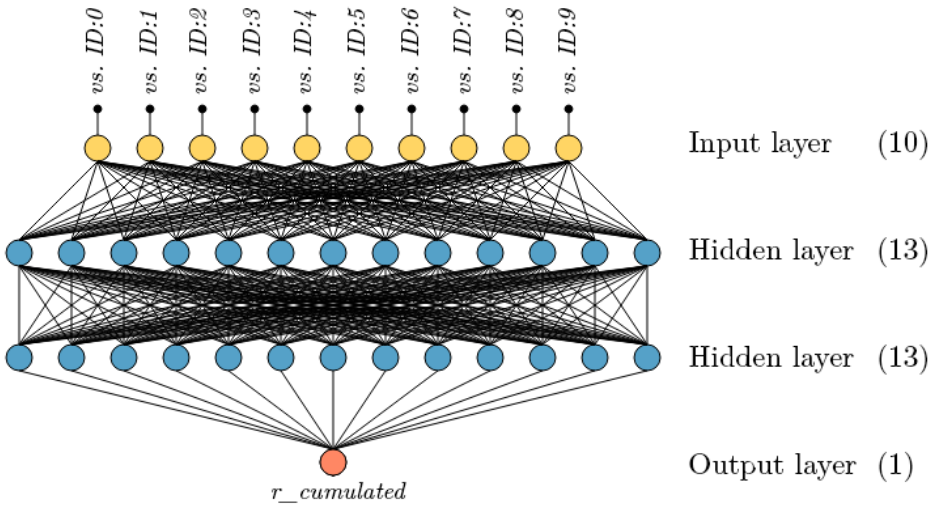


Figure 3.16: 10-13-13-1 Neural Network setup

In contrast to the previous version, this NN does not consist of two input nodes (where a given opponent *Unit's* ID is fixed and all agent's *Units* are tested on), but of a whole set of 10 candidates that state which *Unit* was used against which. Since all 10 IDs are included in each iteration, perfect information is provided for the input array, transformed to particular fractions of the total reward r_cumulated. This network is able to represent and compare whole constellations instead of single assignments of *Units*, which is why it is able to achieve the performance visible in Figure 3.18.
Thus, the TDL-NN approach is already able to record strategies rather than moves, assigning estimated rewards directly to chunks of *Unit*-vs-*Unit* confrontations, while prompting whenever a more optimal strategy is found during the course of iterations, as seen in Figure 3.17, which can be interpreted the following:

Starting with a random assignment of *Units*, which in this case yields a r_cumulated of 0.3, so 3 winning confrontations in a chunk of 10 battles (coming close to the performance of a random player), the agent increases its performance successively while playing, by coming up with better strategies. These can be completely new strategies (as from the strategy resulting in 0.3 to the one resulting in 0.4) or refinements, as seen in the later stages (the last improvement keeps a certain number of working *Unit* matches and alters these which are able to defeat bigger *Units* and thus lead to a better overall performance).

new best strategy (r_cumulated =0.3) :[1 9 3 0 7 5 2 4 8 6 0.3]
new best strategy (r_cumulated =0.4) :[6 0 7 4 1 2 3 9 8 5 0.4]
new best strategy (r_cumulated =0.5) :[5 2 6 4 7 9 1 8 3 0 0.5]
new best strategy (r_cumulated =0.6) :[1 3 4 2 6 7 0 9 5 8 0.6]
new best strategy (r_cumulated =0.7) :[1 3 6 4 8 7 9 0 2 5 0.7]

Figure 3.17: TDL-NN output

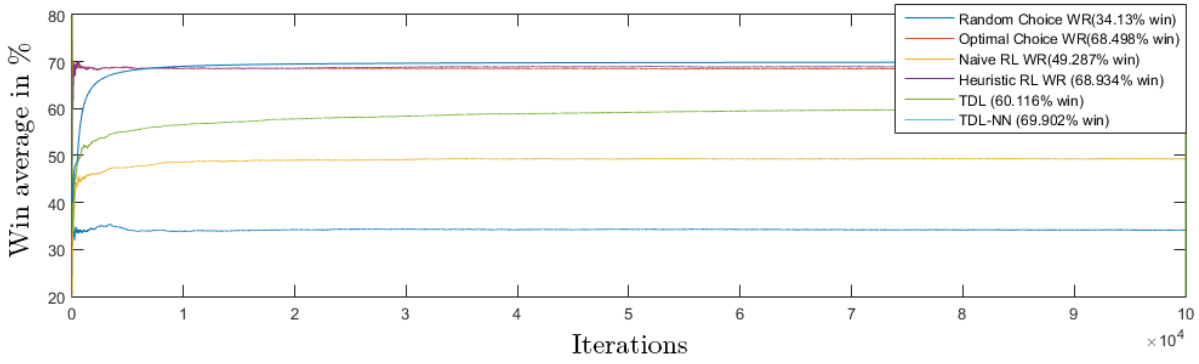


Figure 3.18: Win average for all variants of setup 2

3.3 Setup 3: Opponent Uncertainty

The following section will treat an important factor in video games, especially important in card games or similar 1-vs-1 situations:
- Uncertainty (Opponent's *Unit* is not known to the agent)

In contrast to the further setups, this section introduces the mechanic that both *Units* are presented at the same time, so the particular task to handle, without discarding the insights made in the previous ones, is an explicit problem of player modelling. 10 different cards are played in a specific order which intrinsically reflects the utility of the particular card, i.e. as long as a non-random strategy is played, certain preferences of some cards over others exist (e.g. possible non-random strategies include starting slowly with weaker *Units* to save trumps, starting right away with useful and strong *Units* to gather a head start, or more complex, composite strategies). Since in this setup the unconstrained choice over all *Units* is still there (thus, the preference is not dependent or influenced by a chance factor), it is further assumed that there are certain situations which render particular *Units* useful or not useful. This situational context is basically made up by:
- The already played cards of the opponent
- The already played cards of the agent
- The opponent's estimation of the agent's choice

The first point here is already covered in the last setup, cards that are already played are not playable anymore, which reduces the choice space and simplifies the choice estimation. The second point presupposes that the opponent is actually remembering the agent's choice, what has not to be the case necessarily, but required for the third point:
If the opponent has a (correct) estimation about the agent, the strategy of the former can be essentially influenced by this. E.g., the opponent would be able to hold back a card that seems to be optimal for the situation at first glance, if he knows (or estimates) that the agent also knows that this optimal card should be played and is able to counter it (by defeating it with a stronger *Unit* or by wasting it with a very weak one). This thinking results naturally in one of the oldest problems in classical scientific AI for game playing, namely the maximization of ply-lookaheads. E.g., the most prominent chess solvers [16] came to fame for their ability to calculate a huge number of plies (turns) beforehand and thus exceeded the computational capability of a human player. This however supposes that the human opponent plays optimal (and furthermore, that every strategy that deviates from this estimated optimal one would be defeated anyhow). Since those chess programs almost exclusively base on this assumption, competing programs do not take a risk. Still, coming back to the card game opponent, these assumptions of optimality can become fallacious.

Example

Player A's hand consists of (ID:9)(3/3), (ID:8)(3/2) and (ID:3)(1/1).
Player B's hand consists of (ID:8)(3/2), (ID:6)(2/2) and (ID:3)(1/1).
Player A is considering, which card to play:
- If A plays (ID:9)(3/3) and B (ID:3)(1/1), A wins this turn.
- If A plays (ID:9)(3/3) and B (ID:6)(2/2), A wins this turn.
- If A plays (ID:9)(3/3) and B (ID:8)(3/2), the turn is tied.

- If A plays (ID:8)(3/2) and B (ID:3)(1/1), A wins this turn.
- If A plays (ID:8)(3/2) and B (ID:6)(2/2), the turn is tied.
- If A plays (ID:8)(3/2) and B (ID:8)(3/2), the turn is tied.

- If A plays (ID:3)(1/1) and B (ID:3)(1/1), the turn is tied.
- If A plays (ID:3)(1/1) and B (ID:6)(2/2), B wins this turn.
- If A plays (ID:3)(1/1) and B (ID:8)(3/2), B wins this turn.

The optimal choice would thus be (ID:9)(3/3), since it wins 2/3 cases.
Nevertheless, if player B knows (or estimates) exactly this, B can play (ID:8)(3/2) and the turn is tied.
After that, the following situation remains:
- If A plays (ID:8)(3/2) and B (ID:3)(1/1), A wins this turn.
- If A plays (ID:8)(3/2) and B (ID:6)(2/2), the turn is tied.

- If A plays (ID:3)(1/1) and B (ID:3)(1/1), the turn is tied.
- If A plays (ID:3)(1/1) and B (ID:6)(2/2), B wins this turn.

The optimal choice would thus be (ID:8)(3/2) since it wins 1/2 cases.
If player B knows (or estimates) this, B can play (ID:6)(2/2) and the turn is tied.
After that, the only solution is:
- A plays (ID:3)(1/1) and B plays (ID:3)(1/1), the turn and the whole game is tied.

This example however neglects the fact that player A could also make an estimation about player B (and based on this, decide to choose another strategy than the optimal). In this case, player A would not waste (ID:9), but e.g.:

- Player A plays (ID:8)(3/2), just as B (ID:8)(3/2), the turn is tied.
After that, the following situation remains:
- If A plays (ID:9)(3/3) and B (ID:6)(2/2), A wins this turn.
- If A plays (ID:9)(3/3) and B (ID:3)(1/1), A wins this turn.

- If A plays (ID:3)(1/1) and B (ID:6)(2/2), B wins this turn.
- If A plays (ID:3)(1/1) and B (ID:3)(1/1), the turn is tied.

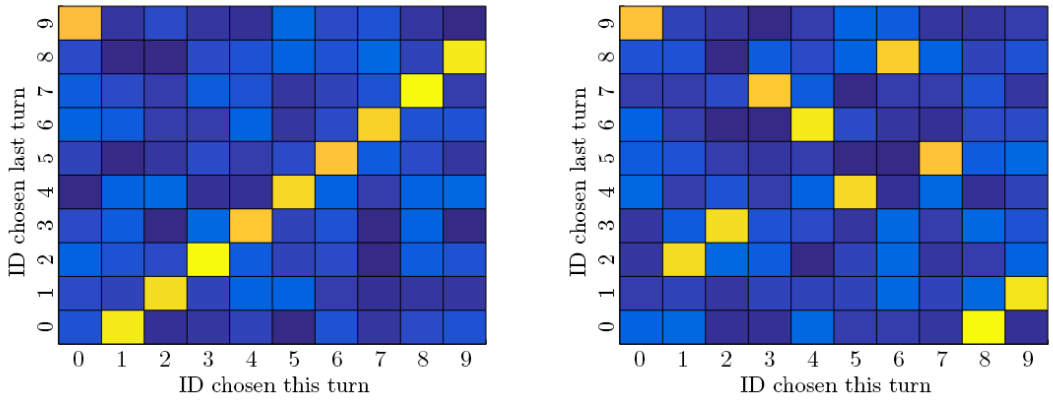
Again, if A would play optimal, he would go for (ID:9)(3/3) to win the turn no matter what - but if B knows (or estimates) that, B would sacrifice (ID:3)(1/1) to win the last battle with his (ID:6)(2/2) against A's (ID:3)(1/1).
If A knows (or estimates) this sacrifice move, he could as well spare the certain win (ID:9)(3/3) and use his remaining (ID:3)(1/1) as well. This leads to a tie for the second turn, but in the last one, A would dominate with (ID:9)(3/3) over (ID:6)(2/2), which declares player A to the winner of the whole game.
The addressed problem thus lies recursively in the estimation of both players: Would B know that A doesn't play optimal, neither in the first, nor the second turn, but actually wants to trick B, he could act in advance and turn the game as following:
1. Turn: A plays (ID:8)(3/2), B plays (ID:6)(2/2), tie.
2. Turn: A plays (ID:3)(1/1), B plays (ID:3)(1/1), tie.
3. Turn: A plays (ID:9)(3/3), B plays (ID:8)(3/2), tie.
Due to the fact that player A has better cards on his hand, player B can never win in this example, but at least he can prevent player A from doing so. The most interesting part about this problem is probably its inductive recursiveness:
If B knows what A plays, the game is a tie.
If A knows that and tricks B, A wins.
If B knows that A tries to trick B, the game is a tie.
If A knows that B knows that A tries to trick B, A wins.
If B knows that A knows that B knows that A tries to trick B, the game is a tie.

Thus, this example (which does not stop at this point but would go on ad infinitum given unlimited numers of ply-lookaheads) shows that there definitely are situations where no optimal moves or strategies exist and/or the assumption of an optimal playing opponent leads to a non-optimal result. This indicates that it is not always reasonable to assume optimality in the opponent, but to prefer the prediction of a learned estimate that models the personal preferences in the context of all related parameters.

Therefore, in setup 3 not only the utility of a card in a given situation is modeled, but first a previous estimation of the opponent's choice is approached, since without an idea about the opposing *Unit*, the playing of any *Unit* could only be based on loose heuristic strategies.

3.3.1 Variant 1: Reinforcement Learning

The utilized RL mechanics in the previous chapters were successful in modeling reward relations between different *Unit* entities, but formulated as a time series prediction problem, they are also capable of giving an estimation about a subsequent turn, given that knowledge about the preceding is provided. Theoretically it is possible to include further knowledge about more preceding choices, but in this approach the number of information included increases the size of the RL map exponentially, so for a prediction using one choice information a 10x10-map is sufficient to represent the information, including two preceding turns however would result in a 10x10x10 map, etc. Even for this simple setup, the information space required to fill would reach infeasibility very quickly, since much time and iterations would be needed. Thus, this RL approach follows a memoryless principle which is adequate for simple purposes or approximations.

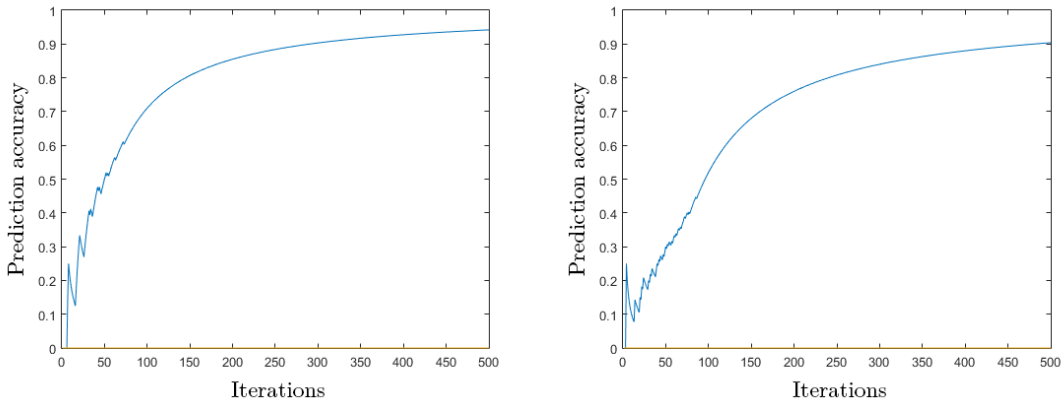


(a) RL map

(b) RL map1

Figure 3.19: RL maps of Setup 3, variant 1

Examining this method again from the bottom up, the first task is to model completely deterministic strategies that do not include contextual integration, i.e. the opponent obstinately plays *Units* according to a strict ordering, like these:
(a) [0 1 2 3 4 5 6 7 8 9]
(b) Random permutation of [0 1 2 3 4 5 6 7 8 9]



(a) RL map (b) RL map1

Figure 3.20: Prediction accuracy of Setup 3, variant 1

As seen in Figure 3.20, no structural performance difference is recognizable, as soon as the deterministic match is found (about $\tilde{1}00$ iterations in a setup of 10 possible preceding and 10 possible following turns), the prediction converges to 100%, as the particular map visualizations (Figure 3.19) also highly suggests.

After all, Time Series Prediction can not only be used to forecast particular single states, but coupled to a classifier module that integrates many dimensions of parameters, it can estimate that an action from a certain action class is likely chosen. Assuming that the player that has to be modeled acts similarly if exposed to the same or a very similar situation, and that this mapping from situation to reaction can be modeled, this approach would display certain preferences and disfavors of the player. Thus, at least to a certain probability, the prediction can be made, which reaction follows which situation. E.g., phase 1 of any turn provides only the option to draw a card, so the likelihood for the model to choose a different outcome of any situation in which the input "phase" is "1" should converge to 0. Given enough iterations of the modeling process (which is why *Korona: Tinker's Curse* is a suitable environment, collecting many clear defined actions, turns and

rounds in few time), not only this trivial connection should be possible to show, but also more complex, strategy-driven preferences. The setup described in the last paragraph is thus implicitly included in the next chapter: The cognitive agent - Deep Reinforcement Learning in *Korona: Tinker's Curse*.

3.4 Integration of the complete card space

Finally, to bridge the path to the next chapter, one of the most constraining assumptions used in the previous setups is removed so that all following models now have to deal with:
- Full card space (The set of cards is no longer restricted)

That means, the actual confrontation of two *Units* has to be assessed, which is not trivially calculable from the *Attack* and *Defense* value, but has to include the manifold number of effects that they may possess. Thus, an integration into the actual game of *Tinker's Curse* is inevitable. The video: *Korona: Tinker's Curse*. Permutational confrontation of all ingame Units [45]) documents how every *Unit*-vs-*Unit* battle is simulated and recorded, which leads to the following result, using techniques and insights gathered from the earlier sections.

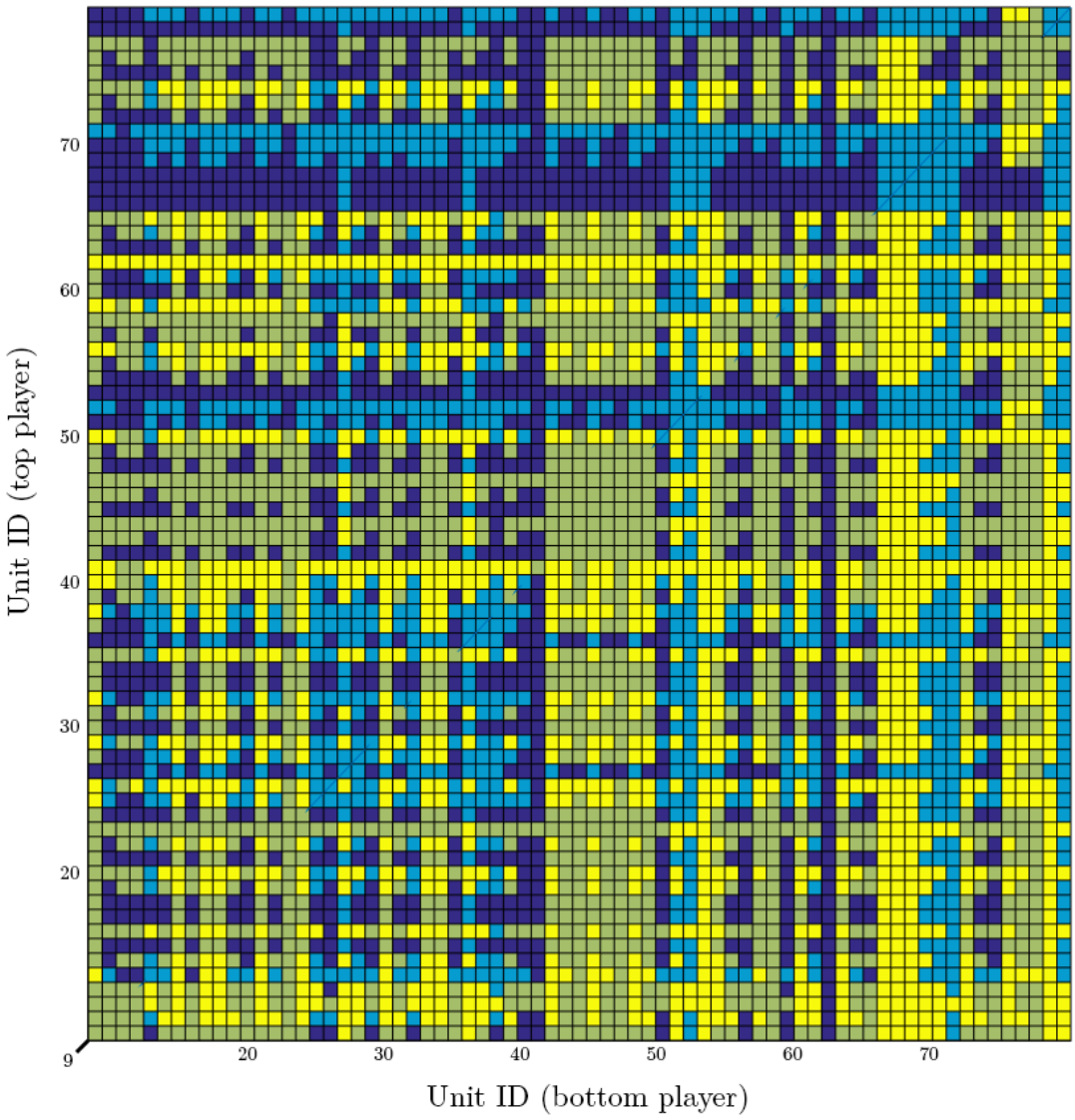


Figure 3.21: Result of every *Unit*-vs-*Unit* permutation

Figure 3.21 demonstrates the learned outcome of all possible configurations of actual in-game *Units*. Yellow fields show a dominating top *Unit* (e.g. row 62 shows that (ID:62)(8/8) is superior to every other *Unit* and can at most be tied), green fields indicate an offensive tie (both *Units* die), light blue ones a defensive tie (no *Unit* dies) and dark blue ones a dominating bottom player.

Although including the actual *Units* with all their effects, further dimensions are completely ignored: the amount of hand cards of both players, the amount of energies, cards in the decks, etc., which can make an influence on the outcome of the battle (e.g. (ID:15) is highly dependent on hand cards, which is why it loses most of the matches in this overview). Above that, certain *Units* are not only used to win battles, but serve other purposes (bring additional energy into play, draw additional cards, deal direct damage, etc.) or are specifically designed to be defeated in battle to trigger certain effects (e.g. some of the violet *Units*).
After all, this model of the complete confrontation space is useful to estimate the outcome of a battle with a high probability, and above that the general utility of *Units* can be approximated by accumulating the positive outcomes in the respective row (or column). To describe an agent that is totally aware of the environmental context and includes this in his decisions however, this approach is not enough. If these dimensions are included into the map shown in Figure 3.21, the already big record gains infeasible complexity - which is, why the following chapter eventually presents the final approach of the cognitive agent.

Chapter 4

The cognitive agent - Deep Reinforcement Learning in *Korona: Tinker's Curse*

Stepping out of the testing environment, this section will deal with the actual implementation of an intelligent agent into the fully functional game setup *Tinker's Curse*. In this, whole game processes are recorded (*chunks*) and every turn of the (human or heuristic) player is logged and interpreted, on the basis of: Complete state space description of Korona: Tinker's Curse at a single point of time. This data is used to develop a predictive classification of the player's next action, given the state that he is in. With this knowledge, the cognitive agent is then able to select an action out of many, according to the particular estimated reward that is learned assigned to these state-action pairs.

4.1 Deep Reinforcement Learning

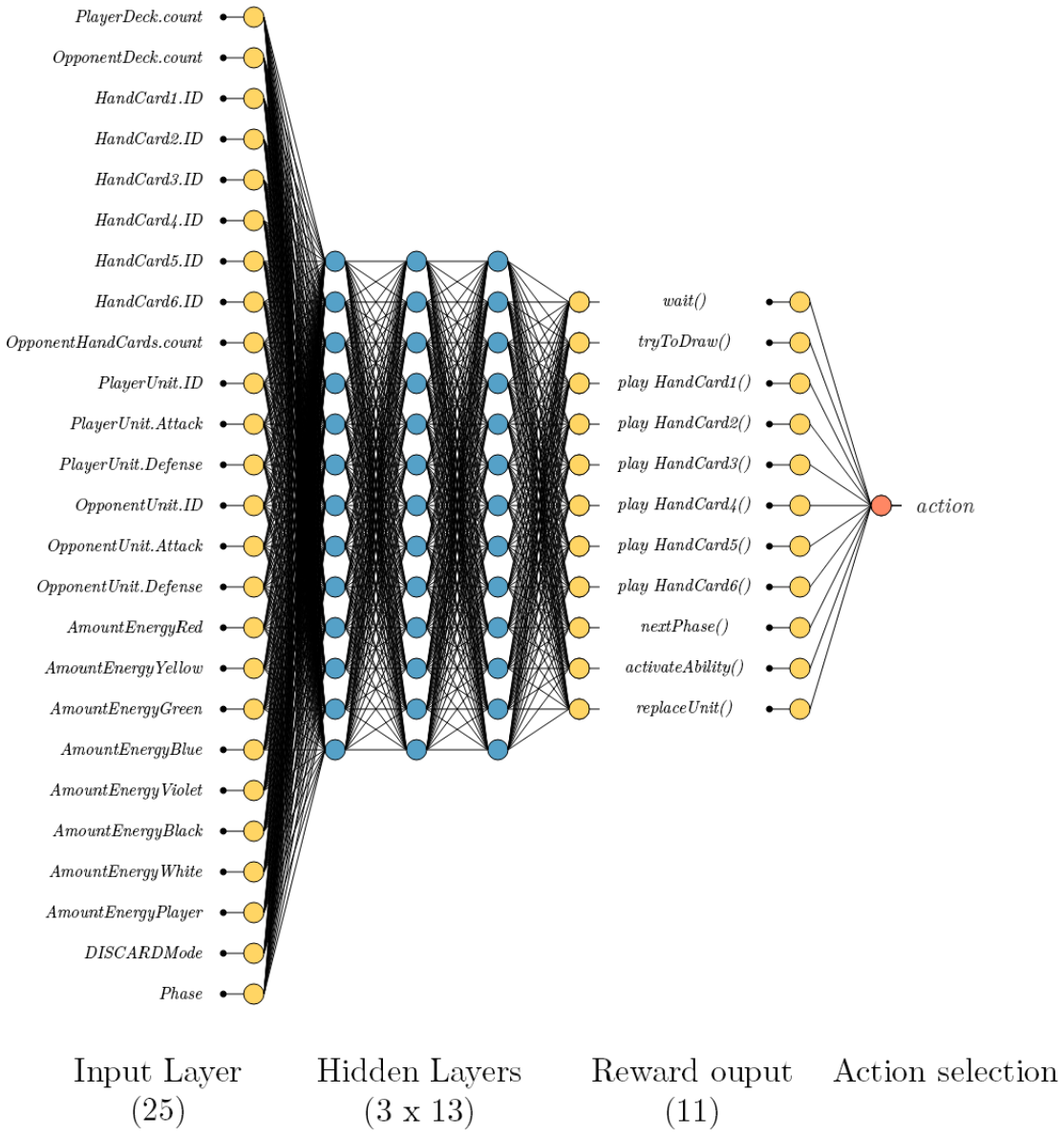


Figure 4.1: Setup of the DeepRL network

The core module for intelligent action selection is displayed in Figure 4.1. The whole contextual information is gathered and represented in the network which assesses the situation by learning from similar game state constellations and evaluates the possible actions by means of learned rewards. These rewards however have to be delivered somehow to estimate whether a choice was valuable or not. In the case of *Korona: Tinker's Curse*, the most important factors of the compound reward r are describable by the sum of:

$$r = \left\{ \begin{array}{ll} +\alpha & (\textit{opposing Unit is defeated by this choice}) \\ +\beta & (\textit{own Unit stays alive after this choice}) \\ -\gamma & (\textit{own Unit is defeated by this choice}) \\ -\delta & (\textit{opposing Unit stays alive after choice}) \\ +\epsilon & (\textit{game is won after this choice}) \\ -\zeta & (\textit{game is lost after this choice}) \\ -\lambda & (\textit{idiosyncratic action, since rule violation}) \\ +\eta & (\textit{damage dealt to opponent}) \\ -\theta & (\textit{damage received}) \end{array} \right\}$$

Figure 4.2: Composition of the total reward r in every choice of *Tinker's Curse*

These values do not only depict the learning intensity, but can also represent biases of various agents to different meta strategies. E.g., a balanced agent could be configured with:

$$\begin{array}{llllll} \alpha & = 10 & \beta & = 10 & \gamma & = 10 \\ \delta & = 10 & \epsilon & = 100 & \zeta & = 100 \\ \lambda & = 5 & \eta & = 10 & \theta & = 10 \end{array}$$

The former agent considers both defeating the enemy's *Unit* and keeping the own one equally important, which is not necessarily the case, since a different agent's reward configuration could look as the following:

$$\begin{array}{llllll} \alpha & = 30 & \beta & = 5 & \gamma & = 5 \\ \delta & = 30 & \epsilon & = 100 & \zeta & = 100 \\ \lambda & = 5 & \eta & = 10 & \theta & = 10 \end{array}$$

In this constellation, the second agent receives much more positive feedback in situations where the opposing *Unit* dies because of his choice, and more negative feedback in those where it stays alive. This definitively describes a more offensive type of playing, not, because the logic of the agent was changed, other components have been used or hard-code constraints are defined that *Units* necessarily have to die, but because this agent obtains more satisfaction (in a technically transferred sense) that other agents would not necessarily perceive. Thus, these configurations define an agent's personality, without a direct influence on its performance. Hence it is further referred to as the agent's *character*, in analogy to the human character that also defines and explains, which decisions human prefer in which situations and how they estimate their personal advantages and disadvantages in these. Building on this, the performance of various *characters* could be learned iteratively by series of comparisons that integrate automatically learned and adapted *character* configurations, which is one of the outlooks of this thesis.

4.2 Performance tests

Since having a contrasting agent to compare with proved to be a good idea in chapter 3, the first agent that challenges the heuristic model is a random player. The disadvantage of the deep approach lies in the non-readability of its complex, hidden structures, which makes the debugging and tracking of the learning process harder. However, this section documents the recording of a number of parameters that shows the development of the agent over time and game iterations, which is in each round constituted by:

n_m	The total number of moves
n_im	The number of idiosyncratic moves (A player without contextual comprehension will often try to make moves that violate the rules of the game, where a constantly learning agent should minimize these moves to 0)
p_im	Percentage of idiosyncratic moves
win	The total average of games the agent won
tie	The total average of games the agent tied
loss	The total average of games the agent lost
n_ul	The number of *Units* lost in battle
n_ud	The number of *Units* defeated in battle
deck	The amount of cards in the agents deck minus the opponent's deck (Measured at the end of the game, thus positive in won, negative in lost games. This gives a hint of how good or bad the win/loss was)
n_turns	The number of turns of the played game
t_turn	The time needed to finish a turn
t_round	The time needed to finish a complete game

4.2.1 Untrained Player

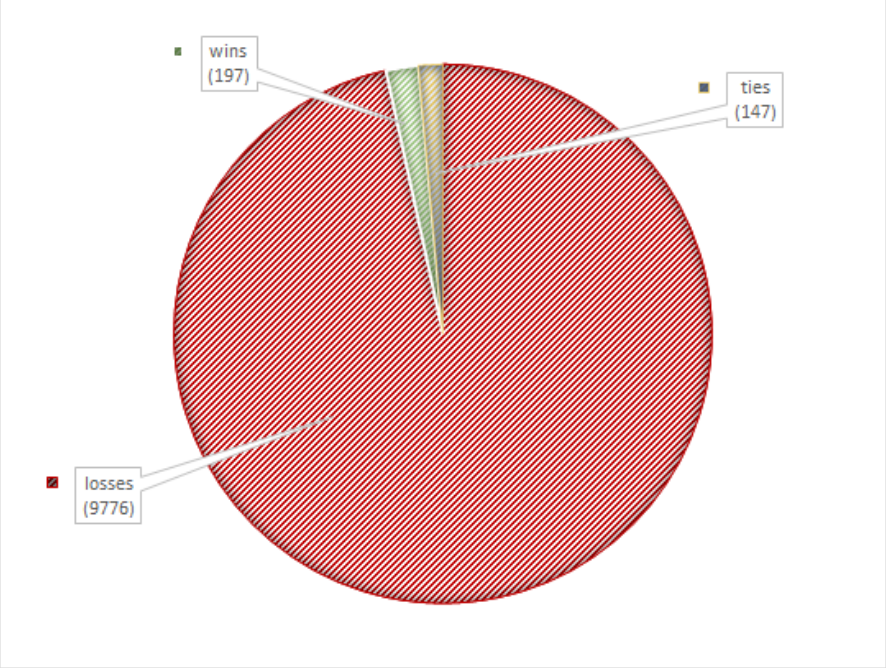


Figure 4.3: Untrained Player: Wins vs. Ties vs. Losses

As shown in the first recorded gameplay Video [44], a completely untrained agent without predictive capabilities or contextual integration competes against the Advanced Heuristic Model that was introduced in chapter 2.3.
The first thing that comes into mind looking at the logged statistics of over 10.000 chunks is the really bad performance overall, as Figure 4.3 depicts: 9776 lost games, where the remaining ones are equally probable tied (147) or won (197). However, this result is not a surprise since a completely clueless agent without an understanding of the game or learning process has to challenge a highly performant, strategic opponent with a great amount of human-made fine tunings over many previous trials. The 3% of the games that are not lost come from the variance of the game, even if the advanced heuristic model is able to choose optimal decisions, it is sometimes confronted with unfortunate situations (e.g., if not enough *Energy* is available to play the *Units* on hand, wrong *Units* are drawn from the deck which are constantly countered by the opposing *Unit* or instead of drawing *Units* after all, too much *Energy* is received - where the totally random player accidentally chooses correct moves.

Figure 4.4 demonstrates that of these chosen moves, the untrained player tries to make on average 35.3% idiosyncratic ones, i.e. more than every third action is against the rules or not applicable to the situation - which does not state that the remaining moves are optimal. E.g., in phase 3 (Play *Unit*), the attempt of playing an *Energy* would violate the rules, skipping this phase would not be. However, skipping phase 3 in the case of not having a *Unit* is a bad idea in most of the cases if other actions are available, since the opposing *Unit* could attack the player's deck directly.

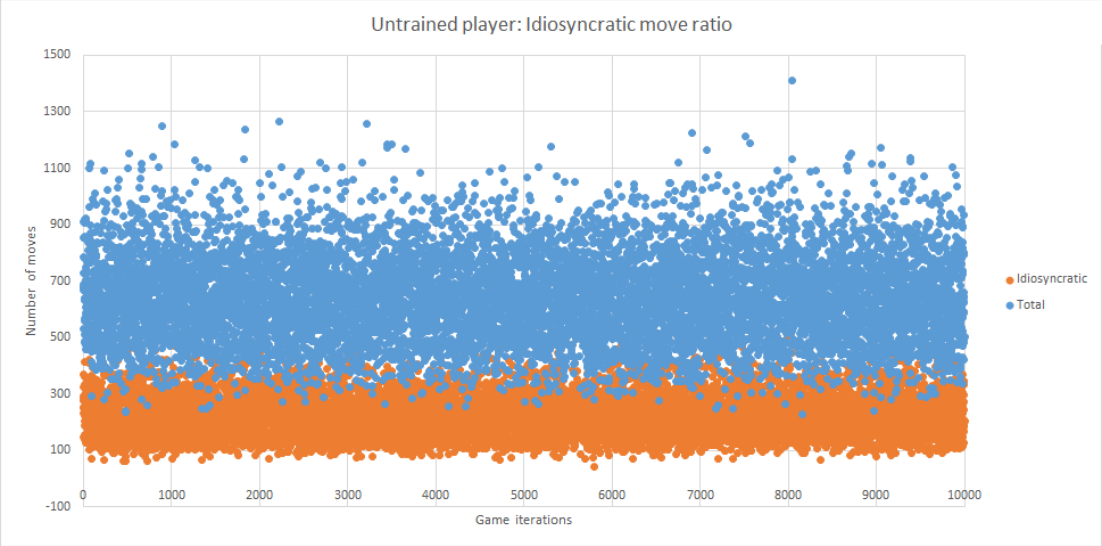


Figure 4.4: Untrained Player: Idiosyncratic move ratio

When it comes to the actual battle, even the random player succeeds in defeating some *Units*. Figure 4.5 records the *Units* defeated by the player minus the ones that he lost in battle and thus reflects the amount of "winning" single turns for him. Even if the untrained agent is able to succeed in several *Unit* confrontations, this does not necessarily end up in a total win for him, since a great proportion of the *Units* are not only made to survive battles or defeat opponent *Units*, but are valuable in their effects (certain yellow *Units* cannot really win battles, but bring advantages to owner's *Energy* or prevent battles, certain blue *Units* deal their damage directly to the opponent and are never able to win a battle, certain violet *Units* are especially designed to die in a fight to trigger certain effects, etc). Thus, the distribution in Figure 4.5 looks pretty balanced with an average of -0.30 (i.e., on average, the untrained player still loses more *Units* than the advanced model), but is in fact not that informative as Figure 4.6 for the actual performance.

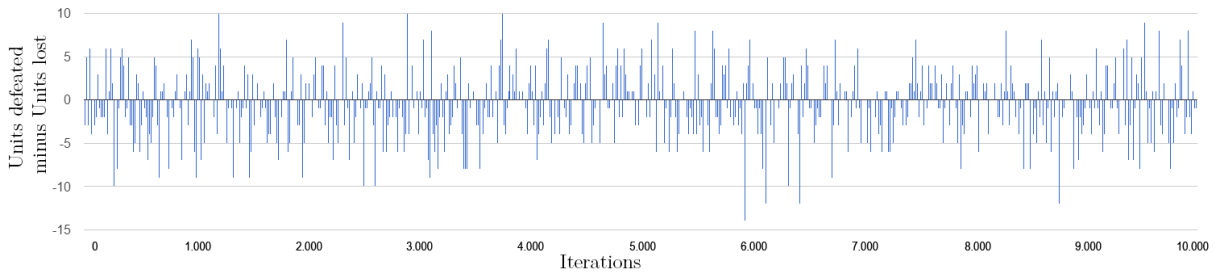


Figure 4.5: Untrained Player: Units defeated - Units lost

In Figure 4.6, the final situation of each game is displayed, quantified by the amount of cards left in the agent's (positive) or opponent's (negative) deck. This demonstrates drastically that the losses of the randomly playing agent are not marginal, but severe (on average -14.1, i.e. the opponent wins on average with 14.1 cards left in his deck). After a mean turn amount of 17.5 per game, 22.5 (1 per turn plus 5 initial) cards are drawn regularly from the total 40 cards, that means that the random player manages to deal 3.4 damage in a complete game, whereby the advanced heuristic opponent deals 17.5 damage to the player. The few positive spikes in Figure 4.6 show the rare winning situations which have a far smaller magnitude than the negative records, since the only winning situations are very close runs accomplished by accidentally fortunate game constellations.

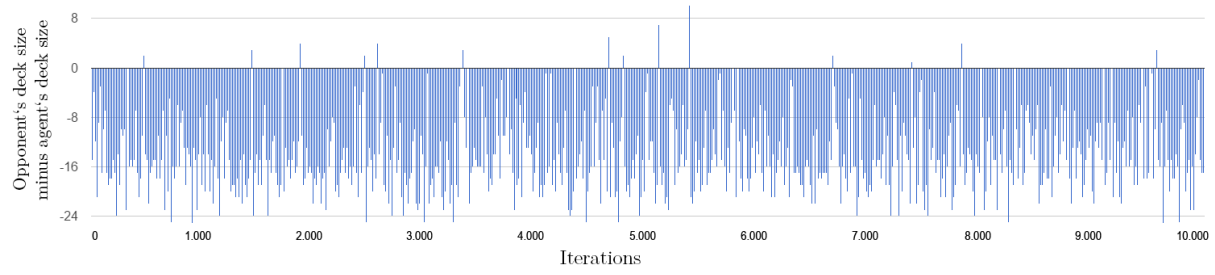


Figure 4.6: Untrained Player: Opponent's - Agent's Deck size (at the end of the game)

For comparison, the time taken for a single turn were 0.16 seconds and 2.78 seconds per *chunk* on average.

4.2.2 Learning player

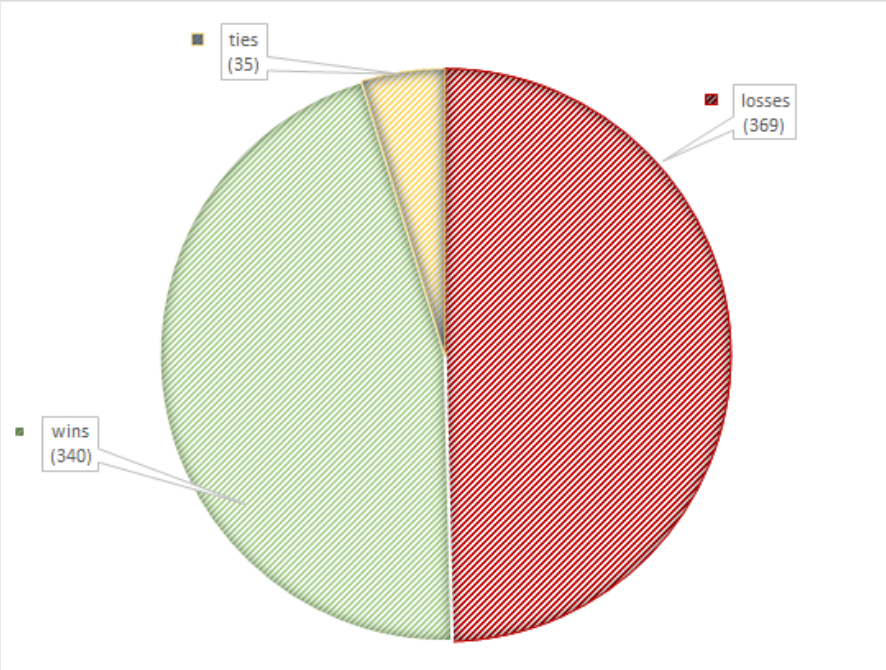


Figure 4.7: Learning Player: Wins vs. Ties vs. Losses

Following the neural architecture of Figure 4.1, the learning player already gathers insights about the outcome of his choices in the first game iterations. Due to temporal restrictions, only 745 chunks were played and recorded, since the learning process after each iteration required a significant amount of time (up to 20s), compared to the fully automated playing procedure (2.78s for the untrained player, 16.9s for the learning player). However, even in this amount of time, the agent learned to approximate the performance of his opponent, as seen in Figure 4.7: 45.6% of the games are won, 4.7% tied and 49.5% lost. The fact that he is still outperformed slightly by the advanced heuristic model does not harm the impact of these results, since the desired outcome was not a total domination, but a dead even opponent, which made a big performance leap since the last introduced model.

This becomes even clearer in Figure 4.8, not only because apparently a huge number of winning spikes is visible (compared to the former agent in Figure 4.6), but also because the magnitude of wins or losses is nearly the same (on average, +7.78 cards for wins and -10.1 cards for losses), which indicates that both players are winning successfully once their strategy leads and are not only amplitudes of the game's variation that managed to decide the outcome very briefly.

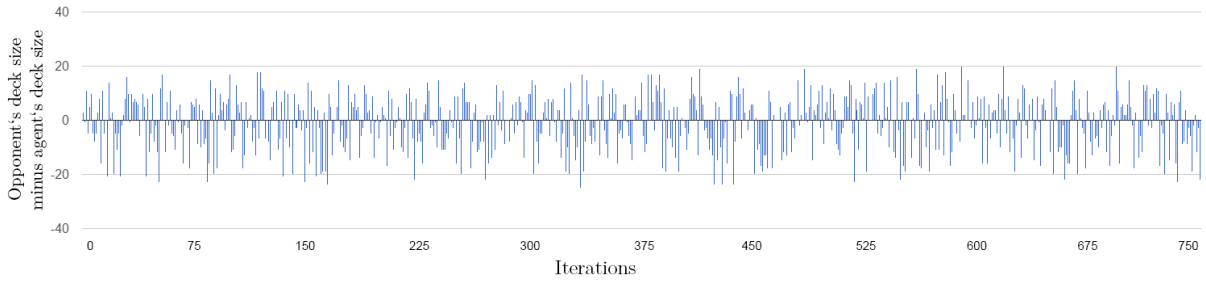


Figure 4.8: Learning Player: Opponent's - Agent's Deck size (at the end of the game)

The only factor that is not satisfying in the learning agent is the ratio of idiosyncratic moves, which is still a third of the total number of moves (Figure 4.9). This depicts the behavior of the agent in that he is able to discriminate which moves and choices are actually preferable to random moves, but in cases with no significant clue, still too many unjustified moves are executed. Since these occurrences critically harm the perceived intelligence of the agent, the architecture is reconsidered and modulated in the next chapter.

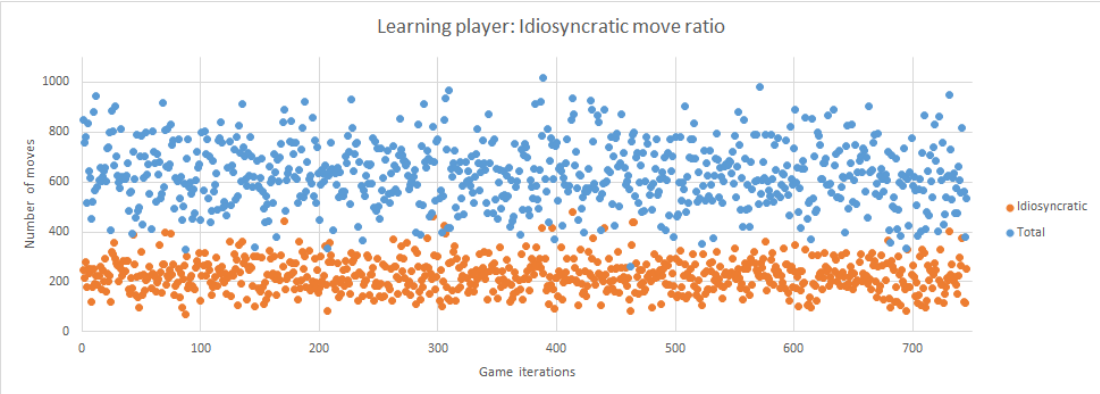


Figure 4.9: Learning Player: Idiosyncratic move ratio

4.2.3 Modularly learning player

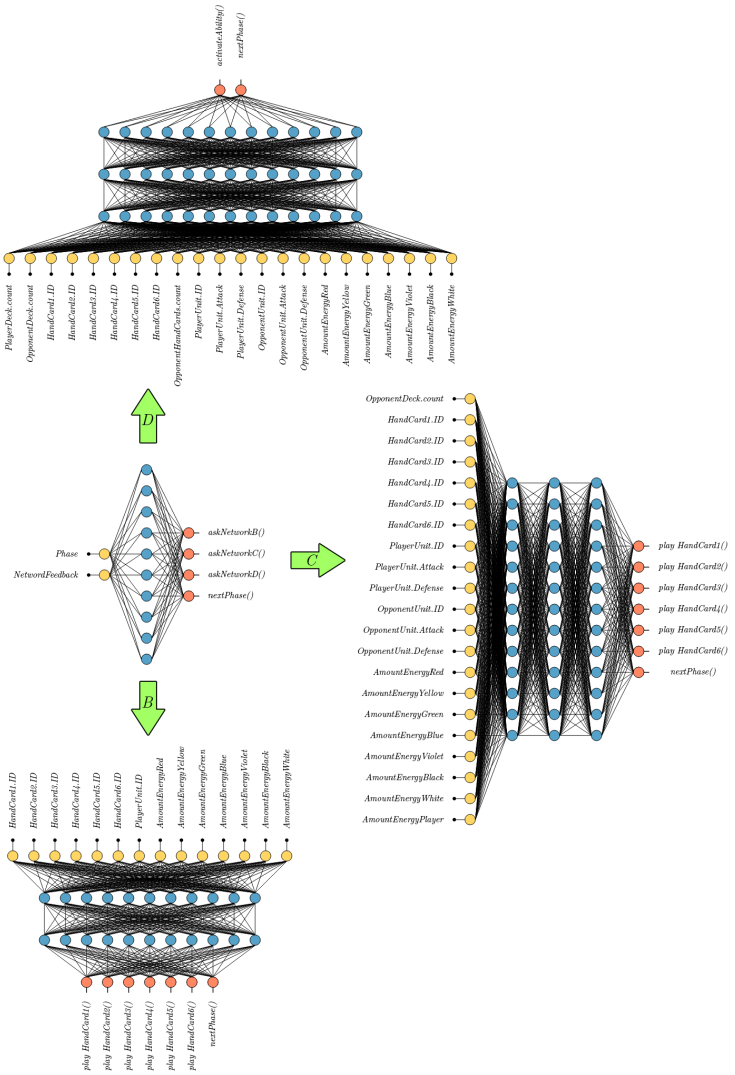


Figure 4.10: Setup of the modular DeepRL *(details: 4.11 - 4.14)*

In contrast to the previous, every possible action including architecture, this approach constitutes several assessments of the game for various situations by using a number of networks that learn different aspects, in analogy to the constellation of several brain regions responsible for different tasks that come up with a compound solution.

Network A: Management
(Decides, if an action should be executed in the given situation, otherwise the phase proceeds. Passes the actual action selection task further to the specific network.)

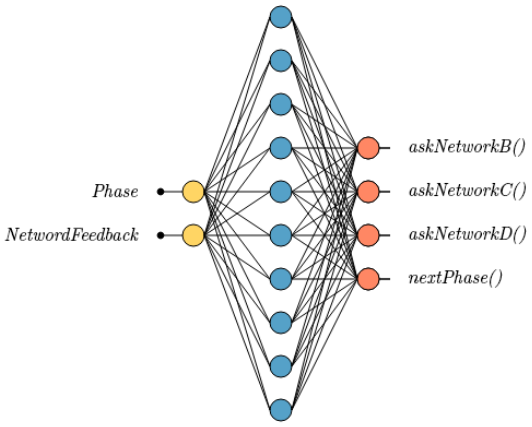


Figure 4.11: Network module A

To keep it as sparse as possible, this network only gets input of the current phase, the main decision criterion for the targeted selection, and the feedback from the particular chosen network, to decide if it makes sense to do additional actions before proceeding (e.g. certain abilities can be used multiple times if the required costs can be paid). It then approximates the learned reward r for the selection with an increased λ (idiocracy punishment) to prevent rule-violating attempts even further.

Network B: Energy
(Decides, if it makes sense to play an *Energy* or not.)
As shown in Figure 4.12, network B essentially gets information about the options to choose from (hand card 1-6), the existing amount of *Energy* on the agent's side, as well as the controlled *Unit* that may benefit from having more *Energy* or more hand cards, respectively. Since the reward is put into the network with a delay, it includes the playing of *Units*, meaning that not only the mapping between available *Energy* within the hand cards to playing them is learned, but also the relationship between the *Units'* IDs on hand to their *Energy* costs (i.e., if the network decides to play an *Energy* that actually enables the playing of a *Unit*, this choice will yield more reward than if this *Unit* would not be playable after the action).

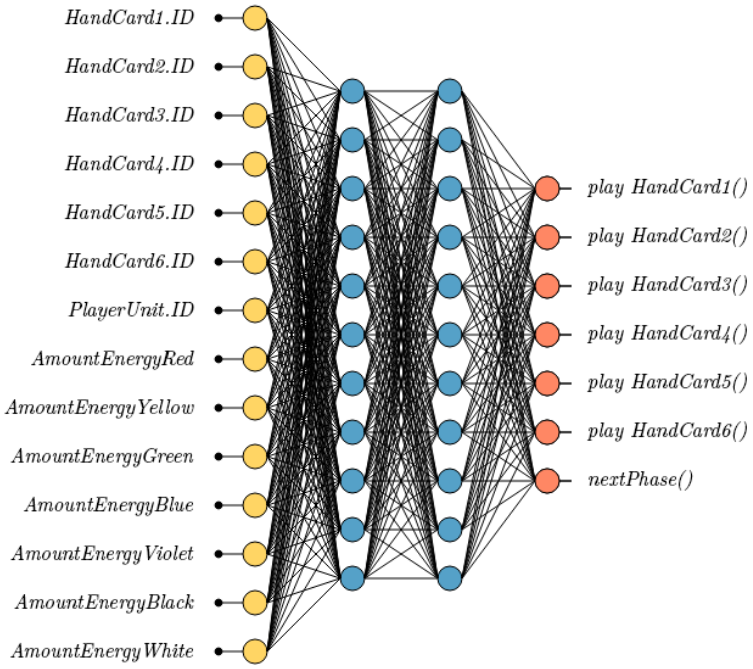


Figure 4.12: Network module B

Network C: Unit

(Decides, which of the available *Units* to play.)

For this decision, not only the previous factors have to be included, but also, if a *Unit* already exists on the agent's side, how strong it is (in terms of *Attack* and *Defense*) and if the opponent has already a *Unit* on the board, since defeating this would definitively a reward, as well as preventing damage do the own deck would. If not, *Units* that deal higher damage to the opponent's deck will also result in an increased r, depending on the agent's *character*.

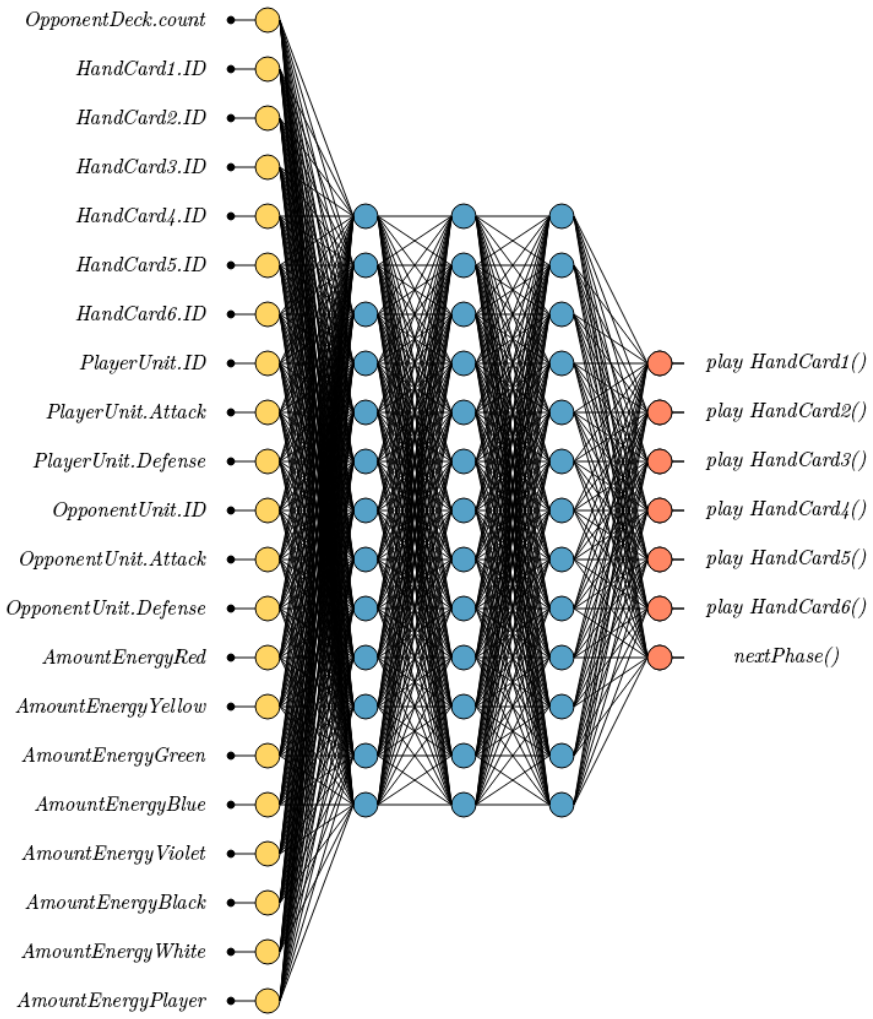


Figure 4.13: Network module C

Network D: Ability
(Decides, if it is useful to activate the ability of the present *Unit*.)

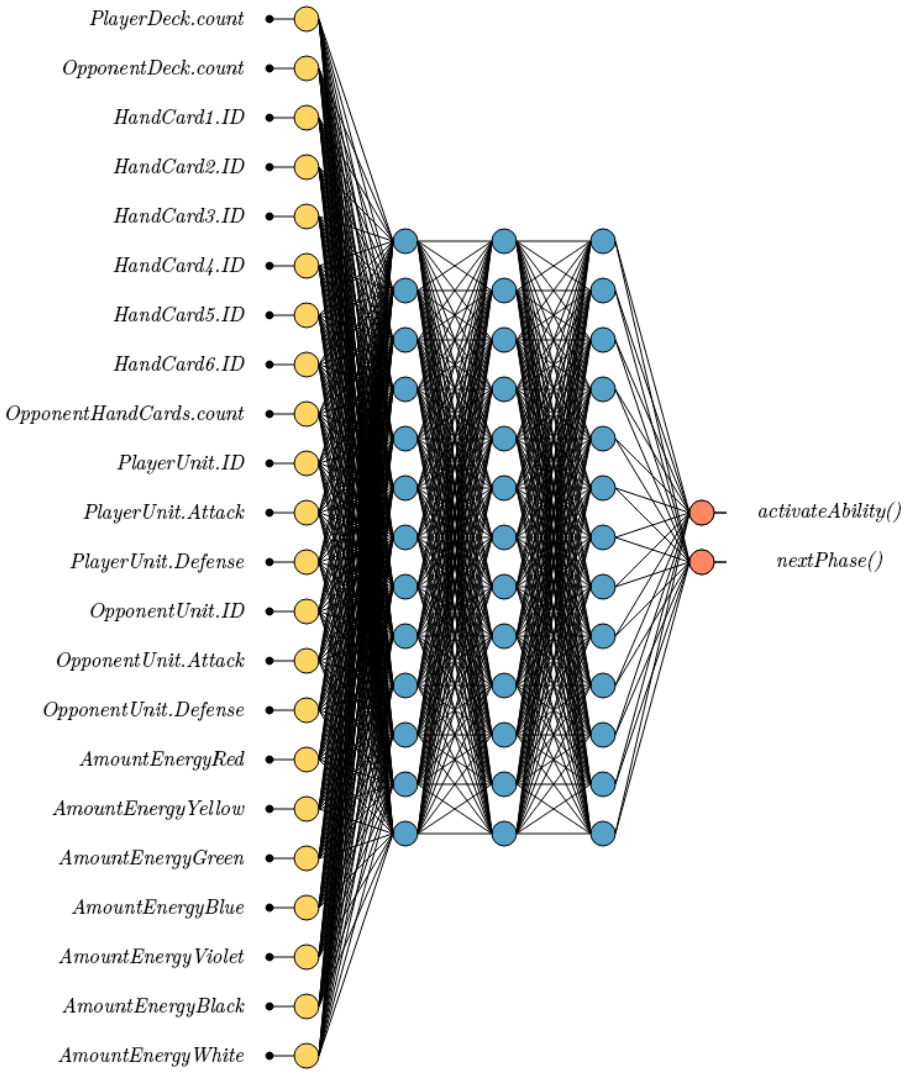


Figure 4.14: Network module D

Finally, the choice of activating the ability of an existing *Unit* requires the most contextual information, since most abilities differ drastically from each other and have to be deployed in the correct context to not waste the resources required to pay their costs.

Results

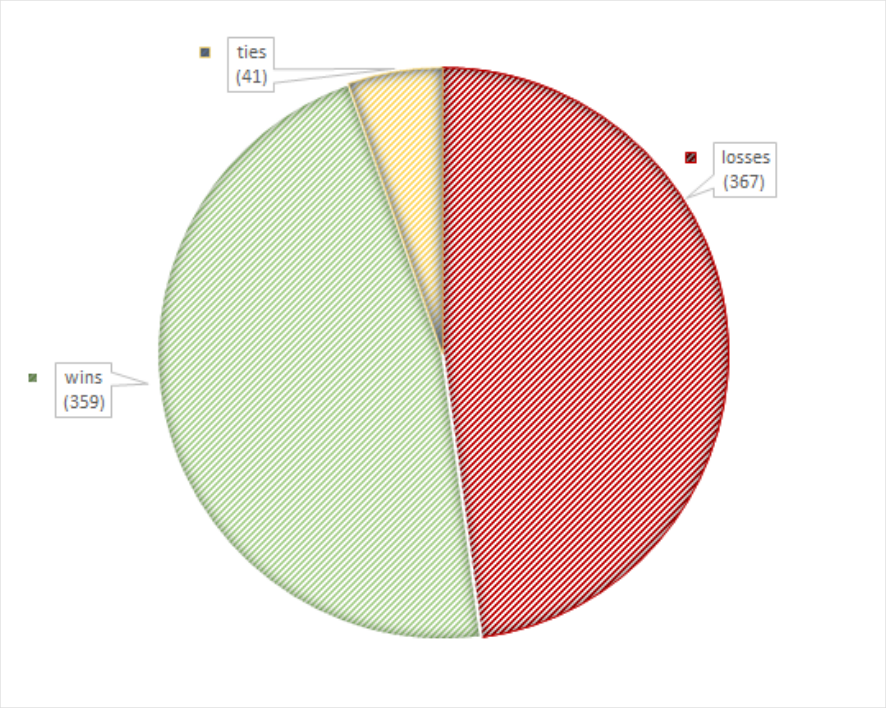


Figure 4.15: Modularly Learning Player: Wins vs. Ties vs. Losses

As in the model before, the agent that is learning modularly can approximately win half of the games, establishing the desired *not too hard, not too easy* player. The visible learning process in contrast to the former approach appears in the ratio of idiosyncratic moves (Figure 4.16). Since not only one network is responsible for making all decisions, rule-violating moves can be recognized and punished earlier, i.e. the managing network in the first step already learns that choosing inappropriate succeeding networks leads to no change of the game state at all and thus it rules out these options systematically.

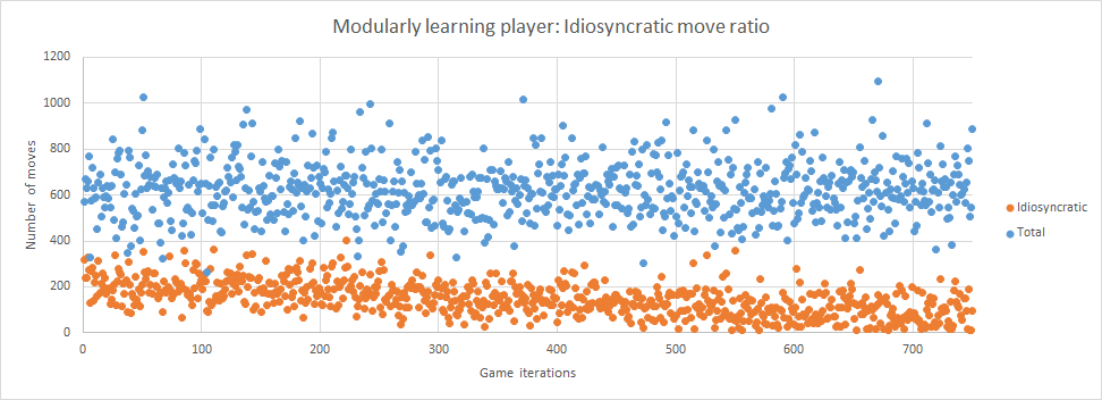


Figure 4.16: Modularly Learning Player: Idiosyncratic move ratio

While in the first 100 iterations, the number of idiosyncratic moves ranges from 67 to 360 (with 206.4 on average), the last 100 of these 750 iterations only contain 10 to 246 idiosyncratic moves, with 91.4 on average. This result is not perfect, but it definitely shows a significant trend in learning which evidently needs a large number of iterations to cover the vast amount of possible game states and all plausible actions.

After all, the application of deep reinforcement learning proved to be successful in performance, adaptability and credibility in a reasonable amount of time. First of all, *Korona* will be shipped with a fully functional implementation of deep learning in video games, demonstrating the successful implementation of academic AI that is rarely found beyond pathfinding up to now and secondly, the technology developed in this thesis will be made available publicly through the use of the user-friendly interface introduced in the following chapter.

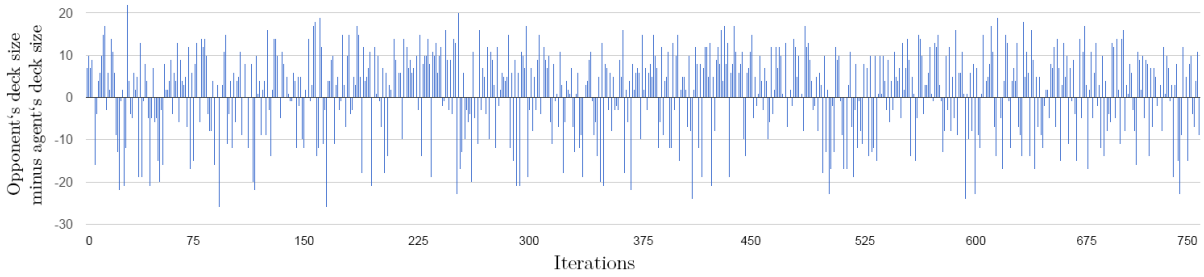


Figure 4.17: Modularly Learning Player: Opponent's - Agent's Deck size

Chapter 5

The Challenge Yourself plugin

In order to come back to the proposed framework, sustain adaptability and spread the idea of intelligent agents through academic AI, the core modules of the thesis are packaged into the *Challenge yourself plugin*. Due to the fact that the implementation of chapter 4 is completely written in C# and thus already adapted to Unity3D (the currently most dominant game development engine), the integration into other games can be realized very fast and extremely easy.

To get access to the introduced techniques, the plugin only has to be imported into the particular resource folder, since all required scripts and libraries are included. From that on, the desired modules can be applied to agents, NPCs or any other *GameObject* via simple *drag & drop* without quantitative constraints, while every single instance keeps distinctly modifiable and adjustable. A quick preview is displayed in the *Component* of the respective *GameObject* to show the most important parameters as well as the kind of used module, as seen in Figure 5.1: The prefab(1) (i.e. the supplied module object) is dragged directly to the desired *GameObject*(2) (e.g. an NPC), where (3) shows the preview of the module.

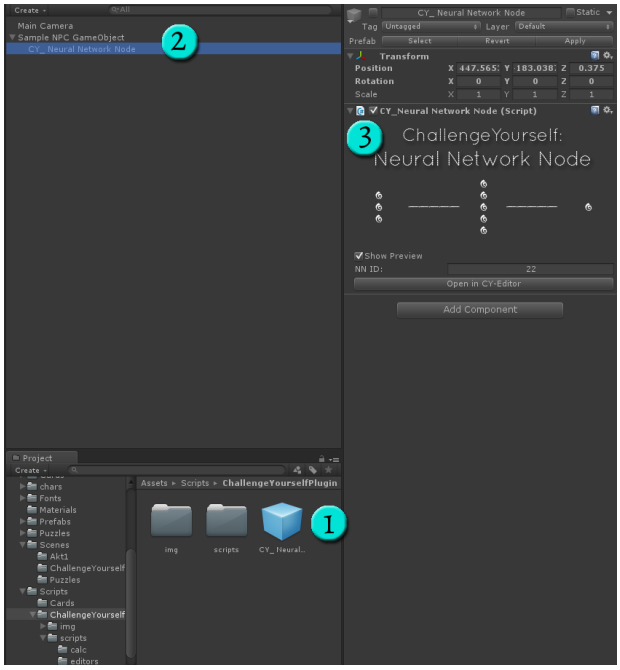


Figure 5.1: Importing a *Challenge Yourself* module to a new game

For the settings of the particular object instance, an own editor window has been designed that lets the developer have full control over the adaptable parameters (Figure 5.2). This editor does not only deliver a visualization and configuration parameters, but also displays statistical values (in this case, the number of calls or trainings and the resulting performance of the network), as well as a comprehensive listing of every function that can be called on the module while playing.

After the publication of this thesis, the *Challenge yourself plugin* will be released in the *Unity Asset Store* where those modules are distributed to a large community of developers. This will not only propagate the usage of academic AI in video games, but also establish a base setup which will be extended by more and more AI modules in the future, while using usability and performance feedback from a whole community.

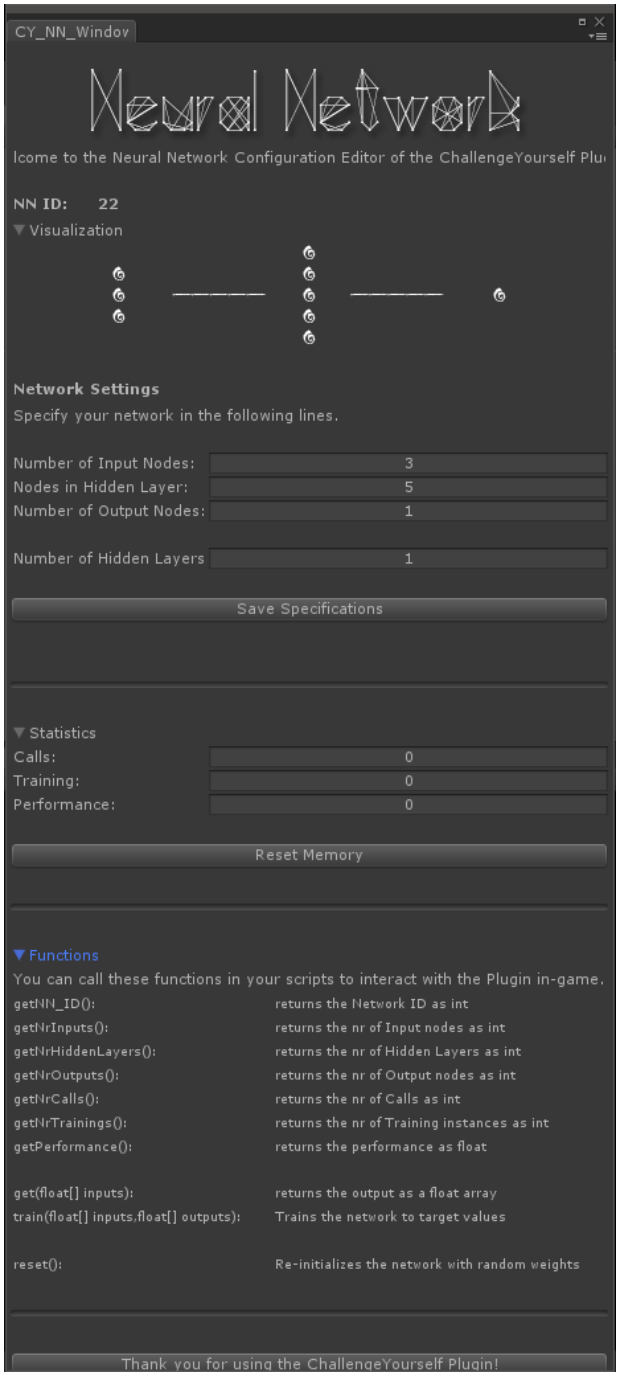


Figure 5.2: An example editor window for a neural network module

Chapter 6

Conclusion

Artificial Intelligence in video games is getting more and more potential applications, not restricted to decision making, pathfinding, learning or content generation. Historically, many academic approaches came up with innovative ideas that nevertheless rarely made it to actual industrial games. Higher computational complexity, lower debuggability and no deeper understanding discouraged the professional implementation, since the common education of game developers (game design, media informatics, computer science) do not include the concepts and applications of advanced AI techniques in most of the cases. However, nowadays developers are highly interested in methods to make their games more believable and innovative, are experimenting around with new approaches and welcome the progress of AI technology. Especially when it becomes well-researched and affordable (where the best recent examples are AI cloud services, e.g. IBM's *Cognitive Solutions*, Microsofts *Cognitive Services* or Googles *Cloud ML*), more and more companies will jump on the bandwagon. As shown in chapter 4, the implementation of intelligent agents can be accomplished successfully in a reasonable amount of time, which improves not only the believability of the opponent, but also the challenge the player experiences. This success is shared via the *Challenge Yourself* plugin to bring an easy-to-use interface for developers interested in human player modelling, adaptation and decision making along with already functioning analogical implementation examples. To demonstrate that neither *Challenge Yourself* is only restricted on one game, nor *Korona: Tinker's Curse* is merely adapted to the learning module, the outline gives a hint on how to use the plugin externally in chapter 7.1.

Chapter 7

Outline

7.1 *Challenge Yourself* in other genres

In principle, the implementation that armed *Korona: Tinker's Curse* with an intelligent opponent is applicable to a non-limited range of games and genres, since every continuous mathematical function can be modeled by multi-layered neural networks (after the universal approximation theorem [7]), which in this case constitute the multivariate mappings from game states to player estimations or rewards. Demonstrating this in practice (without the actual source code or access to the game variables) however leads to two major challenges: The assessment of game states and the common occurrence of movement. Nevertheless, the videos of Figure 7.1 demonstrate that the *Challenge Yourself plugin* can also be used to play games *"from outside"*, i.e. without an implementation of the actual developers.

MMORPG

Guild Wars 2 is one of the most dominant representatives in the genre of MMORPGs, which intrinsically offer great potential for player modelling and game learning, since an active internet connection is required to play (enabling live feedback and cloud computing), a large player count is available (>5 million sold copies in this case, enabling drawing from a vast dataset), several tasks have to be repeated over and over (enabling a basis for adaptive difficulties) and the battle mechanic is mainly defined by the decision making of certain moves in certain states.

In this example, the game state consists of a list of moves (1-12) that can be either available or on a cool-down period, the opponent's health points and the own health points, which are

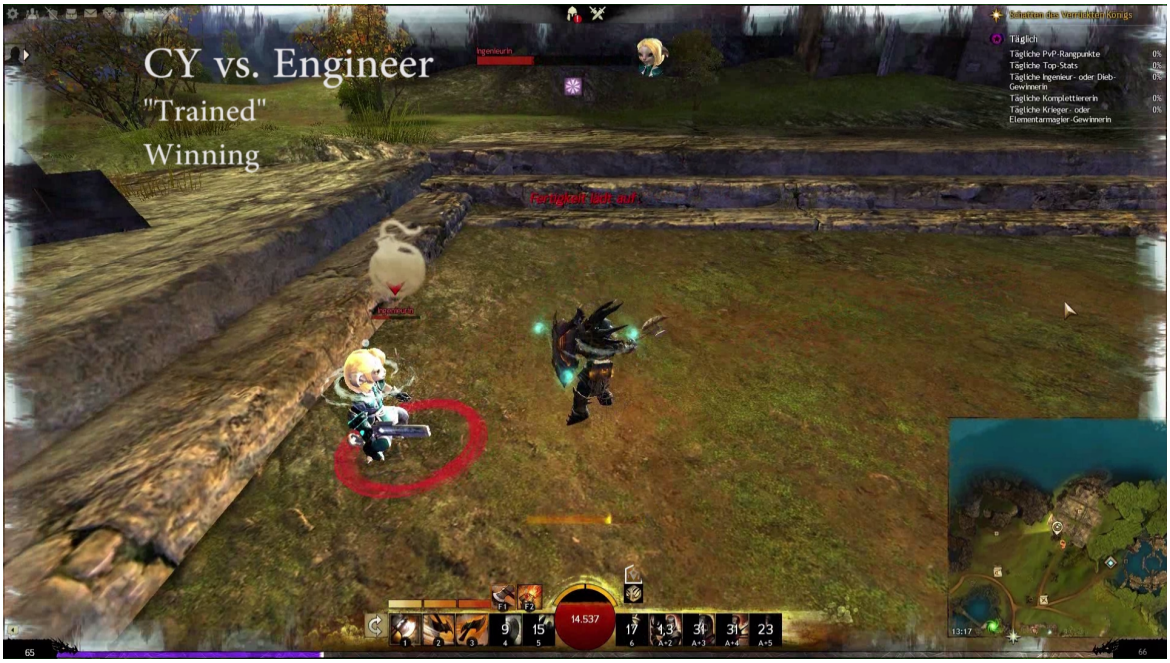


Figure 7.1: Application of the framework to another environment (over 100 iterations)
(Video: ChallengeYourself vs. Guild Wars 2 (untrained) [43])
(Video: ChallengeYourself vs. Guild Wars 2 (trained) [42])

assessed heuristically. Other approaches could include the readout of the memory reserved for these variables or reading from the screen using *optical character recognition*, but all these methods do not result in assessments that are as reliable as a direct implementation of the plugin would be. Thus, *Challenge Yourself* is still primarily intended for developers that wish to include learning and modeling in video game projects and not to automate 3rd party games. The outcome of the particular network is then forwarded to keystrokes in the respective game process.

Secondly, movement is an important factor in nearly all video games that somehow model an environment, be it 2D or 3D. Apparently, the purpose of this plugin is not movement, since much more feasible implementations already exist to solve these problems, which found their place in industrial game AI (See: chapter 1.2.2). Thus, the former example focuses on fighting without moving, choosing opponents with a minimal amount of movement. In actual implementations of intelligent agents, *Challenge Yourself* should be used for prediction and reward estimation, while being coupled to established pathfinding algorithms that take care of the movement problem.

While these confrontations in *Guild Wars 2* have a very low probability of winning when playing randomly under the depicted circumstances (1 of 9 attempts), already after 100 training steps the performance raises to winning 8 of 9 cases.

Adventure

The classic setup of Adventure games does not contain one-versus-one opponent scenarios in which *Challenge Yourself* proved to shine. However, I introduced the framework and its potential applications to one of the most successful german video game studio for development and publishing, *Daedalic Entertainment*, and tailored it to a performant solver for automated game testing. This successor called ICARUS (Intelligent Completion of Adventure-Riddles via Unsupervised Solving) is based on the environment-action observation, evaluation and reinforcement learning of the game state and is applied for autonomous Quality Assurance trackings that had a high demand on human resources prior to this. Since January 2017 it is deployed in the development process of upcoming high-quality Adventure titles, e.g. the game-adaption of Ken Follett's world-bestseller "The Pillars of the Earth".

Appendices

Descriptive appendices

Documentation of industrial video game company correspondence

Note: Developer Names and Contact Emails have been removed due to privacy reasons.

Company	Country	Notable Games	Webpage	Contact Email	Sent?	Survey?	Response?
2K Games	USA	Borderlands, Civilization, BioShock	www.2k.com	inquiries@2kgames.com https://twitter.com/2KSupport	×✓		
2K Czech	Czech Republic	Mafia	www.2kczech.com	https://twitter.com/2kczech?lang=de	✓✓		
3D Realms	USA	Duke Nukem	3drealms.com	hello@3drealms.com	✓✓		
343 Industries	USA	Halo	www.halowaypoint.com	Community@halowaypoint.com BSANGEL@halowaypoint.com https://twitter.com/Halo	✓✓		
4A Games	Ukraine	Metro	www.4a-games.com.mt	http://4a-games.com.mt/52-2/contact/	✓		
5th Cell	USA	Scribblenauts	www.5thcell.com	http://www.5thcell.com/contact/	×		
Activision Blizzard	USA	Call of Duty, Diablo, Warcraft	www.activisionblizzard.com	https://support.activision.com/ https://us.battle.net/support/en/ pr@blizzard.com	✓✓	×	No Time
ACE Team	Chile	Rock of Ages	aceteam.cl	contact@aceteam.cl	✓✓		
Avalanche Studios	Sweden	Just Cause	www.avalanchestudios.se	press@avalanchestudios.se https://twitter.com/thomaswiborgh	✓✓		
Access Games	Japan	Deadly Premonition	http://www.accessgames.co.jp	http://www.accessgames.co.jp/contact/webmaster/	✓		
Aventurine SA	Greece	Darkfall	http://www.aventurine.gr/	contact@aventurine.gr	✓✓		
ArenaNet	USA	Guild Wars	http://www.arena.net/	pr@arena.net ArenaNetRecruitment@arena.net PubRel: ronb@arena.net	✓✓	?	Public Relations Contact
Arkane Studios	France	Dishonored	www.arkane-studios.com	https://twitter.com/arkanestudios	✓✓		
Bandai Namco Entertainment	Japan	Tekken, Dragon Ball	bandainamcoent.co.jp	support@bandainamcoent.com	✓✓		
Bethesda Softworks	USA	Elder Scrolls, Fallout, Doom	bethsoft.com	info@bethsoft.com press@bethsoft.com	✓✓		
Big Huge Games	USA	Rise of Nations	bighugegames.com	https://twitter.com/thebighugegames	✓✓		
BioWare	Canada	Baldurs Gate, Neverwinter Nights, Mass Effect	www.bioware.com	https://twitter.com/BioWare	✓✓		
Black Shell Games	USA	Sanctuary	http://www.blackshellmedia.com	http://blackshellmedia.com/contact/ contact@blackshellmedia.com	✓	✓	Developer Contact!
Blue Byte	Germany	Die Siedler, Anno	bluebyte.com	info@bluebyte.de	✓✓		
Bugbear Entertainment	Finland	Flatout	www.bugbear.fi	bugbear@bugbear.fi press@bugbear.fi	✓✓		
Bungie	USA	Halo	www.bungie.net	https://twitter.com/bungie media@bungie.com	✓✓		
Capcom	Japan	Resident Evil, Devil may cry	http://www.capcom.com/	feedback@capcom.com	✓	×	No Time
CCP Games	Iceland	Eve Online	www.ccpgames.com	info@ccpgames.com	✓✓		
CD Projekt RED	Poland	The Witcher	www.cdprojekt.com	recepcja@cdprojekt.com	✓✓	×	No Time
CodeMasters	England	Overlord, F1, Colin McRae	http://www.codemasters.com/	custservice@codemasters.com	✓✓	×	No Time, Company Secret
Creative Assembly	England	Total War	www.creative-assembly.co.uk	https://twitter.com/CAGames	✓✓		
Criterion Games	England	Burnout, Need for Speed	http://www.criteriongames.com/	https://twitter.com/CriterionGames	✓		
Croteam	Croatia	Serious Sam	www.croteam.com	support@croteam.com press@croteam.com	✓	✓	Developer Contact!
Cryptic Studios	USA	City of Heroes, Neverwinter	http://www.crypticstudios.com	http://www.crypticstudios.com/business	✓		
Crystal Dynamics	USA	Legacy of Kain, Tomb Raider	www.crystald.com	https://crystald.com/	✓		
Crytek	Germany	Crysis, Far Cry, Ryse	www.crytek.com	contact@crytek.com press@crytek.com	✓✓	✓	Developer Contact!
Deck13	Germany	Ankh, Jack Keane	www.deck13.com	support@deck13.com mhoss@deck13.com	✓✓		
DICE	Sweden	Battlefield, Mirror's Edge	www.dice.se	info@dice.se https://twitter.com/ea_dice	✓✓		
Eidos Montreal	Canada	Deus Ex, Thief	www.eidosmontreal.com	community@eidosmontreal.com https://twitter.com/eidosmontreal	✓✓		
Egosoft	Germany	X	Egosoft.com	info@egosoft.com	✓✓		
EA Games	USA	...	www.ea.com	https://twitter.com/askeasupport	✓✓	×	No Time
Epic Games	USA	Unreal, Gears of War	www.epicgames.com	http://help.epicgames.com/customer/portal/emails/new	✓	?	Forwarded to Developer
Facepunch Studios	England	Rust	facepunchstudios.com	contact@facepunchstudios.com	✓✓		
Firefly Studios	England	Stronghold	http://www.fireflyworlds.com/	http://www.fireflyworlds.com/contact/	✓		
Frictional Games	Sweden	Amnesia	www.frictionalgames.com	http://www.frictionalgames.com/site/contact	✓		
Frontier Developments	England	Tycoons, Coasters	http://www.frontier.co.uk/	https://twitter.com/Frontier_Help	✓	?	Forwarded to Developer
Funcom	Norway	Age of Conan, The Longest Journey	www.funcom.com	http://help.funcom.com http://pr.funcom.com	✓		
Gaijin Entertainment	Russia	War Thunder	gaijinent.com	http://gaijinent.com/en/contacts/ press@gaijinent.com	✓		
Game Freak	Japan	Pokemon	www.gamefreak.co.jp	https://twitter.com/Pokemon_cojp	✓✓		
Gearbox Software	USA	Borderlands	www.gearboxsoftware.com	http://www.gearboxsoftware.com/contact/	✓		
GSC Game World	Ukraine	S.T.A.L.K.E.R., Cossacks	http://www.gsc-game.com/ http://www.haemimontgames.com/	info@gsc-game.com	✓✓		
Haemimont Games	Bulgaria	Tzar, Tropico		info@haemimontgames.com	✓✓		
Harebrained Schemes	USA	Shadowrun	harebrained-schemes.com	http://harebrained-schemes.com/contact/	✓	✓	Developer Contact!
Hello Games	England	No Man's Sky	www.hellogames.org	https://twitter.com/hellogames	✓✓		
id Software	USA	Quake, Doom	idsoftware.com	http://www.idsoftware.com/en-us https://twitter.com/DOOM	✓		
Infinity Ward	USA	Call of Duty	www.infinityward.com	https://twitter.com/InfinityWard	✓✓		
IO Interactive	Denmark	Hitman, Kane & Lynch	www.ioi.dk	ioi@ioi.dk https://twitter.com/iointeractive	✓	×	No Time
Koei	Japan	Dynasty/Samurai Warriors	www.koei.co.jp	https://www.koeitecmo.co.jp/inquiry/	✓		
Konami	Japan	Metal Gear, Silent Hill, Castlevania	https://www.konami.com/en/	https://www.konami.com/inquiry/ir_e.php	✓		
Level-5	Japan	Yo-Kai Watch, Prof. Layton	level5ia.com	https://level5ia.com/contact/	✓		
Massive Entertainment AB	Sweden	The Division, Ground Control	www.massive.se	hello@massive.se	✓✓		
Maxis	USA	Sims, Spore	http://www.maxis.com/	http://www.ea.com/de/1/kontaktformular-ea https://twitter.com/TheSims	✓		
Microsoft Studios	USA	...	www.microsoftstudios.com	https://twitter.com/coalitiongears	✓✓		
Mojang	Sweden	Minecraft	mojang.com	https://twitter.com/mojang https://twitter.com/mojangsupport	✓✓		
Monolith Productions	USA	F.E.A.R.	www.lith.com	remi.sklar@warnerbros.com https://twitter.com/MonolithDev	✓✓		
Nadeo	France	TrackMania	www.nadeo.com	contact@nadeo.com	✓✓		
Naughty Dog	USA	Uncharted, The Last of Us	http://www.naughtydog.com/	ndi-dog@naughtydog.com	✓✓		
NCsoft	South Korea	Lineage, Aion, Blade & Soul	http://www.ncsoft.com/	info@ncsoft.com NCWPR@ncsoft.com	✓✓		
NetherRealm Studios	USA	Mortal Kombat, Injustice	www.netherrealm.com	https://twitter.com/NetherRealm	✓✓		
Niantic	USA	Ingress, Pokemon Go	www.nianticlabs.com	partnerships@nianticlabs.com	✓✓		
Ninja Theory	England	Enslaved, DmC	http://www.ninjatheory.com/	info@ninjatheory.com	✓✓		
Nintendo	Japan	...	www.nintendo.com	support@noa.com contact_NDA@nintendo.co.jp	✓✓	×	Error
Oddworld Inhabitants	USA	Oddworld	Oddworld.com	http://www.oddworld.com/contact-oddworld/ info@oddworld.com	✓✓		
Obsidian Entertainment	USA	Stick of Truth, Fallout: New Vegas	www.obsidian.net	support@obsidian.net https://twitter.com/obsidian	✓✓	✓	Developer Contact!
Overkill Software	Sweden	Payday	www.overkillsoftware.com	press@overkillsoftware.com info@starbreeze.com	✓✓		
Panther Games	Australia	Airborne Assault	www.panthergames.com	http://twitter.com/PantherGames	✓✓		
Paradox Interactive	Sweden	Magicka, Hearts of Iron	www.paradoxplaza.com	ir@paradoxplaza.com https://twitter.com/PdxInteractive	✓✓	✓	Developer Contact!
Perfect World	China	Perfect World	wanmei.com	service@wanmei.com global@wanmei.com	×		
Piranha Bytes	Germany	Gothic, Risen	www.piranha-bytes.com	twitter.com/Piranha_Bytes	✓✓		
Playdom	USA	Facebook games	playdom.com	https://twitter.com/playdom?lang=de	✓		
Playground Games	England	Forza	http://www.playground-games.com/	https://twitter.com/weareplayground	✓✓		
PlatinumGames	Japan	Bayonetta, MadWorld	http://platinumgames.com	https://twitter.com/platinumgames	✓✓		
Polyphony Digital	Japan	Gran Tourismo	http://www.polyphony.co.jp/	SCE_Recruit02@hq.scei.sony.co.jp	✓✓		
Quantic Dream	France	Heavy Rain, Fahrenheit, Beyond: Two Souls	http://www.quanticdream.com/en/	http://www.quanticdream.com/en/#!/en/contact https://twitter.com/quantic_dream?lang=de	✓		
Radical Entertainment	Canada	Prototype	www.radical.ca	https://twitter.com/radical_ent?lang=de	✓✓		
Rare	England	Sea of Thieves, ...	www.rare.co.uk	https://twitter.com/rareltd editor@rare.co.uk	✓✓		

Raven Software	USA	Call of Duty	ravensoftware.com	https://twitter.com/ravensoftware	✓✓		
Red Storm Entertainment	USA	Tom Clancy's ...	http://redstorm.com/	contact@redstorm.com	✓✓		
Reflections Interactive	England	Driver	reflections.ubisoft.com	reflections@ubisoft.com	✓✓		
Remedy Entertainment	Finland	Max Payne, Alan Wake	www.remedygames.com	press@remedygames.com feedback@remedygames.com	✓✓		
Respawn Entertainment	USA	Titanfall	www.respawn.com	info@respawn.com	✓✓		
Retro Studios	USA	Metroid Prime	www.retrostudios.com	https://twitter.com/retrostudios	✓✓		
Riot Games	USA	League of Legends	www.riotgames.com	info@riotgames.com https://twitter.com/riotgames	✓✓	×	No Time
Rockstar Games	USA	GTA, Red Dead Redemption	www.rockstargames.com	http://www.rockstargames.com/#/?lb=mouthoff PR@rockstargames.com https://twitter.com/RockstarGames	✓✓		
Rocksteady Studios	England	Batman	http://www.rocksteadyltd.com/	info@rocksteadyltd.com https://twitter.com/rocksteadygames/	×		
Running with Scissors	USA	Postal	runningwithscissors.com	https://twitter.com/RWSbleeter	✓✓		
Sony Computer Entertainment	Japan	...	http://www.sie.com/en/index.html	jennifer_clark@playstation.sony.com	✓✓		
Square Enix	Japan	Final Fantasy, ...	www.square-enix.com	https://twitter.com/SQUARE_ENIX_EU/ https://form3.square-enix.com/a.p/210/	✓✓		
Sucker Punch Productions	USA	Infamous	http://suckerpunch.playstation.com/	info@suckerpunch.com http://twitter.com/suckerpunchprod	✓✓		
Supercell	Finland	Clash of Clans	http://supercell.com/en/	https://twitter.com/supercellgames info@supercell.net	✓✓		
Techland	Poland	Dead Island, Call of Juarez	www.techland.pl	http://company.techland.pl/contact	✓		
Turtle Rock Studios	USA	Left for Dead, Evolve	www.turtlerockstudios.com	https://www.turtlerockstudios.com/our-studio/contact-us/ https://twitter.com/TurtleRock	✓		
Ubisoft	France	Assassins Creed, ...	www.ubisoft.com	infoMTL@ubisoft.com https://twitter.com/Ubisoft	✓	×	No Time
Valve	USA	Half-Life, Portal, Counter Strike, TF2, Dota	www.valvesoftware.com	academiclicensing@valvesoftware.com sourceengine@valvesoftware.com sdk@valvesoftware.com	✓		
Visceral Games	USA	Dead Space	www.visceralgames.com	https://twitter.com/visceralgames	✓✓		
Visual Concepts	USA	NFL, NBA, MLB 2K	http://www.vcentertainment.com/us/	https://twitter.com/2ksports	×		
Volition, Inc	USA	Saints Row	www.dsvolition.com	http://www.dsvolition.com/contact/ https://twitter.com/SaintsRow	✓		
Wargaming	Cyprus	World of Tanks	http://wargaming.com/	https://twitter.com/wargaming_net?lang=de	✓✓		
ZeniMax Online Studios	USA	The Elder Scrolls Online	zenimaxonline.com	https://twitter.com/TESOnline	✓✓		
Zynga	USA	FarmVille	zynga.com	https://twitter.com/zynga	✓✓		

AI and Machine Learning in state-of-the-art Video Games: A Digital Interview

Institute of Cognitive Science University of Osnabrück Albrechtstraße 28 D-49069 Osnabrück

AI and Machine Learning in state-of-the-art Video Games: A Digital Interview

Please input as much information as you are able to provide.
(Blank fields will be treated as anonymous statements.)
The data you state will be used (and only used) in the Master Thesis **"Challenge yourself: A cognitive agents' machine learning framework for adaptive, intelligent and interesting competition in video game opponents"**, by Johannes Pfau, 2016.

Name:
Company:
Country:
Affiliated Game(s):

Which disciplines of AI are implemented in your Video Games?

- ☐ Pathfinding — Purpose:
- ☐ Finite State Automata — Purpose:
- ☐ Dynamic game difficulty balancing — Purpose:
- ☐ Machine Learning — Purpose:
- ☐ Computational Intelligence — Purpose:
- ☐ Logic / Reasoning Systems — Purpose:
- ☐ Evolutionary Computation — Purpose:
- ☐ Computational Creativity — Purpose:
- ☐ Affective Computing — Purpose:
- ☐ Player Modeling — Purpose:
- ☐ Knowledge-Based Systems — Purpose:
- ☐ Procedural Content Generation — Purpose:
- ☐ Human-Computer Interaction — Purpose:
- ☐ Psychophysiological Measurements — Purpose:
- ☐ Intelligent Multi-Agent Interaction — Purpose:
- ☐ Contextual Language Generation — Purpose:
- ☐ Ontology Engineering — Purpose:
- ☐ Big Data Analysis — Purpose:
- ☐ Cognitive Architectures — Purpose:

Other:

What kind of explicit AI algorithms/techniques did you use in your Video Games?

- ☐ A* — Purpose:
- ☐ q-Learning — Purpose:
- ☐ Artificial Neural Networks — Purpose:
- ☐ Neuroevolution — Purpose:
- ☐ Dynamic Scripting — Purpose:
- ☐ Prolog (or other Logic Programming Languages) — Purpose:
- ☐ Fuzzy Logic Reasoning — Purpose:
- ☐ Lisp (or other Functional Programming Languages) — Purpose:

☐ STRIPS (or other Planning systems) Purpose:
☐ Chatbots Purpose:
☐ Hidden Markov Models Purpose:
☐ Memory/Knowledge Models for NPCs Purpose:
☐ Semantic Networks Purpose:
☐ OWL (or other Ontology Languages) Purpose:
☐ NEEDSIM (or other NPC need simulation) Purpose:
☐ SOAR, ACT-R (or other Cognitive Architectures) Purpose:

Other (including own implementations/developments; in this case please provide a short description of the functionality):

Verbal Statement: How is your company using artificial intelligence to improve Video Games?

What is your personal opinion about AI in Video Games?

AI in Video Games is interesting for the players:

Disagree — Agree

AI in Video Games can make the environment more believable:

Disagree — Agree

AI in Video Games is too much effort to implement (in terms of money and labor investment):

Disagree — Agree

The Video Game industry actually wants to make use of academically developed AI:

Disagree — Agree

Additional comments:

☐ Anonymity: I don't want that my name, the name of my company or the name of affiliated games is used or mentioned anywhere.

Submit!

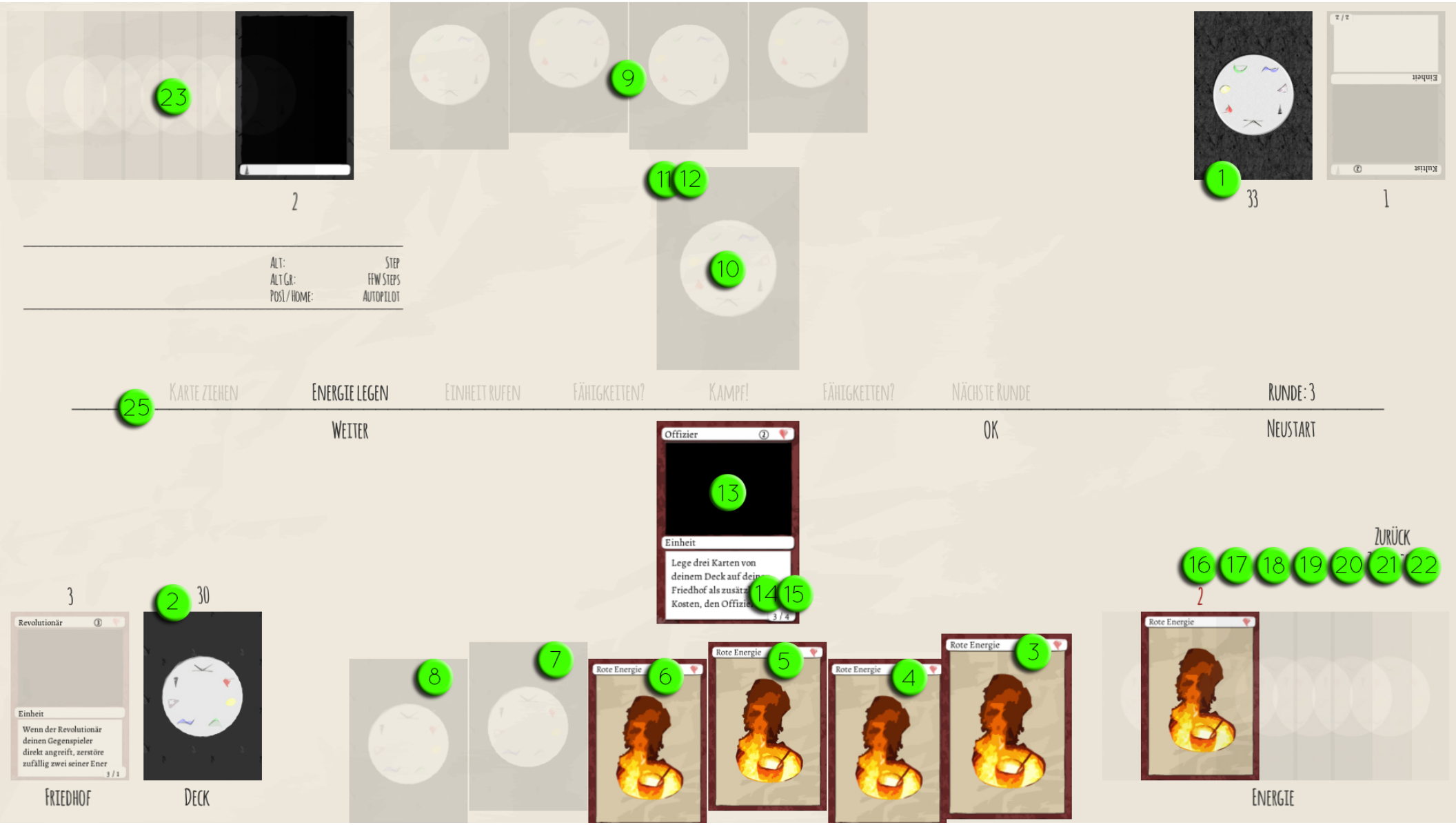
Thank you for your participation!
Please hit "Submit" and have a wonderful day.
You allow us to use your data (anonymously), exclusively for this thesis, without transfer to third parties.
If you have any questions, comments or suggestions, don't hesitate to contact me at jpfau@uos.de.

Complete state space description of Korona: Tinker's Curse at a single point of time

INPUT: Environment

{1} PlayerDeck.count - The amount of cards in the human player's deck left (0-60)
{2} OpponentDeck.count - The amount of cards in the AI player's deck left (0-60)
{3} HandCard1-ID - The ID of card 1 in hand (0 if not existing)
{4} HandCard2-ID - The ID of card 2 in hand (0 if not existing)
{5} HandCard3-ID - The ID of card 3 in hand (0 if not existing)
{6} HandCard4-ID - The ID of card 4 in hand (0 if not existing)
{7} HandCard5-ID - The ID of card 5 in hand (0 if not existing)
{8} HandCard6-ID - The ID of card 6 in hand (0 if not existing)
{9} OpponentHandCards.count - The amount of cards in the human player's hand
{10} PlayerUnit-ID - The ID of human players unit (0 if not existing)
{11} PlayerUnit-Atk - The attack value of human players unit
{12} PlayerUnit-Def - The defense value of human players unit
{13} OpponentUnit-ID - The ID of AI players unit (0 if not existing)
{14} OpponentUnit-Atk - The attack value of AI players unit
{15} OpponentUnit-Def - The defense value of AI players unit
{16} AmountEnergyRed - The amount of red energy of the AI player
{17} AmountEnergyYellow - The amount of yellow energy of the AI player
{18} AmountEnergyGreen - The amount of green energy of the AI player
{19} AmountEnergyBlue - The amount of blue energy of the AI player
{20} AmountEnergyViolet - The amount of violet energy of the AI player
{21} AmountEnergyBlack - The amount of black energy of the AI player
{22} AmountEnergyWhite - The amount of white energy of the AI player
{23} AmountEnergyPlayer - The total amount of energy of the human player
{24} DISCARD-Mode - 1 if DISCARD-mode is on, 0 otherwise
{25} Phase - The ID of the current phase (1-7)

INPUT: Environment

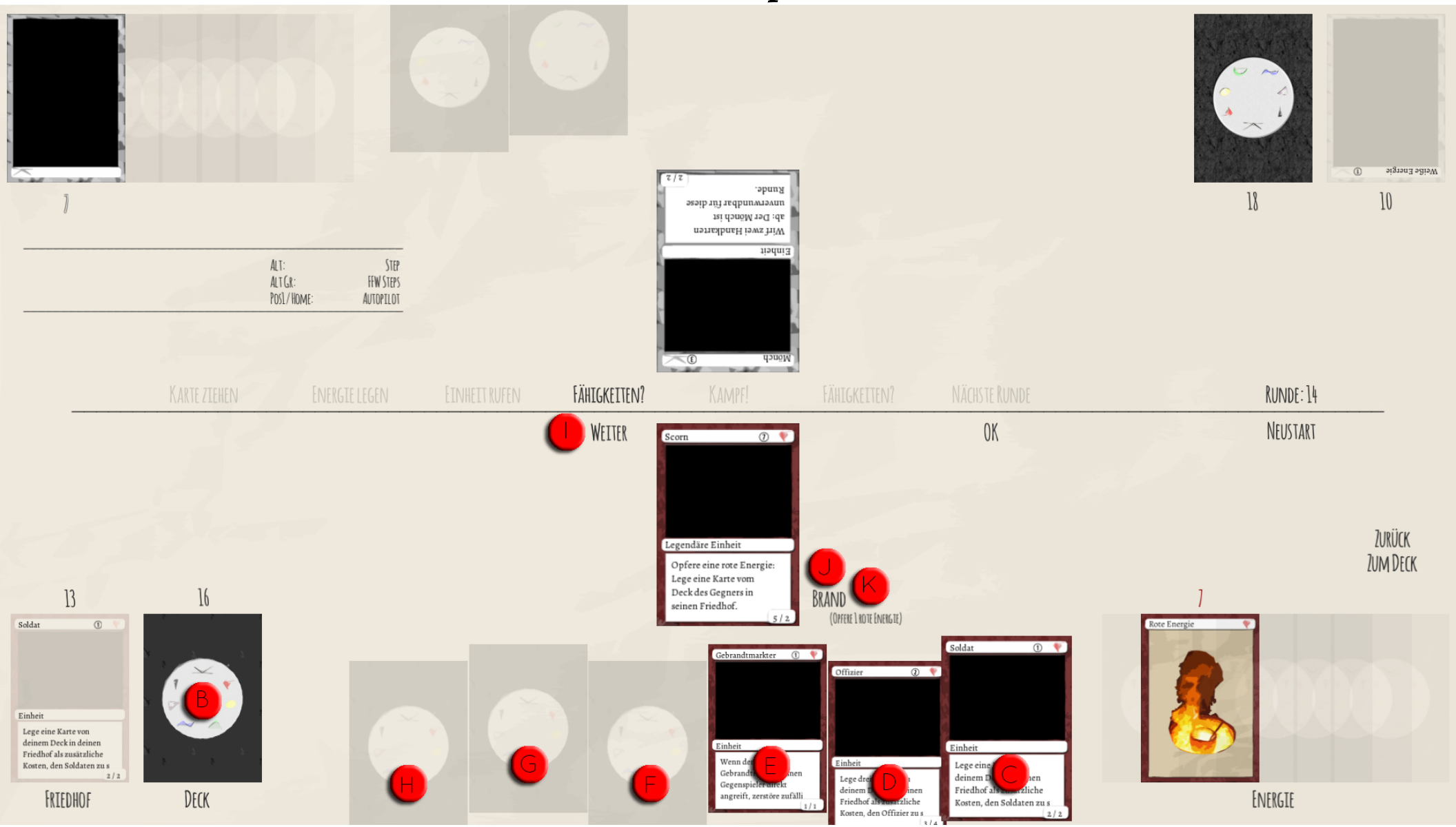


OUTPUT: Action Space

{A} wait() - The agent will wait for a players action
{B} tryToDraw() - The agent will try to draw a card
{C} play HandCard1() - The agent will try to play hand card 1
{D} play HandCard2() - The agent will try to play hand card 2
{E} play HandCard3() - The agent will try to play hand card 3
{F} play HandCard4() - The agent will try to play hand card 4
{G} play HandCard5() - The agent will try to play hand card 5
{H} play HandCard6() - The agent will try to play hand card 6
{I} nextPhase() - The agent will try get into the next phase
{J} activateAbility() - The agent will try to activate the ability of his unit
{K} replaceUnit() - The agent will try to replace his unit

Note: In DISCARD-Mode choosing C-H results in discarding the respective Card into the graveyard instead of playing it. This happens when too many cards are on the hand or due to several unit abilities.

OUTPUT: Action Space



List of all cards in *Korona: Tinker's Curse*

ID	Color	Name	Type	Cost	A	D	Effect
1	Black	Schwarze Energie	Energy				
2	White	Weiße Energie	Energy				
3	Red	Rote Energie	Energy				
4	Yellow	Gelbe Energie	Energy				
5	Green	Grüne Energie	Energy				
6	Blue	Blaue Energie	Energy				
7	Violet	Violette Energie	Energy				
8	-	Die Sonne	Legendary Energy				If you control one Energy of each color, you win the game.
9	White	Der Erzpatriarch	Legendary Unit	7	3	1	When you play "Der Erzpatriarch", destroy all non-white Energies and Units.
10	Black	Cor Isaet	Legendary Unit	7	3	3	When "Cor Isaet" dies, destroy all Units, Energies and Hand Cards of all players.
11	Red	Scorn	Legendary Unit	7	5	2	Sacrifice 1 Red Energy: Deal 1 Damage to the opponent's Deck.
12	Red	Butch	Legendary Unit	7	4	2	All other Red Units gain +2/+0 for the rest of the game.
13	Yellow	Der Geldsack	Legendary Unit	7	1	4	If you control 10 or more yellow Energies, you win the game.
14	Green	Iyalara'Nqar	Legendary Unit	7	2	2	Whenever "Iyalara'Nqar" survives a battle, she gets permanently +1/+1. Whenever she would lose a battle, she gets permanently -1/-1 instead, if her Defense is greater than 1.
15	Blue	Thvindazar	Legendary Unit	7	0	0	Thvindazar gets +1/+1 for each card in your hand.
16	Blue	Vaira	Legendary Unit	7	4	2	When you play "Vaira", return all Energies of your opponent to his hand.
17	Violet	Khan	Legendary Unit	7	1	1	When "Khan" dies, discard all your hand cards. Then, take control over all opponent's Energies and Hand Cards.
18	Black	Friedhofsgärtner	Unit	1	1	1	
19	Black	Kultist	Unit	2	2	2	
20	Black	Nekromant	Unit	3	3	3	
21	Black	Skelett	Unit	1	1	1	Sacrifice 1 Black Energy: Return "Skelett" from your Graveyard to play.
22	Black	Rastloser	Unit	3	2	2	Discard 2 Hand Cards: Return "Rastloser" from your Graveyard to play.
23	Black	Dämon	Unit	5	5	1	Sacrifice 1 Unit: Return "Dämon" from your Graveyard to play.
24	Black	Dieb	Unit	1	1	1	Whenever "Dieb" survives a battle, draw a card.
25	Black	Spion	Unit	3	1	4	Whenever "Spion" survives a battle, draw a card and your opponent discards a card.
26	Black	Assassine	Unit	5	2	5	Whenever "Assassine" deals damage to a Unit, destroy it at the end of the battle.
27	White	Messdiener	Unit	1	0	2	
28	White	Kirchgänger	Unit	2	1	3	
29	White	Sonnenanbeter	Unit	3	2	4	
30	White	Novize	Unit	1	1	1	Sacrifice 1 White Energy: "Novize" becomes invincible until end of turn.
31	White	Mönch	Unit	3	2	2	Discard 2 Hand Cards: "Mönch" becomes invincible until end of turn.
32	White	Priester	Unit	3	1	4	Discard "Priester" from your Hand: Your Unit becomes invincible until end of turn.
33	White	Exekutor	Unit	1	1	1	"Exekutor" gets +2/+0 if the opponent controls a non-white Unit.
34	White	Exorzist	Unit	3	1	1	"Exorzist" gets +4/+0 if the opponent controls a non-white Unit.
35	White	Richter	Unit	5	3	3	Other Units don't have effects.
36	Red	Koch	Unit	1	0	2	
37	Red	Wachposten	Unit	2	1	3	
38	Red	Blutarzt	Unit	3	1	5	
39	Red	Soldat	Unit	1	2	2	When you play "Soldat", put the top card of your Deck into your Graveyard.
40	Red	Offizier	Unit	2	3	4	When you play "Offizier", put the top 3 cards of your Deck into your Graveyard.
41	Red	General	Unit	3	5	4	When you play "General", put the top 5 cards of your Deck into your Graveyard.
42	Red	Gebrandmarkter	Unit	1	1	1	When "Gebrandmarkter" deals direct damage to your opponent, destroy one of his Energies at random.
43	Red	Revolutionär	Unit	3	3	1	When "Revolutionary" deals direct damage to your opponent, destroy two of his Energies at random.
44	Red	Weltbrandler	Unit	5	4	1	When "Weltbrandler" deals direct damage to your opponent, destroy all Energies on the board.
45	Yellow	Erbsenzähler	Unit	1	1	1	
46	Yellow	Geizkragen	Unit	2	3	1	
47	Yellow	Gierschlund	Unit	3	5	1	
48	Yellow	Händler	Unit	1	1	1	When you play "Händler", you can play an additional Energy this turn.
49	Yellow	Spekulant	Unit	3	2	1	When you play "Spekulant", bring the next Energy card from your Deck into play.
50	Yellow	Vorsitzender	Unit	5	3	4	Whenever "Vorsitzender" survives a battle, bring the next Energy card from your Deck into play.
51	Yellow	Verwalter	Unit	1	0	2	In the turn in which "Verwalter" is played, no battle takes place.
52	Yellow	Abgeordneter	Unit	3	0	4	Sacrifice 1 Yellow Energy: No battle takes place in this turn.
53	Yellow	Politiker	Unit	5	0	1	Move 1 Card from your Deck into your Graveyard: No battle takes place in this turn.
54	Green	Zoologist	Unit	1	1	1	
55	Green	Wilderling	Unit	2	2	2	
56	Green	Druide	Unit	3	3	4	
57	Green	Dornenkind	Unit	1	2	1	When "Dornenkind" has more Attack than the opponent's Unit, deal the rest damage directly to his Deck.
58	Green	Dornenmaid	Unit	3	4	1	When "Dornenmaid" has more Attack than the opponent's Unit, deal the rest damage directly to his Deck.
59	Green	Dornenwall	Unit	5	0	5	In each battle, the opposing Unit deals itself damage in the amount of its own Attack.
60	Green	Giftmischer	Unit	1	1	1	When "Giftmischer" fights a Unit, it gets permanently -1/-1.
61	Green	Experimentator	Unit	3	1	3	When "Giftmischer" fights a Unit, it gets permanently -2/-2.
62	Green	Missglücktes Experin	Unit	5	8	8	After each battle, "Missglücktes Experiment" gets permanently -3/-3.
63	Blue	Matrose	Unit	1	1	1	
64	Blue	Seemann	Unit	2	2	2	
65	Blue	Pirat	Unit	3	4	2	
66	Blue	Aeronaut	Unit	1	1	1	"Aeronaut" deals damage directly to the opponent's Deck.
67	Blue	Wolkenwanderer	Unit	3	2	1	"Wolkenwanderer" deals damage directly to the opponent's Deck.
68	Blue	Gewitterhetzer	Unit	5	3	1	"Gewitterhetzer" deals damage directly to the opponent's Deck.
69	Blue	Bachlenker	Unit	1	0	2	When you play "Bachlenker", return the opponent's Unit back to his hand.
70	Blue	Flussbändiger	Unit	3	0	3	When you play "Flussbändiger", return 2 Energies of the opponent chosen by random back to his hand.
71	Blue	Flutbrecher	Unit	5	0	5	When you play "Flutbrecher", put the opponent's Unit, 1 Energy chosen by random and 1 of his hand cards chosen by random under his Deck.
72	Violet	Polizist	Unit	1	1	1	
73	Violet	Inspektor	Unit	2	2	2	
74	Violet	Kommissar	Unit	3	3	3	
75	Violet	Konditionierer	Unit	1	1	0	When "Konditionierer" dies, take control of 1 Energy of the opponent chosen by random.
76	Violet	Manipulator	Unit	3	2	0	When "Manipulator" dies, take control of the opponent's unit (if it didn't die also).
77	Violet	Lobotomist	Unit	5	3	0	When "Lobotomist" dies, your opponent can't draw cards, play cards or activate effects next turn.
78	Violet	Hellseher	Unit	1	0	1	Your opponent plays with revealed cards.
79	Violet	Medium	Unit	3	0	3	When you play "Medium", add the same amount of Energy your opponent controls to your own Energy pool.
80	Violet	Orakel	Unit	5	0	5	During your ability phase, you can look at the top card of your Deck and decide to shuffle it.
100	Violet	Das Kollektiv	Unit	7	0	3	"Das Kollektiv" gets +1/+0 for each non-violet Energy you control.
101	Green	Der Alchemist	Unit	7	1	3	Sacrifice 1 Green Energy: The opponent's Unit gets permanently -1/-1.

Pseudocode: Advanced heuristic model

```
if( phase == 1 )
        AND if( "opponent (ID:77) died last round" )
                Continue()
        ELSE
                DrawCard()
if( phase == 2 )
        AND if( "opponent (ID:77) died last round" )
        OR ( "no Energy on Hand" )
        OR ( "already played Energy this turn" )
        OR ( "opponent Unit == (ID:42——ID:43——ID:44) && Energy.count == 0
                && "no playable Unit"" )
        OR ("black Energy >= MostExpensiveCardInDeck.costs && Deck.contains(ID:22)")
        OR ("white Energy >= MostExpensiveCardInDeck.costs && Deck.contains(ID:31)")
        OR ( "blue Energy >= 7 && Deck.contains(15)" )
                Continue()
        ELSE
                if( "only Energy of one color on hand" )
                        PlayEnergy()
                ELSE
                        if( "Hand.contains( "Unit that requires Energy X" ) )
                                PlayEnergy(X)
                        ELSE
                                PlayEnergy( "Fewest controlled Energies" )
if( phase == 3 )
        AND if( "opponent (ID:77) died last round" )
        OR ( "no Unit on hand" )
        OR ( "not enough Energy for any Unit on hand" )
                Continue()
        ELSE
                if ( "already Unit on board" ) // replace rule
                AND ( "opponent has already Unit on board" )
                AND ( "Unit can't defeat opponent Unit" )
                AND ( "Hand.contains( "Unit that can defeat opponent Unit" ) )
                OR ( "Hand.contains( "Unit that fulfills one of the following criteria,
                                        while existing Unit doesn't:"
```

```
                ID == 66
                ID == 67
                ID == 68
                ID == 9 && opponentUnit.color != white
                ID == 13 && yellow Energies >= 10
                ID == 14 && opponentUnit.Attack <2
                ID == 15 && handCards >= opponentUnit.Defense
                ID == 26
                ID == 33 && opponentUnit.color != white
                        && opponentUnit.Defense <= 3
                ID == 34 && opponentUnit.color != white
                        && opponentUnit.Defense <= 5
                ID == 50 && opponentUnit.Attack <4
                ID == 59 && opponentUnit.Attack >= opponentUnit.Defense
                ID == 60
                ID == 61
                ID == 71
                ID == 76 && opponentUnit.Defense >2
                ID == 11 && redEnergies >= opponentDeck.count" ) )
                    replaceUnit()
                    X = " cheapest Unit with Attack >= opponentUnit.Defense
                        or superiorUnit"
                    PlayUnit(X)
            ELSE:
                if( "opponent has already Unit on board" )
                    X = " cheapest Unit with Attack >= opponentUnit.Defense "
                    PlayUnit(X)
                ELSE
                    X = "most expensive Unit"
                    PlayEinheit(X)



if( phase == 4 OR phase == 6 )
    if( "already controls ID:35 OR opponentUnit.ID==35")
    OR( "no Unit controlled and no Unit with effect in Graveyard" )
        Continue()
    ELSE
```

```
if( graveyard.contains(ID:21) && blackEnergy >0 && noUnitControlled )
AND (blackEnergy >= 5 OR opponentUnit.Defense <= 1 OR graveyard.contains(ID:23))
        activateAbilityInGraveyard(21)

if( graveyard.contains(ID:22) && handCard.count >= 2 && noUnitControlled)
AND ( handCards >= 4 OR opponentUnit.Defense <= 2 OR graveyard.contains(ID:23)
            OR handCards.contains(ID:23))
        activateAbilityInGraveyard(22)

if( graveyard.contains(ID:23) && unitControlled && unit.ID != 23)
AND ("opponent controls no unit" OR (opponentUnit.Defense <= 5
            && opponentUnit.Defense >Unit.Attack && Unit.ID != 26) )
        activateAbilityInGraveyard(23)

if( unit.ID==30 && "opponent controls unit" && whiteEnergy >0
            && "unit is not invulnerable" )
AND ( opponentUnit.Attack >= unit.Defense )
AND ( "whiteEnergy >= 7 OR unitAttack >= opponentUnit.Defense OR
            handCardCount == 0)
        activateAbility(30)

if( unit.ID==31 && "opponent controls unit" && handCards >= 2
            && "unit is not invulnerable" )
AND ( opponentUnit.Attack >= unit.Defense )
AND (handCards >= 6 OR (whiteEnergy >= 7 && handCards.contains("whiteEnergy")))
        activateAbility(31)


if( "unit controlled" && "opponent controls unit" && handCards.contains("ID:32")
            && "unit is not invulnerable" )
AND ( opponentUnit.Attack >= unit.Defense )
AND ( unit.costs >= 3 )
        activateAbility(32)

if( unit.ID==11 && redEnergy >0)
AND ( redEnergy >7 OR redEnergy >= opponentDeck.count )
        activateAbility(11)
```

```
            if( unit.ID==52 && phase==4 )
            AND ( "opponent controls unit" && opponentUnit.Attack >= unit.Defense
                    OR opponentUnit.ID==26 )
            AND ( yellowEnergy >10 OR handCards == 0 )
                activateAbility(52)

            if( unit.ID==53 && phase==4 )
            AND ( "opponent controls unit" && opponentUnit.Attack >= unit.Defense
                    OR opponentUnit.ID==26 )
            AND ( Deck.count >opponentDeck.count )
                activateAbility(53)

            if( unit.ID==101 && phase==4 && greenEnergy >0
                    && "opponent controls unit" )
            AND ( opponentUnit.Attack >= unit.Defense )
                activateAbility(101)

if( phase == 5 OR phase == 7 )
        Continue();
```

References

[1] *Neural Network Toolbox*. MathWorks, 2016. https://de.mathworks.com/products/neural-network/.

[2] Doug Aamoth. *Heres How Much Time People Spend Playing Video Games*. Time magazine, 2014. http://time.com/120476/nielsen-video-games/.

[3] Robert Ahlers Avelino J. Gonzalez. *Context-based Representation of Intelligent Behavior in Simulated Opponents*. Transactions of the Society for Computer Simulation, 1995.

[4] Paul Carlstroem. *Deep Learning Systems, Artificial Intelligence and Cloud Computing*. Cloud Academy, 2016. http://cloudacademy.com/blog/deep-learning-artificial-intelligence-and-cloud/.

[5] Gerhard Sagerer Christian Bauckhage, Christian Thurau. *Learning Human-like Opponent Behavior for Interactive Computer Games*. Pattern Recognition, Volume 2781, 2003.

[6] Peter Cowling. *Board Evaluation For The Virus Game*. CIG05, 2005.

[7] Balzs Csand Csji. *Approximation with Artificial Neural Networks*. 2001.

[8] Sarah L. Hahn James Quon David B. Fogel, Timothy J. Hays. *Further Evolution of a Self-Learning Chess Program*. CIG05, 2005.

[9] Oxford Dictionary. *Definition of artificial intelligence in English*. Oxford University Press, 2016. https://en.oxforddictionaries.com/definition/artificial_intelligence.

[10] Dongbing Gu Erfu Yang. *A Survey on Multiagent Reinforcement Learning Towards Multi-Robot Systems*. CIG05, 2005.

[11] Ethan Dereszynsk et al. *Learning Probabilistic Behavior Models in Real-time Strategy Games*. Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 2011.

[12] Robert E. Wray et al. *Intelligent opponents for virtual reality trainers*. (I/ITSEC), 2002.

[13] Simon Egenfeldt-Nielsen et al. *Understanding video games: the essential introduction*. Taylor & Francis, 2008.

[14] Volodymyr Mnih et al. *Human-level control through deep reinforcement learning*. Nature 518, 529533, 2015.

[15] Nelis Franken Evangelos Papacostantis, Andries P. Engelbrecht. *Coevolving Probabilistic Game Playing Agents Using Particle Swarm Optimization Algorithms*. CIG05, 2005.

[16] M. S. Campbell A. Nowatzyk F.-h. Hsu, T. S. Anantharaman. *Deep Thought*. Computers, Chess, and Cognition, pp. 55-78, 1990.

[17] Damien Ernst Firas Safadi, Raphael Fonteneau. *Artificial Intelligence in Video Games: Towards a Unified Framework*. International Journal of Computer Games Technology, 2015.

[18] Colin Fyfe Gayle Leen. *Training an AI player to play Pong using a GTM*. CIG05, 2005.

[19] John Hallam Georgios N. Yannakakis. *A Generic Approach for Generating Interesting Interactive Pac-Man Opponents*. Proceedings of the IEEE Symposium on Computational Intelligence and Games, 2005.

[20] John Hallam Georgios N. Yannakakis. *A Generic Approach for Generating Interesting Interactive Pac-Man Opponents*. CIG05, 2005.

[21] John Hallam Georgios N. Yannakakis. *Evolving Opponents for Interesting Interactive Computer Games*. 2012.

[22] Peter M.Todd Gerd Gigerenzer. *Prcis of Simple heuristics that make us smart*. BEHAVIORAL AND BRAIN SCIENCES 23, 2000.

[23] Aliza Gold. *Academic AI and Video games: a case study of incorporating innovative academic research into a video game prototype*. CIG05, 2005.

[24] P.H.M. Spronck H.J. van den Herik, H.H.L.M. Donkers. *Opponent Modelling and Commercial Games*. CIG05, 2005.

[25] Risto Miikkulainen Jacob Schrum, Igor V. Karpov. *UTˆ 2: Human-like Behavior via Neuroevolution of Combat Behavior and Replay of Human Traces*. 2011. http://nn.cs.utexas.edu/?schrum:botprize2011.

[26] Gillian Hayes Jay Bradley. *Adapting Reinforcement Learning for Computer Games: Using Group Utility Functions*. CIG05, 2005.

[27] Bram Bakker Nikos Vlassis Jelle R. Kok, Pieter Jan t Hoen. *Utile Coordination: Learning interdependencies among cooperative agents*. CIG05, 2005.

[28] Sung-Bae Cho JinHyuk Hong. *Evolving Reactive NPCs for the Real-Time Simulation Game*. CIG05, 2005.

[29] Simon M. Lucas Julian Togelius. *Forcing neurocontrollers to exploit sensory symmetry through hard-wired modularity in the game of Cellz*. CIG05, 2005.

[30] Yoshiyuki Kotani Kazutomo Shibahara, Nobuo Inui. *Adaptive Strategies of MTD for Actual Games*. CIG05, 2005.

[31] Donald Kehoe. *Designing Artificial Intelligence for Games*. Intel Developer Zone, 2005. https://software.intel.com/en-us/articles/designing-artificial-intelligence-for-games-part-1.

[32] Donald Fleming Kenneth J. Chisholm. *A Study of Machine Learning Methods using the Game of Fox and Geese*. CIG05, 2005.

[33] Marco Tomassini Leslie Luthi, Mario Giacobini. *Synchronous and Asynchronous Network Evolution in a Population of Stubborn Prisoners*. CIG05, 2005.

[34] Simon M. Lucas. *Evolving a Neural Network Location Evaluator to Play Ms. Pac-Man*. CIG05, 2005.

[35] Jonathan Schaeffer Michael Chung, Michael Buro. *Monte Carlo Planning in RTS Games*. CIG05, 2005.

[36] Yael Niv. *Reinforcement learning in the brain*. Journal of Mathematical Psychology, 2009.

[37] J. Pearl. *Heuristics: Intelligent search strategies for computer problem solving*. Addison-Wesley Pub. Co., Inc.,Reading, MA, 1984.

[38] Barry OSullivan Peter Blackburn. *Building Reactive Characters for Dynamic Gaming Environments*. CIG05, 2005.

[39] Johannes Pfau. *Intelligent adaptive opponents in video games: A machine learning approach to dynamically challenging competition*. 2015.

[40] Johannes Pfau. *Advanced heuristic model of Korona: Tinker's Curse*. 2016. http://nevermindcreations.de/nvmWeb/CY/HM2.png.

[41] Johannes Pfau. *AI and Machine Learning in state-of-the-art video games: A Digital Interview*. 2016. www.nevermindcreations.de/nvmWeb/IndustrialAI/survey.html.

[42] Johannes Pfau. *Video: ChallengeYourself vs. Guild Wars 2 (trained)*. 2016. https://www.youtube.com/watch?v=n_nbOTUCYMs.

[43] Johannes Pfau. *Video: ChallengeYourself vs. Guild Wars 2 (untrained)*. 2016. https://www.youtube.com/watch?v=n_nbOTUCYMs.

[44] Johannes Pfau. *Video: Korona: Tinker's Curse. Game 1, Untrained Agent vs. Advanced Heuristic Opponent*. 2016. https://www.youtube.com/watch?v=3R-bSp9StHE.

[45] Johannes Pfau. *Video: Korona: Tinker's Curse. Permutational confrontation of all ingame Units*. 2016. https://www.youtube.com/watch?v=OUwxD7-VyAg.

[46] Johannes Pfau. *Video: Visualization: Slow Reinforcement Learning*. 2016. https://www.youtube.com/watch?v=GcxmEYo4WQ8.

[47] Eric Postma Pieter Spronck, Ida Sprinkhuizen-Kuyper. *Online adaptation of game opponent AI in simulation and in practice*. Proceedings of the Fourth International Conference on Intelligent Games and Simulation, 2003.

[48] Eric Postma: Pieter Spronck, Ida Sprinkhuizen-Kuyper. *Online Adaptation of Game Opponent AI with Dynamic Scripting*. International Journal of Intelligent Games and Simulation, 2004.

[49] Jordan B. Pollack. *Nannon: A Nano Backgammon for Machine Learning Research*. CIG05, 2005.

[50] Anna-Lena Popkes. *Representation Learning with Feedforward Neural Networks*. 2015.

[51] Microsoft Research. *Video Games and Artificial Intelligence*. 2009. https://www.microsoft.com/en-us/research/project/video-games-and-artificial-intelligence/.

[52] Brendan Sinclair. *Gaming will hit $91.5 billion this year - Newzoo*. Newzoo, 2015. http://www.gamesindustry.biz/articles/2015-04-22-gaming-will-hit-usd91-5-billion-this-year-newzoo.

[53] G. Smith. *Using Co-evolved RTS Opponents to Teach Spatial Tactics*. (CIG10), 2010.

[54] Ugo Merlone Umberto Cerruti, Mario Giacobini. *A New Framework to Analyze Evolutionary 2 2 Symmetric Games*. CIG05, 2005.

[55] Michael van Lent et al. *Intelligent Agents in Computer Games*. AAAI-99 Proceedings, 1999.

[56] J.M.P. van Waveren. *The Quake III Arena Bot*. 2001. http://www.kbs.twi.tudelft.nl/docs/MSc/2001/Waveren_Jean-Paul_van/thesis.pdf.

[57] Ricardo R. Gudwin Victor K. Tatai. *Using a Semiotics-Inspired Tool for the Control of Intelligent Opponents in Computer Games*. International Conference on Integration of Knowledge Intensive Multi-Agent Systems, 2003.

[58] Nick Wedd. *Human-Computer Go Challenges*. CIG05, 1986. http://www.computer-go.info/h-c/.

[59] S. Woodcock. *Game AI: The State of the Industry 2000-2001: It's not Just Art, It's Engineering*. 2001.

[60] Hao Shi Xiao Cui. *A*-based Pathfinding in Modern Computer Games*. IJCSNS International Journal of Computer Science and Network Security, VOL.11, 2011.

[61] Louis E. Yelle. *The learning curve: historical review and comprehensive survey*. Decision Sciences Vol. 10, 1979.

[62] Robert C. Holte Jonathan Schaeffer Yngvi Bjornsson, Markus Enzenberger. *Fringe Search: Beating A* at Pathfinding on Game Maps*. CIG05, 2005.