F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Bachelor's Thesis

Leveraging Reward Regularization in Imperfect Information Games

Tomáš Holeček

20.1. 2024

Supervisor: Ing. Ondřej Kubíček

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Holeček** Jméno: **Tomáš** Osobní číslo: **507264**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Specializace: **Počítačové hry a grafika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Využití metody regularizace odměn ve hrách s neúplnou informací

Název bakalářské práce anglicky:

Leveraging Reward Regularization in Imperfect Information Games

Pokyny pro vypracování:

Seznam doporučené literatury:

Shoham, Y. and Leyton-Brown, K.: Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations, Cambridge University Press, 2008, ISBN 9780521899437.

J. Perolat, et. al., From poincar e recurrence to convergence in imperfect information games: Finding equilibrium via regularization. In M. Meila and T. Zhang, editors, Proceedings of the 38th International Conference on Machine Learning, volume 139 of Proceedings of Machine Learning Research, pages 8525–8535. PMLR, 18–24 Jul 2021.

J. Perolat, et. al., Mastering the game of stratego with model-free multiagent reinforcement learning. Science, 378(6623):990–996, 2022.

S. Sokota, R. D'Orazio, C. K. Ling, D. J. Wu, J. Z. Kolter, and N. Brown. Abstracting imperfect information away from two-player zero-sum games, 2023.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Ondřej Kubíček katedra počítačů FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **07.02.2024** Termín odevzdání bakalářské práce: **24.05.2024**

Platnost zadání bakalářské práce: **21.09.2025**

Ing. Ondřej Kubíček podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Acknowledgement / Declaration

I would like to thank my supervisor Ing. Ondřej Kubíček for guiding the creation of this thesis. I would also like to thank the authors of the OpenSpiel game library for providing a framework for my work. Computational resources were provided by the e-INFRA CZ project (ID:90254), supported by the Ministry of Education, Youth and Sports of the Czech Republic. Last but not least I want to thank my family for their support during my studies.

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague date 24.05.2024

.....

Abstrakt / Abstract

Regularizace odměn se ukázala jako užitečná technika pro algoritmy posilovaného učení určené k řešení her s neúplnou informací. Jeden takový algoritmus používající tuto techniku je nedávno vyvinutý algoritmus Regularizovaná Nashova Dynamika (RNaD), který dosáhl výsledku na úrovní expertních hráčů ve hře Stratego. Studie těchto technik se zatím ale zaměřovala na dvouhráčové hry s nulovým součtem a není zřejmé, jestli tyto techniky budou užitečné i v širší skupině her jako jsou vícehráčové hry či hry bez nulového součtu. Proto je cílem práce najít obecněji použitelné rozšíření těchto technik a použít ho k upravení již existujících algorithmů pro použití v novém typu her. Efektivnost těchto úprav je ukázána na několika experimentech provedených na hrách typu pronásledování-únik.

Klíčová slova: hry s neúplnou informací; hry bez nulového součtu; vícehráčové hry; posilované učení; hry pronásledování-únik; regularizace odměn; Nashovo ekvilibrium; Regularizovaná Nashova dynamika.

Překlad titulu: Využití metody regularizace odměn ve hrách s neúplnou informací

Reward regularization proved to be a powerful technique in reinforcement learning algorithms for solving imperfect information games. One such algorithm using this technique is the recently developed Regularized Nash Dynamics (RNaD), which achieved a human-expert level performance in the game Stratego. However, research about this technique has focused on two-player zero-sum games, and it is currently unknown if this technique proves useful even in a broader class of games like general-sum or multiplayer games. Hence, this work aims to devise a more generally applicable extension of these techniques and modify the developed algorithms into a new class of games. The effectiveness of these modifications is shown in several experiments on pursuit-evasion games.

Keywords: imperfect information games; general-sum games; multiplayer games; reinforcement learning; pursuit-evasion games; reward regularization; Nash equilibrium; Regularized Nash Dynamics.

Contents /

Tables / Figures

Chapter 1
Introduction

Regularization is a well-known technique in machine learning, which is used to alter the learned probability distribution to prevent overfitting. In the context of reinforcement learning in games, such a technique is used to prevent excessive exploitation, thus making the learning more robust [1–2]. It has also been studied, that regularizing reward yields beneficial results [3–4], which has been used as a basis for current state-of-the-art algorithms for solving imperfect information games [5].

However, all the achieved results apply only to two-player zero-sum games. This is a case of games, where the loss of one player must result in a gain for the other player. Some examples of these games include rock-paper-scissors, chess, Go, or heads-up Poker. Games, where this does not hold, are called general-sum. One example of such a game is Blackjack because the dealer is not a player in the game, but rather a chance element with a fixed strategy. Other examples are certain cooperative games, such as Prisoners´s Dilemma. Thus, the achieved results need not apply to these types of games, and neither they need to apply to multiplayer games.

The goal of this work is to devise an extension to the used regularization techniques, that would be applicable to a broader class of games. The key idea of this work is that by using domain-specific knowledge, the regularization can be changed to account for various relations between players rather than just two purely adversarial.

In this work, the regularization techniques will be evaluated on pursuit-evasion games. These games can represent a broad class of games, ranging from two-player zero-sum games to multiplayer general-sum games. This allows a comparison of eventual changes to algorithms with the unchanged version in both the instances where the current algorithms should work well and those where their performance is not guaranteed. Furthermore, this is a well-studied type of game in game theory, that is both popular as a benchmark [6–7] and useful for representing real-world problems from various fields such as security [8], autonomous aircraft control [9–10], orbital control [11–12] or military [13]. However, as no simulator of such games was publicly available, it was implemented in the public game theory library OpenSpiel [14] as a part of this work.

1.1 Outline

The second chapter explains the needed theoretical background for games as well as for used algorithms and metrics used to measure their performance. In the third chapter, certain details of the pursuit-evasion game simulator are presented. The fourth chapter presents the changes made to Regularized Nash dynamics. Experimental results are discussed in the fifth chapter. Finally, the sixth chapter presents the conclusion and possible future improvements.

Chapter 2
Theoretical background

2.1 Game definition

Although there are multiple formal representations of a game, this work operates with the FOSG formalism[15]. This formalism was chosen, because of a natural way to split the player observations into public and private parts, which may be difficult to achieve in other formalisms.

Definition 2.1. [15] A **factored-observation stochastic game** (FOSG, only referred to as game from this point) is a tuple $G = \langle \mathcal{N}, \mathcal{W}, p, w^{init}, \mathcal{A}, T, R, \mathcal{O} \rangle$ where:

$\mathcal{N} = \{1, ..., N\}$ for some $N \in \mathbb{N}$ is the **player set**.

$\mathcal{W}$ is the set of **world states**.

w^{init} is a designated **initial state**.

$p: \mathcal{W} \to 2^{\mathcal{N}}$ is a **player function** determining which players act in the given state.

$\mathcal{A} = \prod_{i \in \mathcal{N}} \mathcal{A}_i$ is the space of **joint actions of all players**, where each $\mathcal{A}_i$ is an arbitrary set of actions of player i. $\mathcal{A}(w) := \prod_{i \in p(w)} \mathcal{A}_i(w)$ is a set of all legal joint actions at given state, where $\mathcal{A}_i(w) \sqsubseteq \mathcal{A}_i$ is a non-empty set of legal actions of player i in state w. $\mathcal{A}_i(w) = \{noop\}$ is a symbol for no action, that is used for all $i \notin p(w)$

$T: \mathcal{W} \times \mathcal{A} \hookrightarrow \Delta\mathcal{W}$ is the **transition function** determining probability distribution over possible states to transition to after action is applied for each $w \in \mathcal{W}$ and $a \in \mathcal{A}(w)$. This function may be defined even in some states where $p(w) = \emptyset$. Such states are called chance nodes and are used to represent an environment outside player control affecting the game [1]. A world state with $p(w) = \emptyset$ and undefined $T(w, a)$ is called a **terminal state**.

$R: \mathcal{W} \times \mathcal{A} \hookrightarrow \mathbb{R}^{|\mathcal{N}|}$ is the **joint reward function**, so that $R(w, a) = ((R_i(w, a)_{i \in \mathcal{N}}))$ where $R_i(w, a)$ is reward for player i when all players perform joint action a at state w.

$\mathcal{O}: \mathcal{W} \times \mathcal{A} \times \mathcal{W} \hookrightarrow \mathbb{O}$ is the **observation function**, where $\mathbb{O}$ is the set of all possible observations. This is factored into **private observations** and **public observations** as $\mathcal{O} = (\mathcal{O}_{\text{priv}(1)}, ..., \mathcal{O}_{\text{priv}(N)}, \mathcal{O}_{\text{pub}})$. Moreover, $\mathbb{O} = \prod_{i \in \mathcal{N}} \mathbb{O}_{\text{priv}(i))} \times \mathbb{O}_{\text{pub}}$, where $\mathbb{O}_{\text{priv}(i))}$ are arbitrary sets of private observations of player i and $\mathbb{O}_{\text{pub}}$ are arbitrary sets of public observations. $\mathbb{O}_i = \mathbb{O}_{\text{priv}(i))} \times \mathbb{O}_{\text{pub}}$ denotes a set of all possible observations of player i. A function $\mathcal{O}(w, a, w') \in \mathbb{O}$ is assumed to be defined for every non-terminal $w, a \in \mathcal{A}(w)$, and w' with non-zero probability in $T(w, a)$. This function returns an element from the observation set described above if action a was taken at state w and state w' was reached. Here it is additionally assumed, that no received observations are forgotten throughout the game (referred to as perfect recall games). An observation received by player i at timestep t is denoted as $o_i^t \in \mathbb{O}_i$.

The game then proceeds as follows: It starts in the initial state w^{init}. In each state w, each acting player $i \in p(w)$ learns the legal actions $\mathcal{A}_i(w)$ (either directly receives this information from the environment, or deduces it on his own), and selects action

[1] As none of the games discussed in this work use chance nodes, this is mentioned only for formal completeness

$a_i \in \mathcal{A}_i(w)$. Once that is done for all acting players, the joint action $a = (a_i)_{i \in p(w)}$ is applied and new state w' is drawn from the distribution $T(w, a)$. Then an observation $\mathcal{O}_i(w, a, w') = (\mathcal{O}_{\text{priv}(i)}(w, a, w'), \mathcal{O}_{\text{pub}}(w, a, w')) \in \mathbb{O}_i$ is generated. This means, that each player receives information that is private to them (such as the history of the actions they took, the cards they are holding in Poker, observed positions around player agents, etc.) alongside public information available to all players (such as cards on the table in Poker, information about used tickets in Scotland-Yard, etc.). Finally, each acting player also receives a reward[2] $R_i(w, a)$. Then new state w' is marked as current state w, and this entire process repeats, until a terminal state is reached, and the game ends. The goal of each player is to maximize the sum of rewards $R_i(w, a)$ received during the game. [15]

Definition 2.2. [16] A **zero-sum game** is a game where the following holds for each state w and action a: $\sum_{i \in \mathcal{N}} R_i(w, a) = 0$.

In a two-player zero-sum case this means, that whenever a player receives some reward, the other player must receive the same reward but multiplied by -1.

Definition 2.3. [15] A **trajectory** $\tau = w^0 a^0 w^1 a^1 ... w^{t-1} a^{t-1} w^t \in (\mathcal{W}\mathcal{A})^* \mathcal{W}$ where $a^l \in \mathcal{A}(w^l)$ and $w^{l+1} \sim T(w^l, a^l)$ for each $l \in \{0, ..., t-1\}$ is a finite sequence of states and actions that occurred during the game. A **trajectory utility** for player i is defined as $u_i(\tau) = \sum_{l=0}^{t-1} R_i(w^l, a^l)$, which is a cumulative reward for player i across trajectory τ.

Definition 2.4. [15] A **history** h is such a trajectory, where $w^0 = w^{init}$. H is a set of all histories. This means, that it starts in the initial state w^{init}. It is assumed that $\mathcal{A}(h) := \mathcal{A}(w^t)$ where w^t is the last state in the history. Additionally, Z denotes a set of terminal histories z where the last state of the trajectory is terminal. When $h \sqsubseteq h'$ it is said that h' extends history h. The initial history, corresponding to the initial state w^{init} is denoted as h^{init}. Lastly, $|h|$ is the length of the history, which corresponds to the number of joint actions played in the history.

However, because the player receives only observations, and not directly states and actions of the environment, some histories may be indistinguishable by him. Two histories are indistinguishable, if the player played action a_i^l and received observation o_i^l at each time step l in these histories. Hence, it is possible to group histories, that are indistinguishable for player i into a single set.

Definition 2.5. [15] An **information set** (infoset) $s_i \in S_i$ is some set of histories, where all histories in the set are indistinguishable by player i. S_i denotes set of all possible infosets for player i. $s_i(h) : H \hookrightarrow S_i$ is an infoset function.

If the game is viewed by an outside observer, that does not have access to any player's private observations, only the public part of observations can be used to distinguish states. This introduces the notion of **public states** $s_0 \in S_0$, which group histories the same way as regular infosets, but use only the public part of observations. So, it can be viewed as a union of infosets, that share the same public information. Furthermore, several infosets can belong to the same public state, but any infoset belongs to exactly one public state because the information contained in the public part of the observation is a subset of information of the whole observation. Similarly, each history is contained in exactly one infoset for each player and by extension in exactly one public state.

$S_i(s_0)$ denotes all infosets of player i belonging to the same public state and $\mathcal{A}_i(s_i)$ denotes actions available to player i under infoset s_i, which must be the same as $\mathcal{A}_i(h)$ for any h, where $s_i = s_i(h)$. Otherwise, player i could distinguish these histories based on the different available actions.

[2] In all of the games discussed in this work, the information about the received reward is assumed to be implicitly available to the player.

2.2 Algorithms

Definition 2.6. [15] **Policy**[3] of player i is a mapping $\pi_i: S_i \hookrightarrow \Delta\mathcal{A}_i$ where $\pi_i(s_i, a_i)$ denotes the probability that player i plays action a_i under infoset s_i,while following policy π_i. Because of the infoset definition, this can also be used to define probability based on history as $\pi_i(h, a_i) := \pi_i(s_i(h), a_i)$, where the policy corresponds to policy in infoset s_i, that the history belongs to. $\pi_i(h)$ denotes the probability distribution over actions $a_i \in \mathcal{A}_i(h)$ for player i. A **strategy profile** is defined as $\pi = (\pi_1, ...\pi_N)$. So it is a joint policy of all players. Moreover, π_{-i} denotes a strategy profile without the policy of player i.

Definition 2.7. [15] **Reach probability** of history h under strategy profile π is defined as $P^\pi(h) = \prod_{h'aw \sqsubset h} \prod_{i \in \mathcal{N}} \pi_i(h', a_i)$, where a_i is player i part of joint action a. This means that it is the probability of reaching history h, if all players follow strategy profile π. This can be rewritten as $P^\pi(h) = P_i^\pi(h) \cdot P_{-i}^\pi(h) = \prod_{h'aw \sqsubset h} \pi_i(h', a_i) \cdot \prod_{j \in \mathcal{N}, j \neq i} \pi_j(h', a_j)$, which signifies each player contribution to reaching that history, where $-i$ are all players except player i.

Definition 2.8. **Expected utility**[15] of player i following strategy profile π is defined as $u_i^\pi = \sum_{z \in Z} P^\pi(z) \cdot u_i(z)$, where $u_i(z) = \sum_{l=0}^{|z|-1} R_i(w^l, a^l)$ is a trajectory utility of terminal history z. So, it is a mean over the expected return of terminal histories, that are generated when all players follow strategy profile π. A joint expected utility is defined as $u^\pi = (u_i^\pi)_{i \in \mathcal{N}}$.

Additionally, denoting $a_i^h(\tau)$ as the action played by i at history h in a given trajectory τ allows defining a **Q function** as $Q_i^\pi(h, a_i) = \sum_{z \in Z, a_i^h(z) = a_i, h \sqsubset z} P^\pi(z) \cdot u_i(z)$. The intuitive interpretation is that this is the expected utility of such terminal histories, where player i chooses action a_i at the last state of h, and then all players follow strategy profile π onwards. Similarly, we can define a **Value function** that does not assume a specific action played at history h as $V_i^\pi(h) = \sum_{a_i \in \mathcal{A}(h)} \pi_i(h, a_i) \cdot Q_i^\pi(h, a_i)$.

Furthermore, the Q-function and value function can be used to compute the advantage of playing action a_i at history h instead of following $\pi_i(h, a_i)$. This is called an **advantage function** defined as $A_i^\pi(h, a_i) = Q_i^\pi(h, a_i) - V_i^\pi(h)$.

Definition 2.9. [17] A **Nash equilibrium** π^* is such a strategy profile, where for all players i and any policy π_i' the following holds: $u_i^{\pi_i' \pi_{-i}^*} - u_i^{\pi^*} \leq 0$.

This means, that if other players continue playing according to the Nash strategy profile, no player is incentivized to change their policy.

Definition 2.10. [18] An **ϵ-Nash equilibrium** is such a strategy profile π^ϵ, where the following holds $u_i^{\pi_i' \pi_{-i}^\epsilon} - u_i^{\pi^\epsilon} \leq \epsilon$ for some $\epsilon > 0$.

This can be viewed as an approximation of Nash equilibrium, where each player can improve their utility by ϵ at most by deviating.

Note, that when playing, a player needs to choose actions by sampling the probability distribution. Thus, the actual policy played by player i can be seen as a deterministic (pure) policy, conditioned on the Nash equilibrium policy. So when playing each player conditions their acting policy independently based on their Nash equilibrium policy. This leads to the idea, that the concept of equilibrium may be generalized, by allowing the players to condition their acting policies differently.

[3] Policy in perfect information games is defined on states and not on infosets. However, this corresponds, because perfect information games are a subset of imperfect information games, where for each player, each history is in a different infoset.

The key idea is introducing some third-party influence with a known probability distribution, which gives each player a private signal, that they can condition their policy on (assumed to be the policy suggestion itself here). This influence will form an equilibrium, if the players cannot improve their expected return by deviating from the suggestion.

Example 2.1. A simple example of such influence could be illustrated in Prisoner´s Dilemma. This is a game with two players, which simultaneously have to choose an action. Both players have the option to cooperate with the other prisoner, or defect to the authorities. If both prisoners cooperate, they get shorter jail time, but if only one defects, he avoids prison at the expense of the other prisoner. This game only has a single decision point, in which neither player has received any observations. Hence, in the following notation states and observations will not be considered, because players choose actions only in this single decision point. In a standard version of the game, the rewards look as follows:

R((cooperate, cooperate)) = (-1, -1)

R((defect, cooperate)) = (0, -3)

R((cooperate, defect)) = (-3, 0)

R((defect, defect)) = (-2, -2)

In this game, a Nash equilibrium is that both players defect. This is caused by the fact, that neither player can improve their expected utility by deviating from this strategy. If a player were to deviate and choose to cooperate instead, his reward would be -3, compared to the -2 when choosing to defect. In contrast, both players cooperating is not an equilibrium, because either player can improve their expected utility (0 compared to -1) by choosing to defect instead.

The players can agree that a coin flip will be performed, and, in the case of heads, they will both cooperate, but, in the case of tails, they will both defect. Then the expected utility for both players will be $-1 \cdot \frac{1}{2} - 2 \cdot \frac{1}{2} = -1.5$. However, if a player chooses not to follow the suggestion and defect instead, the expected utility will be $0 \cdot \frac{1}{2} - 2 \cdot \frac{1}{2} = -1$. So this suggestion will not form an equilibrium, because either player can improve their expected utility by deviating from it.

Formal definition of this concept uses the following components:

1. p, which is a probability distribution over all possible strategy profiles.

2. $\sigma = (\sigma_1, ..., \sigma_N)$, where σ_i is a policy to policy map for player i. Specially, σ^* is such a map, where $\sigma_i^*(\pi_i) = \pi_i$ for any π_i and all players i. This mapping can be viewed as "accepting" the suggested strategy profile, whereas any other mapping may be deviating.

Definition 2.11. [18] A **Correlated equilibrium** of an n-player game is such a probability distribution p, where $\sigma^*(\pi)$ forms a Nash equilibrium for any π drawn from p. Similarly, as in the case of Nash equilibrium, it is said to be ϵ-**correlated equilibrium** if $\sigma^*(\pi)$ forms an ϵ-Nash equilibrium for any π drawn from p.

Note, that because the probability distribution p is known, it can allow players to not act independently of each other (hence the name correlated equilibrium). Furthermore, it can be shown that Nash equilibrium is just a special case of correlated equilibrium, where each player's signals are probabilistically independent [18]. It can also be shown, that in two-player zero-sum games correlated equilibrium corresponds to Nash equilibrium [19].

2.2.1 Performance metrics

Definition 2.12. [20] A **Best response** $\pi_i^{BR}(\pi_{-i}) = \text{argmax}_{\pi_i} u^{\pi_i \pi_{-i}}$ is a best response of player i against strategy profile π. So it is such a policy, that yields the best reward if other players follow strategy profile π_{-i}

Definition 2.13. [20] A player i **incentive to deviate** is defined as $\delta_i(\pi) = u_i^{\pi_i^{BR}(\pi_{-i})\pi_{-i}} - u_i^{\pi}$.

Here, it can be noted that, recalling the Nash equilibrium definition, it is required that no player has an incentive to change their policy and thus no incentive to deviate. Thus $\delta_i(\pi^*) = 0$ for each $i \in \mathcal{N}$, which means that in Nash equilibrium all players are playing their best response to each other. This also means that incentives to deviate can be used to measure the current distance from Nash equilibrium.

Definition 2.14. [20] **NashConv** of strategy profile π is defined as $\text{NashConv}(\pi) = \sum_{i \in \mathcal{N}} \delta_i(\pi)$.

Definition 2.15. [20] **Exploitability** of strategy profile π in two-player zero-sum games is defined as $\text{Exploitability}(\pi) = \text{NashConv}(\pi)\ /2$.

Both of these metrics are widely used to exactly measure the quality of a strategy profile found by algorithms [4, 21–22].

2.2.2 Counterfactual regret minimization

Counterfactual regret minimization (CFR) is an algorithm for solving imperfect information games. This algorithm is run in iterations, where a starting strategy profile is first initialized (usually uniformly) and then it is updated at each iteration. In the end, the mean of strategy profiles at all iterations is returned.

This algorithm works by computing the counterfactual value of an infoset, which is a sum of expected utilities of histories falling under this infoset weighted by their reach probability, if the given player is playing to reach them. Then counterfactual regret is computed for each action at all infosets. This value can be seen as an indicator, of how would the player's expected utility change, if he played the given action at the infoset, instead of following the current policy. If it is positive, the player utility would improve, and it would get worse, if the value is negative.

Definition 2.16. [23] **Counterfactual value** for infoset s_i, player i, under strategy profile π is defined as $v_i^{\pi}(s_i) = \frac{\sum_{h \in s_i} P_{-i}^{\pi}(h) \cdot V_i^{\pi}(h)}{\sum_{h \in s_i} P_{-i}^{\pi}(h)}$.

Note that player i contribution to the reach probability of infoset s_i is not considered here, so it is assumed that the player will play to reach that infoset. $v_i^{\pi}(s_i, a_i)$ is computed the same way, but it is also additionally assumed that player i plays action a_i at s_i.

Definition 2.17. [23] **Immediate counterfactual regret** of infoset s_i, action a_i at iteration T, where π^T is current strategy profile of the CFR algorithm at iteration T, is defined as $rg_i^T(s_i, a_i) = v_i^{\pi^T}(s_i, a) - v_i^{\pi^T}(s_i)$.

Definition 2.18. [23] A **Cumulative counterfactual regret** of player i, infoset s_i and action a_i at iteration T is defined as $RG_i^T(s_i, a_i) = \sum_{t=1}^{T} rg_i^t(s_i, a_i)$. If the action is not fixed, then the regret is computed as $RG_i^T(s_i) = \max_{a_i \in \mathcal{A}_i(s_i)} RG_i^T(s_i, a_i)$. **Average counterfactual regret** is defined as $RG_i^{T-}(s_i) = RG_i^T(s_i)\ /\ T$.

Definition 2.19. [23] An algorithm is said to have **no regret** if $RG_i^{T-}(s_i) \to 0$ as $T \to \infty$ holds for all players $i \in \mathcal{N}$ following strategy profile π found by the algorithm and all infosets s_i.

This condition means, that the regret at higher iterations eventually has to converge to 0, otherwise the average regret would be greater than 0. Furthermore, it was proven

that if $RG_i^{T-} \to \epsilon$ as $T \to \infty$, then the algorithm converges to some ϵ-correlated equilibrium for all players i [24]. By extension, this means that no-regret algorithms eventually converge to a correlated equilibrium. So, in two-player zero-sum games, they converge to Nash equilibrium since it coincides with correlated equilibrium.

In CFR, any such no-regret algorithm may be used to update the player policies at each infoset. One of the most common regret minimizers for this purpose is regret matching[23], which is used in this work as well. This algorithm uses cumulative regrets as weights, thus preferring actions with high cumulative regret. The process policy of player i at timestep T:

Step 1: Compute $RG_i^{T,+}(s_i, a_i) = max(RG_i^T(s_i, a_i), 0)$

Step 2: update policy as

$$\pi_i^{T+1}(s_i, a_i) = \begin{cases} \frac{RG_i^{T,+}(s_i,a_i)}{\sum_{a_i \in \mathcal{A}_i(s_i)} RG_i^{T,+}(s_i,a_i)} & \text{when } \sum_{a_i \in \mathcal{A}(s_i)} RG_i^{T,+}(s_i, a_i) > 0, \\ \frac{1}{|\mathcal{A}_i(s_i)|} & \text{otherwise.} \end{cases}$$

So actions have their probabilities assigned based on cumulative regret and, if all cumulative regrets are smaller or equal to zero, uniform policy is used instead. In the end, a mean of strategies at all timesteps is returned, which is no-regret [23].

In zero-sum two-player games, it is guaranteed that if the average counterfactual regret of the CFR algorithm is ϵ, then the resulting mean of strategy profiles forms a 2ϵ-Nash equilibrium [23]. This algorithm is run for T iterations and, as it is no-regret, convergence to Nash equilibrium is guaranteed as $T \to \infty$

This algorithm was adapted in many forms to solve games. However, for large games, it is not suitable, because, in its base form, it needs to build the entire game tree and keep the counterfactual regret information for all infosets. It is trained via self-play, which means that during training the algorithm updates the whole strategy profile, rather than just the policies of certain players.

2.2.3 Follow the regularized leader

Follow the regularized leader (FoReL)[25] is an algorithm that, similarly to CFR, is run in iterations and the mean of strategies at all timesteps is no-regret. This algorithm utilizes reward regularization.

Definition 2.20. [3] A **regularizer** $\phi_i(\pi_i)$, where $\pi_i \in \Delta\mathcal{A}_i$, is a function, that is assumed to be: 1) continuous and strictly convex on $\Delta\mathcal{A}_i$. 2) smooth on the relative interior of every face $\Delta\mathcal{A}_i$ (including $\Delta\mathcal{A}_i$ itself). Common choices for regularizer function are either L2 norm $\phi_i(\pi_i) = \sum_{a_i}(\mid \pi_i(a_i) \mid^2)$, or cross entropy $\phi_i(\pi_i) = \sum_{a_i} \pi(a_i) \log(\pi_i(a_i))$.

This regularizer function is used in the algorithm maximization objective. At each iteration T a value $y_i^T(h, a_i) = \sum_{t=1}^T Q_i^{\pi^t}(h, a_i)$ is computed for each action a_i. These values can be arranged into a single vector as $y_i^t(h) = (y_i^T(h, a_i))_{a_i \in \mathcal{A}_i(h)}$ The process of policy update is defined as function:

updatepolicy(h, t, i): $\pi_i^t(h) = \text{argmax}_{\pi' \in \Delta\mathcal{A}_i(h)} \langle \pi', y_i^t(h) \rangle - \phi_i(\pi')$, where $\langle \pi', y_i^t(h) \rangle$ is a dot product between the two vectors

In other words, policy for player i and history h is set up to such a policy π', that maximizes probabilities of playing the "good" actions, while using the regularizer function, to prevent too much exploitation and force to explore different actions. The run of the FoReL algorithm requires three parameters: the number of iterations T, the starting strategy profile π, and the function Q, which returns the state-action value (this is as an input here, because often it is an estimate rather than the exact Q function). It proceeds as follows:

runFoReL(T, π, Q):

$h = h^0$
for $t = 1$ to T **do:**
  while h is **not** terminal, **do:**
    Choose action a according to current π, apply it to current history h.
    Receive reward $R(h, a)$, get history h' drawn from $T(h, a)$
    for each $i \in \mathcal{N}$ **do:**
      compute $y_i^T(h, a)$
      $\pi_i^{new}(h) = \text{updatepolicy}(h, t, i)$
    $h = h'$
    $\pi = \pi^{new}$
return π

Note that this is just the theoretical version of the algorithm. In practical implementation, certain modifications are usually used (such as bounded length histories instead of terminal histories). Also certain components of the algorithm, such as computing the Q-function exactly or finding the $\text{argmax}_{\pi'}$ are typically intractable.

2.2.4 Regularized Nash Dynamics

Regularized Nash Dynamics (RNaD) is an algorithm utilizing reward regularization, which also runs in iterations. In the original work, it was proven, that in the case of two-player zero-sum games, this leads to eventual convergence to Nash Equilibrium in the last iterate [3]. This is unlike the previously mentioned algorithms and removes the need to keep the average strategy profile over the iterations. Furthermore, this algorithm was successfully applied to Stratego[5].

This algorithm uses FoReL for the dynamics step but uses a different method of regularization. Instead of the regularizer function, it introduced the concept of policy-dependent rewards based on KL divergence and regularization strategy profile. In more detail, the reward is transformed in such a way, that the difference in the probability of playing some action compared to the regularization strategy profile affects the reward for this action.

Definition 2.21. [3] A **KL-divergence** between two policies π_i, π_i' in history h while playing action a_i is defined as $D^{KL}(\pi_i, \pi_i', h, a_i) = \log(\frac{\pi_i(h,a_i)}{\pi_i'(h,a_i)})$.

Definition 2.22. [3] A **policy dependent KL-divergence reward** in two-player zero-sum games is defined as $R_i^{KL}(h, a, \pi, \pi^R) = R_i(h, a) - \eta \cdot D^{KL}(\pi_i, \pi_i^R, h, a_i) + \eta \cdot D^{KL}(\pi_{-i}, \pi_{-i}^R, h, a_{-i})$ for each player i, where π^R is a regularization strategy profile, $i \in p(h)$ is the currently acting player, and $\eta > 0$ is a chosen constant representing the regularization strength. $R^{KL}(h, a, \pi, \pi^R) = (R_i^{KL}(h, a, \pi, \pi^R))_{i \in \mathcal{N}}$

So, in a case where only one player acts in each state, the KL divergence formula of the currently acting player is subtracted from the reward and the formula for the other player is added. This preserves the zero-sum formula. Additionally, this reward is designed in such a way, that both players have their reward decreased, if the probability of playing such an action was much higher compared to the regularization strategy profile. This is done to prevent excessive exploitation and motivate to explore more, just like the regularizer function in FoReL

The basic run of the RNaD algorithm then proceeds in the following steps:

1. Transform the game rewards to be policy-dependent rewards $R^{KL}(h, a, \pi, \pi^R)$ as described in 2.22.

2. FoReL dynamics are run on the transformed game, where the rewards represent the regularizer function. This is done until convergence, or for K steps.

3. The strategy profile found by the FoReL dynamics is set as the new regularization strategy profile.

4. Repeat steps 1 to 3 for T steps.

When $T \to \infty$ the regularization strategy profile will converge to the the Nash equilibrium π^* of the game[3].

runRNaD(T, K, π, η):

Initialize regularization strategy profile π^R arbitrary (e.g. uniform)
for $t = 1$ to T **do:**
  Transform the game rewards from $R(h, a)$ to $R^R(h, a, \pi, \pi^R)$
  Estimate or compute Q
  Set $\pi =$ runFoReL(K, π, Q)
  $\pi^R = \pi$
return π^R

However, there remain two problems. Firstly, FoReL in base form requires solving argmax, which is typically difficult and often not possible. Thiscan be solved by changing the FoReL updatepolicy function. The key idea is that, instead of computing the argmax exactly, it is approximated using gradient ascent. First, a gradient used to update the policy needs to be computed, where the value for each action is the advantage function of this action weighted by the probability of playing that action: $\nabla\pi_i(h, a) = \pi_i(h, a_i) \cdot A_i^\pi(h, a_i)$. The update function is defined as

updatepolicy(h, t, i): $\pi_i^{new}(h) = (\pi_i(h, a_i) + \nabla\pi_i(h, a_i))_{a_i \in \mathcal{A}_i(h)}$.

This is an update commonly used in evolutionary game theory called replicator dynamics update.[26].

Another problem is the Q-function. Computing it exactly is intractable, but it can be estimated by known methods from reinforcement learning, such as the V-trace estimator[27]. This estimator estimates the value function for infoset by sampling trajectories according to some sampling policy μ and then recursively computes the estimate from the end of the trajectory to the beginning. This estimate is used to train a neural network representing the value function and the Stratego authors show how to estimate the regularized Q-function out of it [5]. The authors[5] modify this estimator for games, where different players take turns.

The final problem is that generally, it is impossible to store both policy and the value function in explicit tabular form. This has to be solved for scalability and the authors[5] used the following solutions:

1. Both the policy and the value function are represented using a neural network. Both of these networks take as an input an infoset and output policy/value for the infoset.

2. The network estimating value function is trained with L2 norm loss, using the value estimate generated by V-trace as ground truth.

3. The policy is trained using Neural Replicator Dynamics (NeuRD)[26] loss. This is a case of the previously mentioned replicator dynamics update, modified to be suitable for neural networks. This handles the actual dynamics of the environment (so is used instead of the runFoReL function in the exact version of RNaD).

Just like CFR, this algorithm is trained via self-play.

Chapter 3
Pursuit evasion games

Pursuit-evasion games are multi-player games, where each player controls a mutually exclusive subset of agents. These agents move in some environment, that is the same for all the players and can be represented with a directed graph. Furthermore, there exists at least one pair of agents, where one is pursuing the other. If agent i is pursuing some other agent j, it means that its goal is to end up in the same position j. Conversely, j has a goal to prevent this situation and it is said that j is evading i. In this work, only pursuit-evasion games played on a n-dimensional grid are considered.

For formal convenience, certain concepts important for pursuit-evasion games are denoted:

C is a set of all agents.

b_c^w is a tuple denoting position of agent c at state w. It is of a size equivalent to the number of dimensions of the grid and is assumed to be ordered as (x coordinate, y coordinate, z coordinate, ...).

$F(i)$ is an agent function, that returns a set of all agents owned by player i.

$\Omega(j)$ is a pursuit function returning a set of all agents agent j is pursuing, and $\Omega^{-1}(j)$ is an evasion function, returning a set of all agents agent j is evading.

3.1 Simulator rules

Exact rules of possible games represented by the used simulator are as follows:

1. All agents move on a 2-dimensional finite grid of size H $\times$ W.

2. Agents can move in 4 directions- up, down, left, right, by one step on the grid, as long as they stay within the boundaries of the grid. They are also allowed to pass their turn, not moving from their position. However, agents can only perform one pass action consecutively. This is done to prevent players from forcing a draw by not moving at all.

3. Each player can control multiple agents, but can only act with one of them per turn. This means that legal actions for a player in any given state are a union over the legal actions of their agents.

4. To make the game finite, the maximum trajectory length is specified, after which games are cut off with a draw.

5. Each agent will have some amount of tokens available, which is the same for all agents. These can be used for the agent to boost their move, by moving not one tile, but several tiles corresponding to the token value. Information, that some agent used a token this turn, is available to all agents, but neither the agent's position nor the direction of the move is revealed.

6. Players take turns acting, with player 1 starting, so $p(w) = \text{mod}(|\tau|, N) + 1$, where τ is the current trajectory.

7. Transitions are deterministic, which means that $T(w, a)$ has 1 probability for some w' and 0 everywhere else for any non-terminal w and any $a \in \mathcal{A}(w)$.

8. The game ends, when any agent j catches an agent out of $\Omega(j)$, with a victory of a player controlling that agent. Alternatively, the game ends with a draw, if the acting player has no legal actions available, or the trajectory limit is reached.

9. If a game ends with a draw, all players receive a reward of 0.

The goal was to make the simulator as parameterizable as possible. So the parameterizable properties of the game are

- Board size: Two integers H, W, where H is the number of rows of the grid and W is the number of columns.
- Number of players: An integer N, which means that the game player set will be $\mathcal{N} = 1, ..., N$
- Number of agents: A list of N integers $(C^1, ..., C^N)$ where C^i is number of owned agents of player i. It is assumed that these agents are ordered by indices and that they are assigned to players in ascending order. For example, in a two-player case where $C^1 = 3$ and $C^2 = 3$ it is assumed that player 1 owns agents $1, 2, 3$ and player 2 owns agents $4, 5, 6$.
- Agent goals: Arbitrary amount of rules of type (k, l, r^k, r^l), which are interpreted as agent k is pursuing agent l, where r^k is the reward of k for catching l and r^l is the reward of l for being caught by k. If the rewards are not specified, they are by default assumed to be $r^k = 1, r^l = -1$.
- Number of tokens: An integer, representing how many tokens each agent has available. $\mathcal{T}(c)$ denotes the current amount of tokens agent c has available.
- Token boost amount: An integer $\mathcal{M}$ defining, how much the tokens boost actions. The standard action move is multiplied by this value.
- Maximum trajectory length: One integer τ^{max}, which is the trajectory limit after which the game is terminated.
- Movement penalty: A float $\gamma \geq 0$ that serves to penalize players for taking turns and thus motivate them to end the game faster. This value is subtracted from the reward of the player who acted on this turn. Note that if $\gamma \neq 0$, the game is no longer zero-sum, and is instead transformed into a general-sum game.

So a set of legal actions, that agents can take in the game, can be described as a set of tuples, where each tuple represents movement caused by the action in the (x, y) order as: $\mathcal{A}^{legal} = \{(0,0), (1,0), (-1,0), (0,-1), (0,1), (\mathcal{M},0), (-\mathcal{M},0), (0,-\mathcal{M}), (0,\mathcal{M})\}$

3.2 Observation structure

Recalling the FOSG definition 2.1, each game has some observations, which are how players perceive the environment. In pursuit-evasion games, it is assumed that all players perceive the environment through their agents, who can see their immediate surroundings in the directions where they can move. Furthermore, it is assumed that players know the current location of their agents, and which player is acting. Specially, in the case of this simulator, there is also the token info publicly available. So if any agent j uses a token, the info that agent j used a token becomes a part of the public part of the observation, but not which move the agent took and the current location of the agent.

As defined before 2.5, infosets are also required. As a set of histories would be impractical to store, infosets are instead simulated with information states that contain all the information currently known to the player. This information can be used to group histories into infosets. The information here corresponds to all actions taken

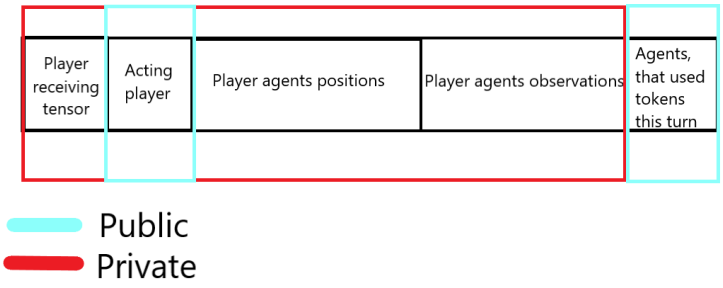


Figure 3.1. Structure of player observation factored into private and public parts

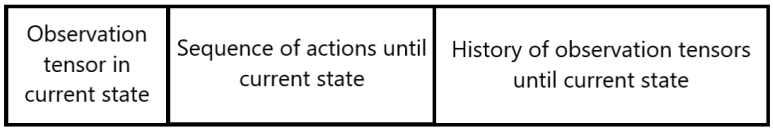


Figure 3.2. Structure of the information state

and observations received by the player throughout the game. Figure 3.1 shows the structure of the observation received decomposed into private and public parts and Figure 3.2. shows the structure of the information state.

The size of the position and observation components depends on the amount of agents owned by a player. However, it is expected that each player receives the same amount of information. Furthermore, in this simulator agents do not need to own the same amount of agents. For these reasons, it is necessary to introduce the concept of invalid position and invalid observations, which will be used to pad the observations to a fixed size. The value (-1, -1) is reserved for invalid position (-1 for both the x and y coordinates), and the value (-2, -2, -2, -2) for invalid observation (-2 in place of each observation performed by an agent).

Another important thing to note here is that the only part of the observation dependent on the action taken is the information about token action. Hence, it is defined even if no action was taken yet (in the initial state). This means that the information state is defined even in the initial state, and it is equivalent to the observation there.

3.2.1 Perceiving position

As stated before, each agent perceives the content of neighboring surroundings in the directions where he can move. In this simulator, this means the 4 neighboring tiles in directions up, down, left, and right. The received information signifies the content of the tile, and the possible values received are:

- -1 if the tile is out of bounds position.

- 0 if the tile is unoccupied
- index of agent c, if any agent c is currently occupying the tile.
- -2, which is reserved for invalid observation used for padding.

Because the agents perform 4 observations, the complete observation of an agent will be a 4-tuple (so it can be seen why the invalid observation is (-2, -2, -2, -2)).

So denoting $x(b_c^w)$ as the x coordinate of the position and $y(b_c^w)$ as the y coordinate the value of agent observation can be described as follows:
$o(b_c^w) = (o^l(x(b_c^w)+1, y(b_c^w)), o^l(x(b_c^w)-1, y(b_c^w)), o^l(x(b_c^w), y(b_c^w)-1),$
$o^l(x(b_c^w), y(b_c^w)+1))$, where:

$$o^l(b^w) = \begin{cases} -1 & \text{when } x(b_c^w) < 0 \vee x(b_c^w) \geq \mathrm{H} \vee y(b_c^w) < 0 \vee y(b_c^w) \geq \mathrm{W}, \\ c & \text{when } b_c^w = b^w \text{ for any } c \in C, \\ 0 & \text{otherwise.} \end{cases}$$

.

3.2.2 Conversion to tensor

So far only the information state and observations were discussed, however, as the information state is intended to be used as neural network input, both of these need to be converted to a tensor. So far all components of infoset were assumed to be represented with integers. However, neural networks do not deal well with categorical data, because they learn by assigning weights. For example, assume two actions, that are encoded as 4 and 1. The first action could be viewed by the network as 4 times better than the action encoded as 1 (because it is multiplied by the learned weight), but this does not need to be the case. Hence, it is necessary to convert the observations and information states to a structured representation.

The representation chosen for this purpose is one-hot encoding. In this representation, any value is encoded as a vector of size equal to the number of possible values for this element, where 1 is at the position corresponding to the encoded value, and 0 is everywhere else. For example, assuming there are four possible actions, action 1 would be encoded as 01000, and action 4 as 00001.

However, there remains the problem of encoding negative values, which are often used in the infoset to signify invalid/no value. This was solved by adding to the encoded value such a number, that the lowest possible value of the element with this number added is 0. For example, assume 5 possible actions, which are encoded in the range $\{-1, ..., 4\}$. Then 1 is added to each stored value before encoding it, so the possible action range becomes $\{0, ..., 5\}$, where one-hot encoding is well defined.

Furthermore, actions need to be encoded as well. Each action can be split into two parts that need to be encoded. First, the actual move that was taken, has 9 possible values. Second, the index of an agent performing the move, which has $\sum_{i \in \mathcal{N}} C^i$ possible values. These parts are encoded separately from each other and then put together in a manner distinguishing between the two to form the encoding of the action. Also, -1 is reserved for invalid action, which is used to signify that the player did not act at that timestep.

3.2.3 Observation tensor example

Assume the following state of board:

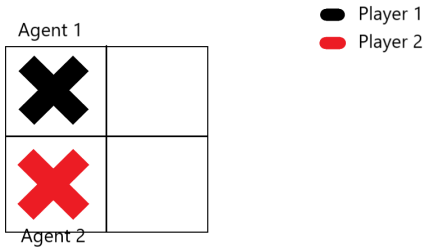


Figure 3.3. Small instance, where observation tensor can be visualized

Additionally assume, that player 2 is currently acting, and that agent 1 did not use a token last turn. In such a case, the observation tensor received by player 2 will look like this (the actual tensor received is from top to down in the picture):

01 001 001 010 00010 01000 01000 00100 100

Component	Tensor part	Description
Receiving player		Player receiving the tensor. Player 2 here.
Acting player		Player acting in this turn. Player 2 here
Player agents´ positions		Positions of agents owned by the player in the row-column order. Player 2 owns only agent 2, whose position is (1,0).
Player agents´ observations		Observations of agent 2 in order: tile up, tile down, tile left, tile right Here (1,-1,-1,0) because agent 1 is on tile up, tile right is empty and the other tiles are out of bounds.
Token info		Index of an agent, that used a token last turn. -1 here, because no token was used.

Table 3.1. Visualization of the observation tensor in a small board.

This is for a case where both players have the same amount of agents. Otherwise, it would be padded with invalid data for players owning fewer agents, than is the maximum amount of agents owned by a player.

3.2.4 Tensor sizes

Denoting $C^{max} = \max_{i \in \mathcal{N}}(C^i)$, which is the maximum amount of agents owned by player, size of components of the observation tensor is:

name	size(in elements)
Player receiving tensor	N
Acting player	$N + 1$
Player agents positions	$(\mathrm{H} + 1 + \mathrm{W} + 1) \cdot C^{max}$
Player agents observations	$4 \cdot (\sum_{i \in \mathcal{N}} C^i + 3) \cdot C^{max}$
Agents, that used token this turn	$\sum_{i \in \mathcal{N}} C^i + 1$

Table 3.2. Size of individual components of observation tensor.

From the table 3.2 the size of a single observation can be seen as a sum of the sizes of its components. This will be denoted as $|o^{oh}|$.

Furthermore, recalling that the action components are encoded separately, the size of an encoded action can be computed as:

$|a^{oh}| = 9 + \sum_{i \in \mathcal{N}} C^i + 1 = 10 + \sum_{i \in \mathcal{N}} C^i$. Note that an additional + 1 is added to the equation because an invalid action needs to be encoded as well.

Additionally, recalling the structure of information state 3.2, information state tensor at timestep T, corresponding to the player information of histories contained in $s_i(h)$ of player i and history h ending in state w^T, can be formally described as: $((o_i(w^T)), (a_i^0, ..., a_i^{T-1}) \ , (o_i(w^{init}), ..., o_i(w^{T-1}))$ if $|h| > 0$, else $o_i(w^{init})$, where a_i^t is the one-hot encoded action at timestep t.

Note that without the bounded trajectories condition the information state would have a variable (potentially infinite) length. However, with this condition, the size of the information state is fixed as $|o^{oh}| + \tau^{max} \cdot (|a^{oh}| + |o^{oh}|)$.

Chapter 4
RNaD changes

As mentioned before, RNaD is an algorithm that has so far been tested only in a two-player zero-sum setting, and its theoretical guarantees were proven to apply there. However, in this work, several changes to the algorithm were evaluated.

4.1 Default implementation in an n-player case

The KL divergence transformed reward used in RNaD is defined only for a two-player zero-sum setting. Thus the first step in the experiments was extending the KL-divergence regularization to apply to the multiplayer setting without changing the underlying algorithm. This was done for reference to observe, how the slightly changed algorithm performs in a multi-player setting, and then compare it with the more altered versions.

In the definition 2.22 the regularization term of the acting player is subtracted and the term of the other player is added. The used extension follows similar logic in a multi-player setting, where the term of the acting player is subtracted and the terms of all other players are added. Note that in a two-player case, this preserves the zero-sum property (it is equivalent to the reward in the original definition), but in a multi-player setting, this transforms the game from zero-sum into a general-sum game. So for generalization of the reward into a multi-player setting a regularization constant for player j first needs to be defined as

$$\rho_j^R(h) = \begin{cases} -1 & \text{when } j \in p(h) \\ 1 & \text{otherwise} \end{cases}$$

Then $R^{KL}(h, a, \pi, \pi^R) = (R_i(h, a) + \eta \cdot \sum_{j \in \mathcal{N}} \rho_j^R(h) \cdot D^{KL}(\pi_j, \pi_j^{reg}, h, a_j))_{i \in \mathcal{N}}$.

4.2 Entropy regularization

The first change tested was inspired by a paper concerning abstracting imperfect information away from two-player zero-sum games [4]. There, it was proven, that even other kinds of regularization allow gradient descent algorithms to converge to a stable point[4]. However, such a point will not be the equilibrium of the original game, but instead some altered equilibrium. The most prominent regularization type they used was entropy regularization, which is used even in a single-player setting to promote exploration [28]. Furthermore, the authors have shown that the choice of the regularization weight η in entropy regularization affects the distance from the Nash equilibrium of the original game. However, it was also shown, that for $\eta = 0$ the algorithms will not converge [4].

Definition 4.1. An **Entropy** [4] of a policy π_i at history h is defined as $D^{ENT}(\pi_i, h) = -\sum_{a_i \in \mathcal{A}_i(h)} \pi_i(h, a_i) \cdot \log \pi_i(h, a_i)$

Definition 4.2. A **Policy dependent entropy reward** is defined as $R^{ENT}(h, a, \pi, \eta) = (R_i(h, a) - \eta \cdot \sum_{j \in \mathcal{N}} \rho_j^R(h) \cdot D^{ENT}(\pi_j, h))_{i \in \mathcal{N}}$, where $\rho^R(h)_i$[1] is a regularization constant (as defined in 4.1).

However, the standard RNaD loop of swapping the regularization policies is not applicable here. There is the fact, that η affects the distance from the equilibrium, bringing the equilibrium closer to the original one as it is decreasing [4]. This is used in the altered algorithm, where η is annealed with increasing iterations, thus decreasing the distance from the Nash equilibrium. However, this may come at the cost of stability, since $\eta = 0$ leads to divergence.

The modified algorithm thus functions as follows: runRNaDEntropy(T, K, π, η):

Initialize $\eta^{cur} = \eta$
for $t = 1$ to T **do:**
  Transform the game rewards from $R(h, a)$ to $R^{ENT}(h, a, \pi, \eta^{cur})$
  Estimate or compute Q.
  Set π = runFoReL(K, π, Q)
  $\eta^{cur} = \frac{\eta}{\log(t+1)}$
return π

The logarithmical annealing was chosen, to prevent η from quickly getting too close to 0.

4.3 Relation of players change

The second tested extension did not change the algorithm loop, but instead changed rewards passed to V-trace. In the modified V-trace described in 2.2.4, only the rewards of player i are considered in the estimate for i. This works in a two-player zero-sum case, because $R_{-i}(h, a) = -R_i(h, a)$ holds for any h, a. Hence, the rewards of the other player implicitly affect the estimate, even when passing only the reward of player i to V-trace.

However, moving into an n-player general-sum setting, such an assumption does not need to hold. For example, actions that result in rewards (1, -1) and (1, -20) are considered equivalent for player 1. This could of course be the intended effect of the reward function. However, intuitively the second action is better because player 1 receives the same reward, whereas player 2 is punished for this action more. The key idea behind these changes was to test, what would happen, if the player was forced to account for the rewards of other players as well. Specifically, the player would seek to maximize his reward, while minimizing the reward of all players adversarial to him.

Because of these reasons, a concept of relation function was introduced.

Definition 4.3. A **relation function** of player i against player j is defined as:

$$\theta_i(j) = \begin{cases} 1 & \text{if } i = j \\ -1 & \text{when there exists a pursuit-evasion relation between agents of } i \text{ and } j \\ 0 & \text{otherwise} \end{cases}$$

And the reward passed to V-trace as a reward for player i is defined as $R_i^\theta(h, a) = \sum_{j \in \mathcal{N}} R_j(h, a) \cdot \theta_i(j)$. This reward considers the rewards of other players in such a

[1] The $-\eta$ is here, because of $-\sum$ in the entropy formula. So a multiplication by -1 is necessary for the desired behavior.

way, that the player wants to maximize the reward of the allied player (only self in this version), minimize the reward of adversarial players, and ignore the reward of neutral players.

This version of the relation function was created specifically for pursuit-evasion games, but the implementation works with any relation function, as long as the environment provides it. If the relation function is not provided, all other players are assumed to be adversarial.

It can also be noted that in the case of two-player zero-sum games, this reward is not exactly equivalent to $R_i(h,a)$, but it is instead multiplied by two because $R_i^\theta(h,a) = R_i(h,a) - R_{-i}(h,a) = 2 \cdot R_i(h,a)$. However, if this value is divided by the number of players of the game N, it is exactly equivalent in two-player zero-sum games to the original reward. There have been tested both changes that divided the reward, and those that did not.

4.3.1 Regularization with the relation function

Additional change tested was implementing this relation function into regularization, in place of the standard regularization constant. However, using this function as is would lead to reversing the desired behavior, because the regularization term would be added to the acting player and subtracted from adversarial players (in the case of only one player acting each turn). This would lead to the fact, that the players would be more motivated to choose actions that have a higher probability than the regularization policy. As such, to achieve the intended behavior of the algorithm, the output of this function also needs to be multiplied by -1. The resulting reward of the regularized game looks like this:

$R_\theta^{KL}(h,a,\pi,\pi^R) = (R_i(h,a) - \eta \cdot \sum_{j \in \mathcal{N}} \theta_i(j)) \cdot D^{KL}(\pi_j, \pi_j^R, h, a_j))_{i \in \mathcal{N}}$.

So this reward is similar to the one described in 4.1, however, the terms of players neutral to the acting player are multiplied by 0 and have no impact on the reward. It can be noted, that these two rewards are exactly equivalent if all players in the game are adversarial.

Also, the fact that regularization terms of neutral players have no impact on the regularization could potentially cause issues. This could be solved, by assigning 1 to even neutral players instead of 0 in the original relation function. For this reason, a regularization function alternative was devised as

$$\delta_i(j) = \begin{cases} -1 & \text{when there exists a pursuit-evasion relation between agents of } i \text{ and } j \\ 1 & \text{otherwise} \end{cases}$$

.

And the reward is then defined as $R_\theta^{KL}(h,a,\pi,\pi^R) = (R_i(h,a) - \eta \cdot \sum_{j \in \mathcal{N}} \delta_i(j)) \cdot D^{KL}(\pi_j, \pi_j^R, h, a_j))_{i \in \mathcal{N}}$.

Chapter 5
Experiments

5.1 Testing instances

The smallest testing instance used was a perturbed variant of rock-paper-scissors, which was presented in the paper discussing reward regularization [4].

Example 5.1. The perturbed rock-paper-scissors (RPS) is a game with two players acting simultaneously and only a single world state. Each player has three actions available, and that is rock, paper, or scissors. Because the game only has a single decision point where players choose action, and players have no information available in this decision point, the notion of states and infosets can be omitted in the following description. The game follows the logic that paper beats rock, rock beats scissors and scissors beat paper. Both players playing the same action results in a draw. Furthermore, in this perturbed variant, if either player plays a scissors action, the rewards for both players is multiplied by two. So the reward function of the game is as follows:

$R(h, a) = (0, 0)$, where h is the only decision point in the game, for any joint action a, where both players choose the same action (so $a_i = a_{-i}$). This is the case for actions (rock, rock), (paper, paper), and (scissors, scissors).

R(rock, scissors) = (2, -2) and R(scissors, rock) = (-2, 2).

R(rock, paper) = (-1, 1) and R(paper, rock) = (1, -1).

R(paper, scissors) = (-2, 2) and R(scissors, paper) = (2, -2).

The Nash Equilibrium of this RPS game for both players is playing scissors with 0.2 probability, and the other actions with 0.4 probability [4]. In the standard rock-paper-scissors a uniform probability distribution over the actions would form a Nash equilibrium. This perturbed variant was used, because algorithms often initialize the starting strategy profile uniformly, which would lead to instantly "guessing" the Nash equilibrium without any learning in the standard variant.

Additional testing instances were pursuit-evasion games generated from the simulator. In total, we have created six pursuit-evasion instances[1]. This section contains the motivation behind including each instance as well as images visualizing each instance, where agents will be differentiated by color to signify, which player they belong to. Additionally, arrows will signify pursuing relations and there will be rewards written next to them (if no reward is written, the default (1,-1), where 1 is the reward for the pursuing player and -1 is the reward for the evading player, is assumed). Certain boards have been tested both in 2-player and multiplayer variants. Additionally, all the multiplayer instances have the movement penalty γ set to 0.1, unless explicitly stated otherwise. This was done to test performance on a general-sum setting, but one instance that is a multiplayer zero-sum game was also included. The starting player on all tested instances is player 1.

[1] One instance was tested twice. First as a zero-sum and then as a general-sum game

5.1.1 Small boards

The first two instances were played on small boards. This allows comparison with already established algorithms, such as CFR, and exact evaluation of algorithm performance by computing NashConv (or exploitability). In these instances, agents have 2 tokens, and $\mathcal{M} = 1$. So token actions result in the same move as the usual action but also announce that the token was used. τ^{max} is set to 5.

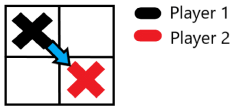


Figure 5.1. Small board

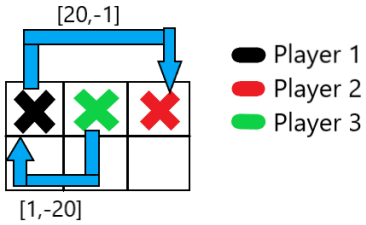


Figure 5.2. Three player version of the small board

5.1.2 Medium-sized board

Another instance was chosen in such a way, that it would be small enough for fast training and testing, but also too large to be solved by base form CFR or compute NashConv. This was a three-player instance played on a 3x3 board. Each player owns two tokens and $\mathcal{M} = 2$. τ^{max} is set to 10.

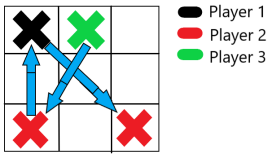


Figure 5.3. Three player medium-sized board.

5.1.3 Board with many goals

The next included instance was used only in a two-player version. The main purpose of this board was to include an instance of non-trivial size, which is heavily skewed in favor of player 2 and has many pursuit-evasion relations. This instance is played on a 4x4 board, with each player owning 3 agents. Each agent has 3 tokens and $\mathcal{M} = 2$. Each agent of player 1 is pursuing one agent of player 2, but never the one on the same row, and evading all other agents of player 2. (The pursuing relations for player 2 are omitted from the visualization to prevent illegibility.) τ^{max} is set to 20.

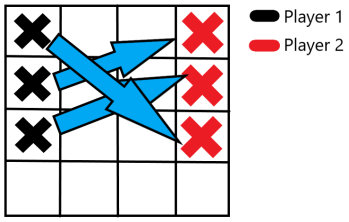


Figure 5.4. Board with many goals. Each agent of player 1 is also evading all agents of player 2, that he is not pursuing. This is omitted from the image to prevent illegibility.

5.1.4 Large board

The last used instance was the largest instance included, where the algorithm performance at scale was tested. This is a multiplayer instance with five players to evaluate scalability with players. Additionally, this instance was tested both with and without the movement penalty. The version without the movement penalty was included to evaluate performance in a multiplayer zero-sum setting. This instance was played on a 6x6 board. Each agent owns 5 tokens and $\mathcal{M} = 4$. τ^{max} is set to 30.

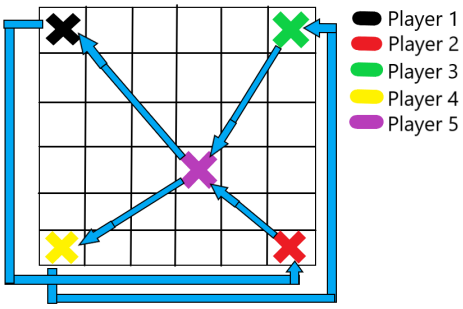


Figure 5.5. Large board with five players

5.1.5 Tensor sizes

Instance name	observation tensor size	information state tensor size
Perturbed RPS	1	1
Small board	34	529
Small board multiplayer	42	577
Medium-sized board	84	2214
Board with many goals	150	5730
Large board	63	5832

Table 5.1. Sizes of observation/information state tensors in the used instances

5.2 Heuristic player

Randomly playing opponents may not provide a strong baseline to test the algorithm's performance against. Because of this, a simple heuristic player was implemented, that serves as a stronger baseline opponent than the random one.

This heuristic player is more likely to play actions, that bring his agents closer to the agents they are pursuing and further from the agents they are evading. This player stores the last-seen position of adversarial agents. Assume that the heuristic player is playing as player i. Then each action is assigned a score, in such a way, as to minimize the distance from agents in $\Omega(i)$ and maximize the distance from agents in $\Omega^{-1}(i)$. In the end, the softmax function is used to convert the vector of scores for each action to a probability distribution.

Additionally, if an action would result in an agent from $\Omega(i)$ being caught, it has an assigned score of ∞, to ensure that it will be played. Analogically, this works with agents from $\Omega^{-1}(i)$ and $-\infty$ score to ensure not playing those actions [2].

So the player stores two types of information: $B^g = (B^g_c)_{c \in F(i)}$, where B^g_c is a set of last seen positions of agents $l \in \Omega(c)$, and $B^f = (B^f_c)_{c \in F(i)}$, where B^f_c is a set of last seen positions of agents $k \in \Omega^{-1}(c)$.

For the following algorithm, the Manhattan distance between two agent positions is defined as $d^M(b^w_k, b^w_l) = \mid x(b^w_k) - x(b^w_l) \mid + \mid y(b^w_k) - y(b^w_l) \mid$, where $x(b^w_k), y(b^w_k)$ correspond to the x and y coordinates of position b^w_k respectively, and softmax function on vector w of length N as $\text{softmax}(w) = (\frac{e^{w_t}}{\sum_{u=1}^{N} e^{w_u}})_{t=1,\dots,N}$, where w_t is the t-th component of the vector w. The algorithm to get policy $\pi_i(h)$ at history h proceeds as follows:

Get set of legal actions $\mathcal{A}_i(h)$ at current history h
Initialize scores vector of length $|\mathcal{A}_i|$ with zeros.
for each $a_i \in \mathcal{A}_i(h)$ **do:**
  Advance to $h^{'}$ ending in state $w^{'}$ by applying a_i
  Receive observation tensor $o_i^{w^{'}}$
  for each agent $k \in$ player agents observations of $o_i^{w^{'}}$ **do:**
    if $k \in \Omega(c)$ **for** any $c \in F(i)$:
      update B^g_c with $b_k^{w^{'}}$
    if $k \in \Omega^{-1}(c)$ **for** any $c \in F(i)$:
      update B^f_c with $b_k^{w^{'}}$
    if $d^M(b_c^{w^{'}}, b) = 0$ **for** any $c \in F(i), b \in B^g_c$
      $\text{scores}_a = \infty$
      Move to the next action
    else if $d^M(b_c^{w^{'}}, b) = 0$ **for** any $c \in F(i), b \in B^f_c$
      $\text{scores}_a = -\infty$
      Move to the next action
  $\text{scores}_a = \text{sum}_{c \in F(i)}(\text{sum}_{b \in B^f_c} d^M(b_c^{w^{'}}, b) - \text{sum}_{b \in B^g_c} d^M(b_c^{w^{'}}, b))$
return softmax(scores)

5.3 Experiments description

The OpenSpiel framework was used for all of the experiments, including the implementations of CFR and RNaD. The modifications to RNaD also used the OpenSpiel implementation as a base.

[2] $\infty, -\infty$ are just here to signify that these actions will almost always/never be taken over other actions with finite score. In the actual implementation, this does not work for the softmax calculation and has to be handled differently.

Both training and testing require sampling random numbers, to realize sampling actions from a stochastic policy. In these experiments, one random number generator seed (100) was used for testing and 3 seeds for training. In the presented plots, the variance caused by the seeds will be represented by a confidence interval, or by plotting the values for all seeds separately, depending on how high impact the seed variance proved to have.

The algorithms were trained for 100000 iterations in all instances, where each iteration corresponds to sampling a batch trajectory and then a single dynamics update. The regularization strategy profile was swapped after each 250 iterations 10 times and then after each 1000 iterations for the rest of the training.

Because of minimal changes to the algorithm and no changes to the most time-intensive parts, the training times of changed versions were roughly the same as those of the base version. Hence, in the following table iterations per hour are stated only once. The training was performed on MetaCentrum hardware.

Board name	Iterations per hour
Perturbed RPS	600000
Small board	100000
Small board multiplayer	100000
Medium-sized board	25000
Board with many goals	6250
Large board	4000

Table 5.2. Approximate iterations per hour of RNaD on tested boards

Some of the experiments evaluated the players trained by the neural network in a play against different players. For those experiments, 10000 game runs were performed.

The original version of RNaD 2.2.4 and its extension into a multiplayer setting 4.1 will be denoted as RNaD KL in the following experiments.

5.4 Entropy regularization experiments

The first tested change was swapping the KL-divergence regularization for entropy regularization, as described in 4.2. As in the case of the regularization policy, the regularization term η was annealed after every 250 iterations 10 times and then after every 1000 iterations for the rest of the training. η was annealed logarithmically as $\eta^{cur} = \frac{\eta}{\log(t+1)}$, where t represents the number of annealing steps performed including the current step.

5.4.1 Perturbed RPS

First experiments were performed on the perturbed RPS game described in 5.1. This experiment served as a proof of concept of whether entropy regularization can work on small problems.

The achieved results proved to be much worse than RNaD KL. During the first iterations of the learning, one of the seeds had similar results to RNaD KL. However, after the 100th iteration, both the exploitability and the expected return started to diverge. A study of the found policies revealed, that the algorithm oscillated between two patterns. Either there was one dominant action with the highest probability assigned (not limited to one specific action), or the policy was rather close to a uniform policy. This happened for both players.

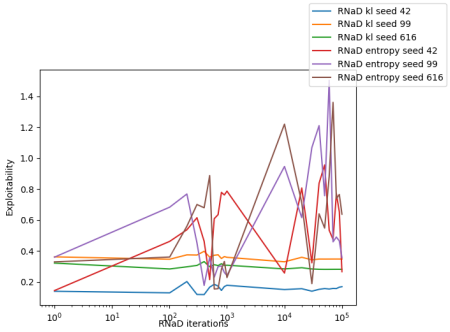


Figure 5.6. The exploitabilities of KL RNaD and entropy RNaD in the perturbed RPS game.

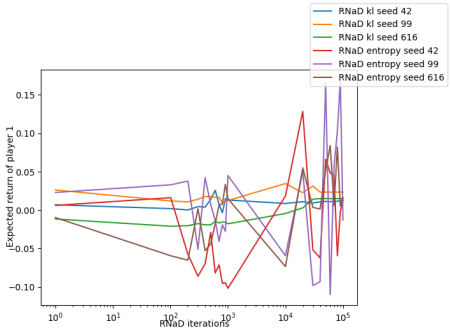


Figure 5.7. Expected return of player 1 of KL RNaD and entropy RNaD in the perturbed RPS game. Because it is a zero-sum game, the expected return for player 2 is equal to this value multiplied by -1

We assume that this was caused due to the η annealing being unstable, and causing the algorithm divergence.

5.4.2 Small board

Additional tests were performed on the two-player version of the small pursuit-evasion board 5.1.1. A very high oscillation in the higher iterations would result in the illegibility of the plot, Hence, the variance is represented here by a confidence interval with 0 as the lower bound.

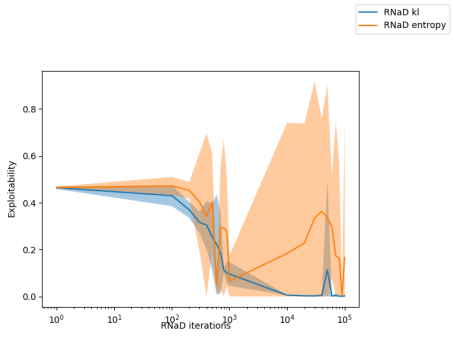


Figure 5.8. The exploitabilities of KL RNaD and entropy RNaD in two-player small board

.

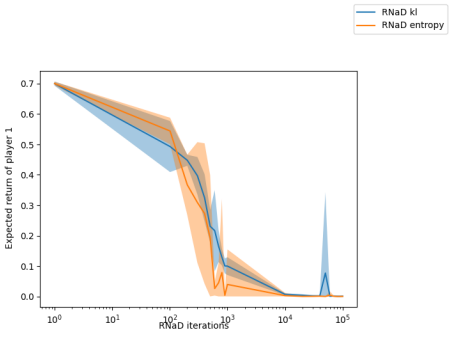


Figure 5.9. The expected return of KL RNaD and entropy RNaD in the two-player small board

.

The achieved results show that although both algorithms reached a similar expected return, the exploitabilities varied greatly. In the later iterations, all seeds of the entropy RNaD oscillated between 0 and 0.5 exploitability values. This would seem to confirm the

hypothesis made based on the RPS results. Here the algorithm starts to be unstable after roughly the 100th iteration and completely diverges after the 1000th iteration. Based on these results, we concluded that the entropy version of RNaD is only stable up to a certain threshold of η, where it starts to diverge. Moreover, this threshold is highly dependent on the specific instance and would have to be found experimentally. Also, because η here has an impact on the distance from Nash equilibrium, the algorithm can only achieve a certain distance from the equilibrium, dependent on the lowest stable η, and diverges upon further annealing. For these reasons, we concluded that entropy RNaD was not suitable for neural network training, and did not use it on larger instances.

5.5 Relation changes experiments

More changes were tested based on the player relation changes proposed in 4.3. Three main altered versions were tested.

The first version utilizes the proposed change to rewards 4.3, where the reward for each player is altered so that the rewards of adversarial players are subtracted from it as $R_i^\theta(h,a) = \sum_{j\in\mathcal{N}} R_j(h,a)\cdot\theta_i(j)$, where $\theta_i(j)$ is the relation function, which is 1 for the player himself, -1 for adversarial players and 0 for neutral players. Otherwise, this version is the same as the KL version. As noted in 4.3 in two-player zero-sum case the used reward will be $R_i^\theta(h,a) = 2\cdot R_i(h,a)$ for both players, so it is not exactly equivalent. This version is called RNaD relation rewards or, more concisely, RNaD rewards.

The second version also uses these relation rewards. Additionally, the relation function is used for regularization, where the regularization term of adversarial players is subtracted, the term of the current player is added, and the terms of neutral players are ignored. The reward is first transformed into $R_i^\theta(h,a)$ and then regularized, so the regularized reward is computed as $R_\theta^{KL}(h,a,\pi,\pi^R) = (R_i^\theta(h,a) - \eta\cdot\sum_{j\in\mathcal{N}}\theta_i(j))\cdot D^{KL}(\pi_j,\pi_j^R,h,a_j))_{i\in\mathcal{N}}$. This version is called RNaD multiplayer (RNaD mp).

Finally, the third and last tested version was designed to prevent two properties of RNaD mp. Namely, the inequality to RNaD KL in a two-player zero-sum case and some players' regularization terms being ignored. The reward $R_i^\theta(h,a)$ is divided by the number of players N, which results in equality to RNaD KL in two-player zero-sum games. Additionally, the alternative relation function δ 4.3, which is -1 for adversarial players, and 1 for neutral players and the player himself, is used for regularization. So the regularized reward is computed as $R_\delta^{KL}(h,a,\pi,\pi^R) = (\frac{R_i^\theta(h,a)}{N} - \eta\cdot\sum_{j\in\mathcal{N}}\delta_i(j))\cdot D^{KL}(\pi_j,\pi_j^R,h,a_j))_{i\in\mathcal{N}}$. This version is called RNaD δ.

When the algorithms were tested by playing against opponents, playing as each player of the instance was tested. However, for increased readability, only plots when playing as certain players are shown in this section. The rest of the plots, which were not deemed to carry any additional information, can be found in the appendices.

5.5.1 Perturbed RPS

As in the case of entropy RNaD, the first tests were performed on the perturbed RPS 5.1. Because it is a two-player zero-sum instance, RNaD δ is equivalent to KL, so it will not be shown on the plots (achieved results were identical). RNaD rewards had only minimal performance changes compared to RNaD KL. RNaD mp performance was much more unstable for the first 1000 iterations, where the found policies often had one

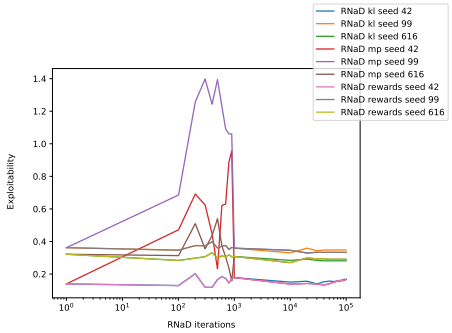


Figure 5.10. The exploitabilities of KL RNaD, RNaD mp and RNaD rewards in perturbed RPS. RNaD δ results were equal to RNaD KL.

.

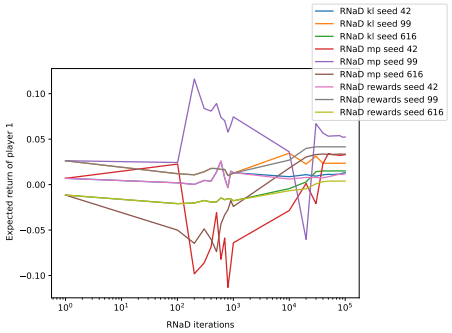


Figure 5.11. The expected return for player 1 of RNaD KL, RNaD mp and RNaD rewards in perturbed RPS. RNaD δ results were equal to RNaD KL. For player 2, the expected return is equal to this value multiplied by -1.

.

dominant action with a much higher probability than the other actions (usually paper or scissors). However, after the 1000th iteration, the results stabilized and were highly similar to those of RNaD KL.

5.5.2 Small boards

Additional testing was performed on the two-player small board instance of the pursuit-evasion game. RNaD δ is presented in the plots as well because unlike RNaD KL it did not have two exploitability spikes in one of the seeds each time. This was caused by giving too high a probability to the pass action in a state, where it would result in a defeat. However, this outlier happened only once for each of the two seeds and immediately disappeared again, so we assume it was a learning error and its absence in RNaD δ performance is caused by random factors rather than the algorithm itself. Otherwise, the performance of the RNaD algorithms was very similar.

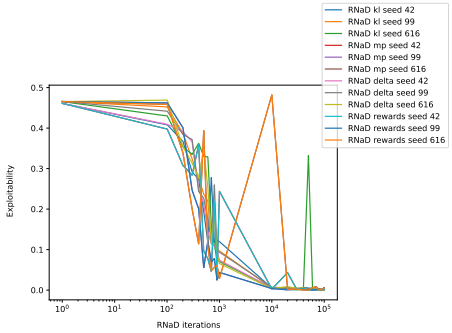


Figure 5.12. The exploitabilities of all tested RNaDs in the two-player small pursuit-evasion game.

.

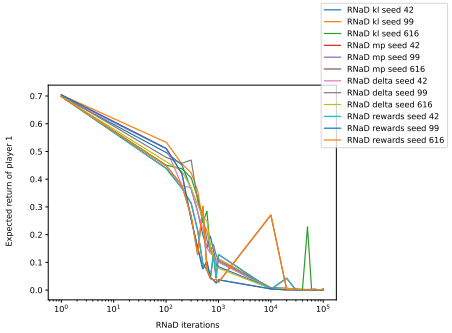


Figure 5.13. The expected return for player 1 of all tested RNaDs in the two-player small pursuit-evasion game.

.

So, in the tested small instances of two-player zero-sum games, the changed algorithms performed similarly or better than RNaD KL. Furthermore, all the algorithms were converging towards an expected draw. With this proof of concept more successful than the entropy changes proved to be, we have moved on to testing these changes in a multiplayer setting.

There is the three-player version of the small board, which is sufficiently small to compute NashConv 5.1.1. Even in this case, all the algorithms yielded similar results, with RNaD KL having a slightly higher variance between the seeds. Eventually, all the algorithms converged to a low NashConv (around 0.1) and a draw.

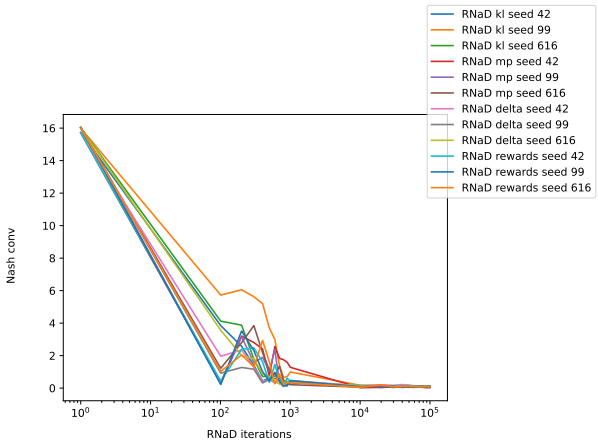


Figure 5.14. NashConv of the tested versions of RNaD in the three-player small instance of pursuit-evasion game.

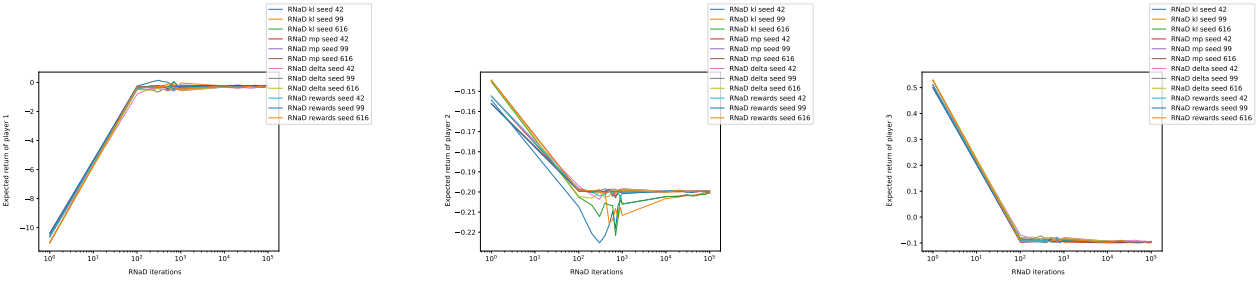


Figure 5.15. The expected returns of the tested RNaD versions in the three-player small pursuit-evasion board. From left to right is the expected return for players 1,2, and 3 respectively.

So far, the algorithms have been successful in the small pursuit-evasion board. A draw was expected in both boards because each player owns only 1 agent, and token movement is only 1. This means, that any player will see the pursuing player in the observation of the surrounding tiles. As a result, rationally playing players will always avoid their pursuer. However, against randomly playing opponents a weakness of the algorithm was perceived. Sometimes, when the algorithm would make a move unexpected by the algorithm (the found strategy profile has only a very small probability of that move), the game would progress into a state, where the algorithm has a very badly trained policy. In such a state, the algorithm would often take a move, that prevents the player victory or even results in a defeat. This caused a higher variance and lowered victory rate against random opponents. RNaD KL generally had a higher variance, especially when playing as player 2 (this is apparent even from the self-play results for player 2). All the altered RNaDs once again performed similarly, whereas RNaD KL performed better as player 1, but worse as player 3.

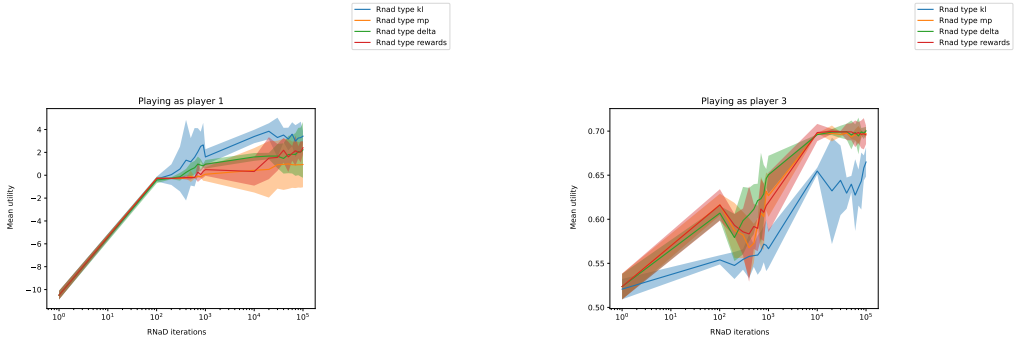


Figure 5.16. The average utilities gained by the ṘNaDs against random opponents in the three-player small board. On the left playing as player 1 and playing as player 3 on the right.

So despite the vulnerability to random opponents, the algorithms learned to defeat the random opponents in most cases. On the other hand, playing against heuristic opponents proved to be more stable. These players play rationally, so the games mostly end in a draw. The only exception was when RNaD was controlling player 3, where occasional heuristic sampling mistakes as player 1 resulted in a defeat. As in the case of random opponents, RNaD KL performed worse when playing as player 3.

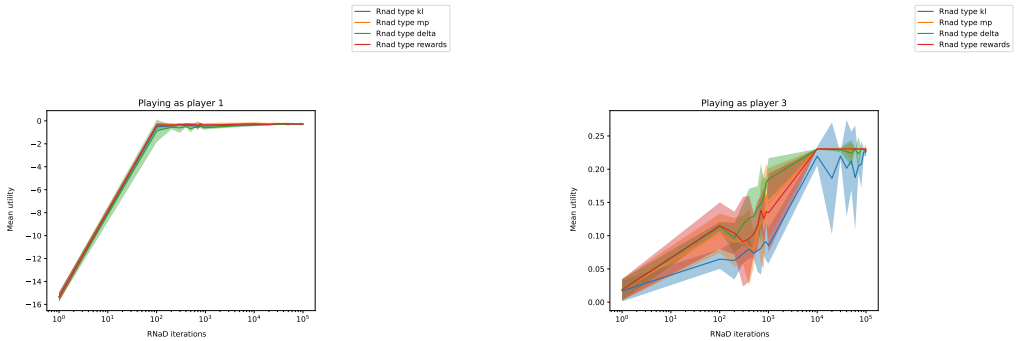


Figure 5.17. The average utilities gained by the ṘNaDs against heuristic opponents in the three-player small board. On the left playing as player 1 and playing as player 3 on the right.

To further test play against rational opponents, tests were performed against the CFR algorithm 2.2.2, which was trained for 1000 iterations on this board. The NashConv of CFR after the 1000 iterations was around 0.02, so it was about 5 times lower than the NashConv of the RNaD algorithms, which converged to around 0.1. This is caused by CFR performing updates for all infosets in the game, whereas RNaD updates are performed based on the trajectories collected during sampling, which causes the CFR results to be more precise. With increasing iterations of the RNaDs, the games ended almost always in a draw. These results were expected from self-play and indicate that all the RNaDs learned a strategy suitable against rational opponents.

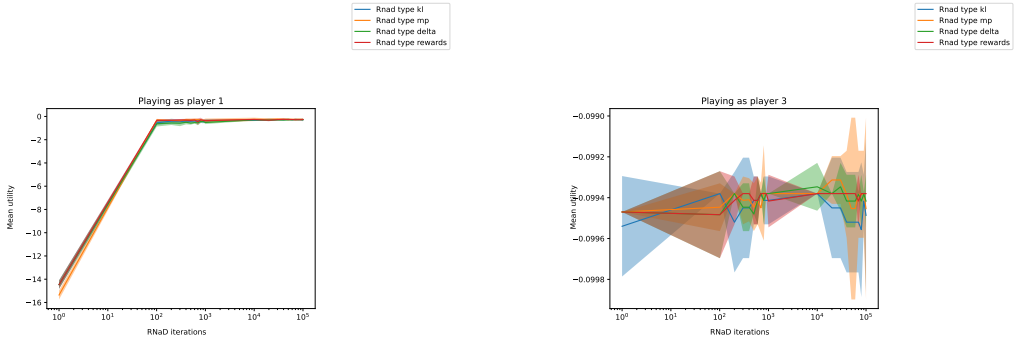


Figure 5.18. The average utilities gained by the RNaDs against CFR in the three-player small board. On the left playing as player 1 and playing as player 3 on the right.

In these small instances, the changed RNaD versions proved to be overall slightly more stable and robust than RNaD KL.

5.5.3 Medium-sized board

Other tests were performed on the three-player version of the medium-sized board 5.1.2. There, NashConv could no longer be computed exactly due to the size, so only the average utility experiments were performed. From self-play, we saw that this time RNaD KL and the altered RNaDs converged to a different fixed point.

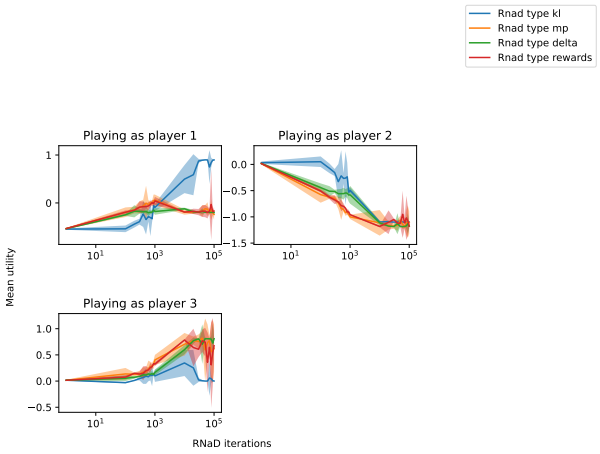
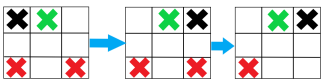


Figure 5.19. The average utilities achieved by the RNaDs in the medium-sized three-player board during self-play.

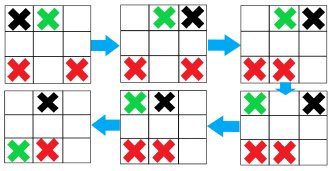
So, in RNaD KL player 1 is the winner, whereas in the other RNaDs player 3 is. In both cases, player 2 was defeated. From the found strategy profiles, we attempted to intuitively determine, which fixed point results in a better play. In the following example, RNaD δ strategy profile is used for the probabilities of the second strategy profile, because it had less variance than RNaD mp and RNaD rewards.

Example 5.2. In the three-player medium pursuit-evasion board, player 1 is acting first. We can see, that taking any action except token-right would result in being caught by player 2 on the next turn. In the KL strategy profile, player 1 takes action token-right with almost 1 probability (around 0.99). On player 2 turn, he can move two agents, either agent A located at (2, 0), or agent B located at (2,2). The only way the game could end now is if the player used the token-up action with Agent B, resulting in Agent

B being caught by player 1. This happens in the KL-strategy profile with a probability over 0.99. So the game proceeded as follows.



In the case of RNaD δ strategy profile, player 1 also had nearly 1 probability of action token-right. However, player 2 acting afterward instead had over 0.99 probability to move agent B left. This action prevents Agent B from being caught by player 1 on his next turn. Then player 3 again with over 0.99 probability moves left, to threaten Agent A of player 2. Afterwards, player 1 acts again. Intuitively, the actions that threaten agent B are either move left or token-down. The player did not learn the token action, but move left had a probability of around 0.6, with most of the remaining 0.4 probability being split almost evenly between the pass action and move down. Assume, that move left was sampled. Now, player 2 is already in a state, where no matter the taken action, he will be caught by either player 3 or 1. The most prominent actions in the policy were pass or take action token-up with Agent A, both with around 0.4 probability. Assume, that pass was sampled. Then, player 3 caught Agent A with action token-down, which had almost 1 probability. So, omitting the state where player 2 passed, the game proceeded as follows.



So at least from the intuitive point of view, the strategy profile found by the altered RNaDs results in a more rational play than the RNaD KL found one. It is also apparent, that RNaD KL will perform poorly against rational players when playing as player 2.

The vulnerability against random opponents was once again observed here. RNaD δ seems the least vulnerable to this, usually having the lowest variance of the tested versions. RNaD KL once again achieved the best results when playing as player 1. We assume this to be caused by the movement-penalty, which motivates RNaD KL towards more aggressive behavior. However, the RNaDs with the relation-based reward are more resistant to this, because a negative reward of adversarial players is added to the player as a positive reward. This causes the player to be rewarded when adversarial players get penalized for movement, making the negative movement penalty less relevant and thus making the algorithm less motivated to end the game quickly.

Against heuristic opponents, the altered versions managed to learn relatively well, achieving a good victory rate with defeats happening rarely. Usually, the games varied between the trained player managing to win, or one of the heuristic opponents eliminating the other before this could happen. Conversely, RNaD KL had a higher variance between the results, being defeated more often. The better results against rational opponents were especially apparent when playing as player 2, where the altered RNaDs had a significantly better victory rate than against random opponents.

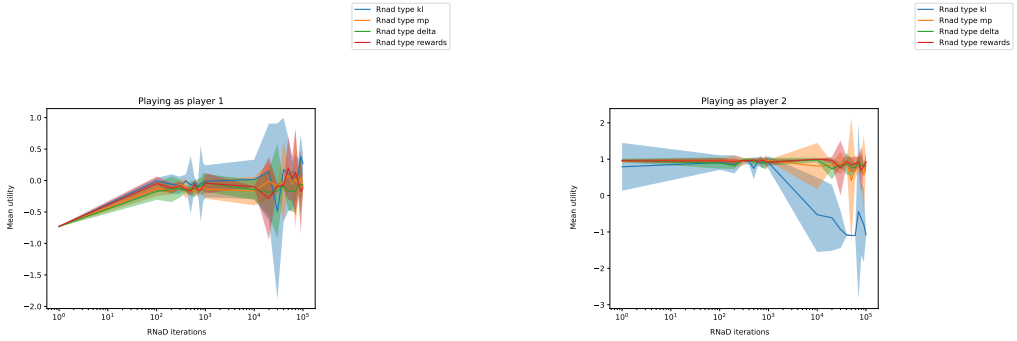


Figure 5.20. Comparison of average utilities of the RNaDs in the medium-sized three-player board against heuristic opponents. On the left is utility if playing as player 1, and on the right if playing as player 2

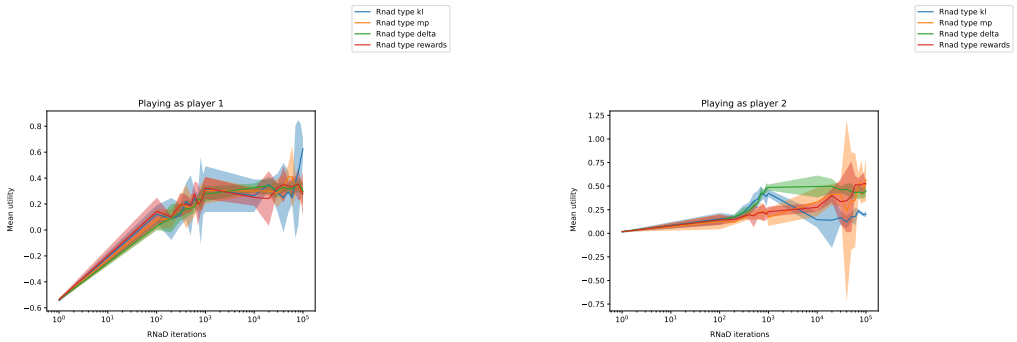


Figure 5.21. Comparison of average utilities of the RNaDs in the medium-sized three-player board against random opponents. On the left is utility if playing as player 1, and on the right if playing as player 2

RNaD δ proved to be the most stable out of the changed RNaD versions here, while also achieving better results than RNaD KL in most cases.

5.5.4 Board with many pursuit-evasion relations

Tests were also performed on the board with many pursuit-evasion relations 5.1.3 to test performance in a larger two-player zero-sum instance. This instance is heavily skewed in favor of player 2, so better results for player 2 than for player 1 were expected. In this case, all the RNaDs performed very similarly, and self-play resulted in around a 60-70% victory rate for player 2. Furthermore, as player 2 they reached almost 100% victory rate against both the heuristic and the random opponent. As player 1, the victory rate was lower, but even against a heuristic opponent, it was around 20%.

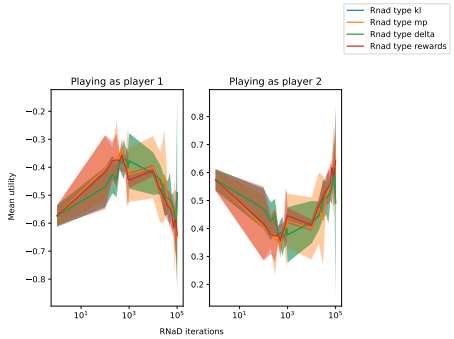


Figure 5.22. The average utilities achieved by the RNaDs in the board with many pursuit-evasion relations during self-play.

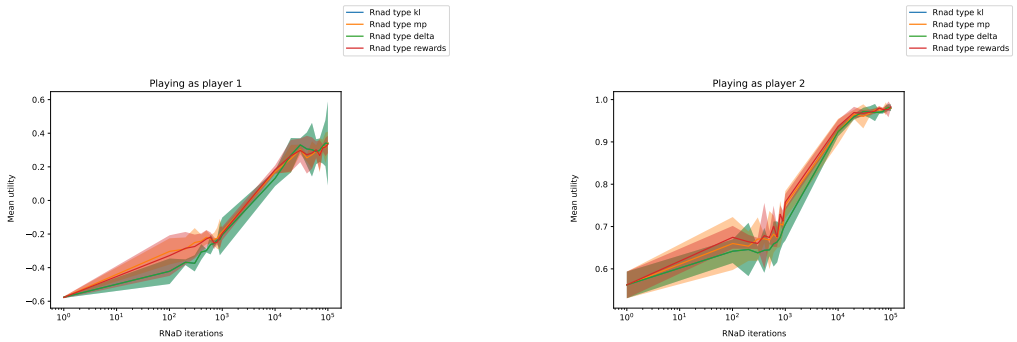


Figure 5.23. The average utilities gained by the ṘNaDs against a random opponent in the board with many pursuit-evasion relations. Utilities for player 1 are on the left, and for player 2 on the right.

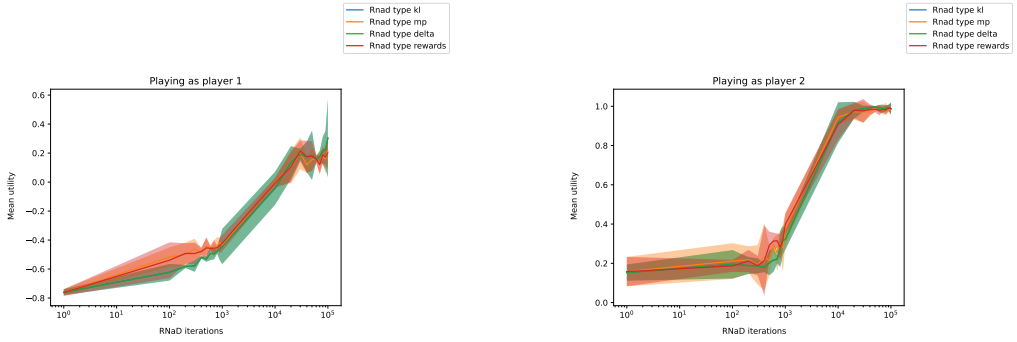


Figure 5.24. The average utilities gained by the ṘNaDs against a random opponent in the board with many pursuit-evasion relations. Utilities when playing as player 1 are on the left, and when playing as player 2 on the right.

From these results, it seems that even in a larger two-player zero-sum setting all the altered versions perform at least as well as RNaD KL.

5.5.5 Large board zero-sum

Tests of performance in a multiplayer zero-sum setting were also performed. These were performed on a zero-sum version of the large board 5.1.4 without the movement penalty.

Removing the movement penalty causes the algorithms to no longer be motivated towards ending the game quickly. Because of this, in the RNaD KL strategy profile player 1 plays much more defensively, preferring smaller moves or passing turns. Conversely, players 2 and 3 are aggressive in chasing player 5. In self-play, it was observed, that the altered RNaDs and RNaD KL once again had a slightly changed fixed point. This is caused by players 1 and 4 playing slightly more aggressively than in RNaD KL, sometimes resulting in being caught by player 5.

We believe, that this is an effect of the relation reward $R_i^\theta(h, a)$. In this board, player 5 is adversarial to players 1 and 4. This means, that these players receive a positive reward even when player 5 is caught by players 2 or 3. As a result, the players are more inclined to move, because there are more states that they consider positive for them. However, this also results in either of these two players being caught by player 5 more often. In RNaD δ, the reward is also divided by the number of players, and so is instead $\frac{R_i^\theta(h,a)}{5}$. Because of this smaller reward, it takes more iterations for the algorithm to learn this behavior. So, the algorithm starts its learning process similarly to RNaD KL, and around 20000-30000 iterations it transitions into the behavior of RNaD mp and RNaD rewards. During this transition, there was a higher variance in the achieved utilities for all players.

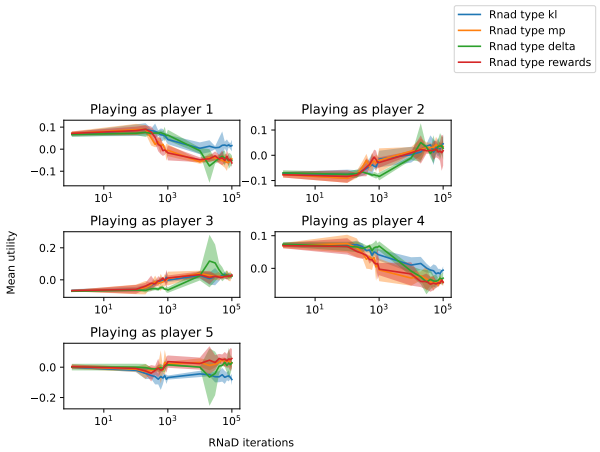


Figure 5.25. The average utilities of the RNaDs in the large zero-sum board during self-play .

Despite these differences in self-play, against the heuristic and random opponents the algorithms performed very similarly, usually reaching around 30-40% victory rate against random opponents and 10-20% victory rate against heuristic opponents, depending on which player they were playing as. The only noticeable differences were seen in RNaD δ performance. It had a slightly slower learning process, requiring more iterations to eventually reach the same results (presumably due to the reward discussed before). Also, the increased variance around 20000-30000 iterations can be seen particularly when playing as player 5. Because most of the plots were very similar for all the algorithms, only plots showing results as player 1 and player 5 are shown here, and the rest will be in appendices.

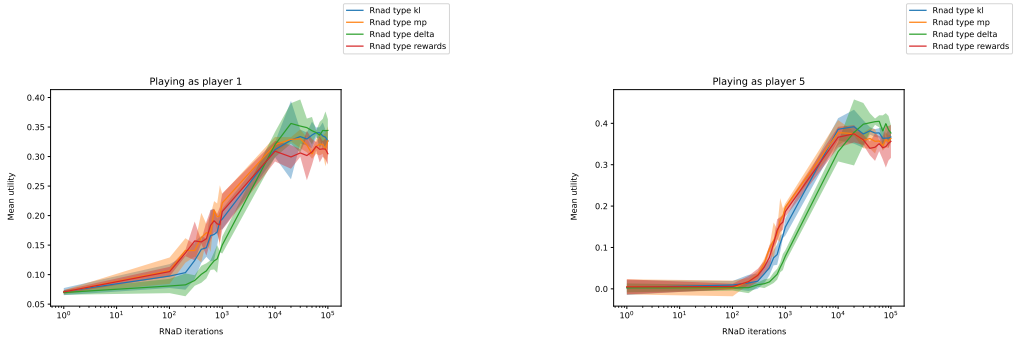


Figure 5.26. The average utilities gained by the ṘNaDs against random opponents in the large zero-sum board. On the left are utilities when playing as player 1, and on the right when playing as player 5.

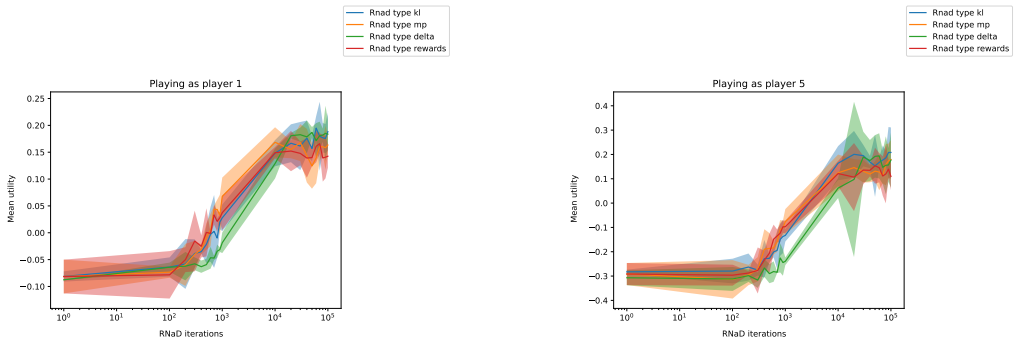


Figure 5.27. The average utilities gained by the ṘNaDs against heuristic opponents in the large zero-sum board. On the left are utilities when playing as player 1, and on the right when playing as player 5.

In this instance, all of the algorithms achieved very similar results against the tested opponents. Despite this, the altered behavior of the RNaDs using the relation-based reward resulted in increased aggressiveness of players 1 and 4, which yielded worse results for them in self-play and could lead to worse performance against rational players more complex than the used heuristic player. So the behavior altered by the relation-based reward may not always be desirable.

5.5.6 Large board general-sum

The final tests were performed on the multiplayer large board 5.1.4 with the movement penalty, so it was a general-sum instance.

For the altered RNaD versions, the found strategy profiles did not change much compared to the zero-sum version, so even the results of self-play were similar. The games mostly ended in a draw, with players 4 or 1 sometimes being caught by player 5. However, the RNaD KL strategy profile had major changes compared to the zero-sum version. Most of the time, the players were significantly more aggressive and had nearly 1 probability of playing a specific action. Moreover, the player 5 policy was very badly trained, often taking action that results in a defeat with almost 1 probability.

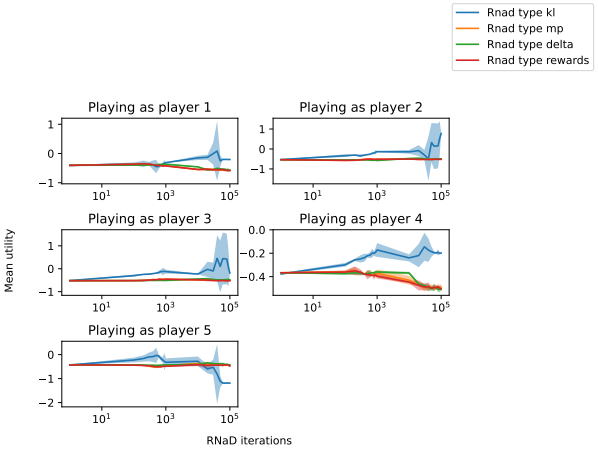


Figure 5.28. The average utilities of the RNaDs during self-play in the large general-sum instance.

This is very similar to the phenomenon, that happened in the medium-sized multiplayer board 5.5.3. So it seems, that RNaD KL generally does not work well for general-sum games. This was further observed in average utilities achieved against heuristic and random opponents. For the altered RNaD versions, the increase in gained utilities with increasing iterations seems comparable to the results achieved in the zero-sum version. On the other hand, RNaD KL achieved worse results, the training process is much less stable and there is a higher variance between the seeds. This can be illustrated when playing as player 1 and player 5, for comparison with the zero-sum version.

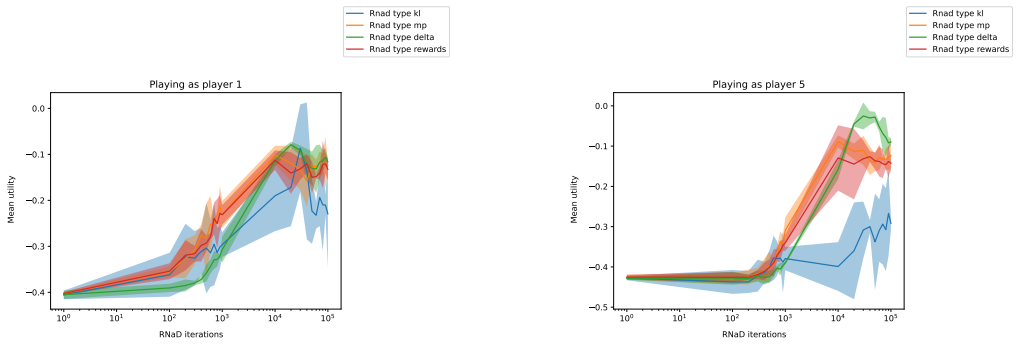


Figure 5.29. The average utilities gained by the RNaDs against random opponents in the large general-sum board. On the left are utilities when playing as player 1, and on the right when playing as player 5.

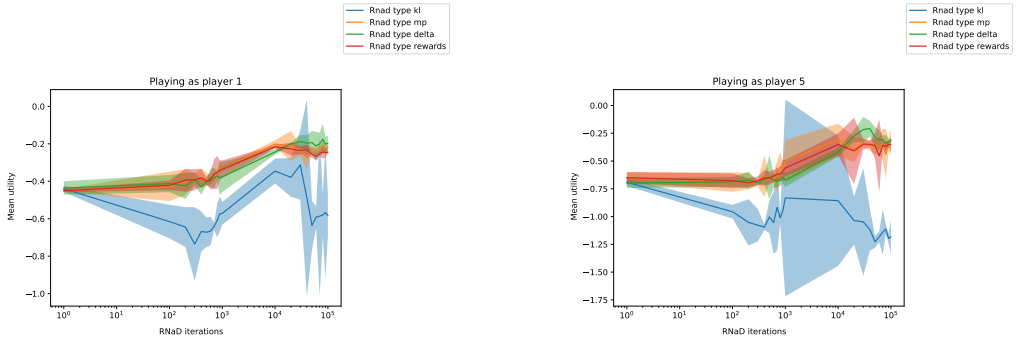


Figure 5.30. The average utilities gained by the ṘNaDs against heuristic opponents in the large general-sum board. On the left are utilities when playing as player 1, and on the right when playing as player 5.

In contrast to the zero-sum version, the altered RNaDs had significantly better results here than RNaD KL. In conclusion, while RNaD KL might be slightly more suitable for multiplayer zero-sum instances, the altered RNaDs are significantly better for general-sum settings.

Chapter 6
Conclusion

In this work, the performance of various modifications to the RNaD algorithm was tested in the multi-player general-sum setting.

The effectiveness of using entropy regularization and regularization constant annealing in place of KL divergence and regularization policy swapping was assessed. However, this change proved to be highly unstable for lower values of the regularization constant and these values proved to be different for each tested instance. Moreover, because of the lower limit of the last stable regularization constant, it proved impossible to get arbitrarily close to the Nash Equilibrium. Because of this, this change was not deemed suitable for further usage.

Other evaluated changes involved changes to reward and regularization, using domain-specific knowledge of player relations. There were three tested versions of these modifications as well as an extension of the original RNaD algorithm (RNaD KL) into a multi-player setting. Even though RNaD KL seems to be most suitable for a multi-player zero-sum setting, these modifications have shown significant improvements in a multi-player general-sum setting. Although there were not any major differences in the performance of these versions, we believe that RNaD δ is the most suitable for future usage. This version uses the alternative relation function and divides the altered rewards by the number of players. Because of this, all players´ regularization terms have an impact on the regularization, and it is equivalent to RNaD KL in a two-player zero-sum case.

However, all of the tested versions suffered from some of the same problems. The most prominent one is being too fixated on rationally playing opponents. So, opponents that play slightly irrationally may lead a player in a state, where he has not trained enough. In this work, we have used an on-policy version of RNaD, so it did not train in some parts of the game. Using RNaD as an off-policy algorithm may help alleviate this issue.

Another problem is that of the algorithm stability. In certain scenarios, there was a high variance between achieved values based on the random seed used to initialize the neural network. Some instability was present in all of the algorithms, however, the proposed RNaD δ generally did not suffer much from this problem.

Finally, in some cases, the relation changes could alter the algorithm behavior in potentially undesirable ways, as was demonstrated in the multi-player zero-sum instance. This is likely a limitation of the extensions, which indicates that they may not be suitable for certain instances.

Because of the improvements compared to RNaD KL, we believe that utilizing player relations to change the regularization method and alter given rewards has potential. Further development of these methods should focus on improving the robustness towards irrational play, as well as improving the algorithm stability. Another possible improvement would be to alter these extensions so that they do not cause undesirable behavior in multiplayer zero-sum instances.

References

[1] Esther Derman, and Shie Mannor. *Distributional Robustness and Regularization in Reinforcement Learning*. 2020.

[2] Jesse Farebrother, Marlos C. Machado, and Michael Bowling. *Generalization and Regularization in DQN*. 2020.

[3] Julien Perolat, Remi Munos, Jean-Baptiste Lespiau, Shayegan Omidshafiei, Mark Rowland, Pedro Ortega, Neil Burch, Thomas Anthony, David Balduzzi, Bart De Vylder, Georgios Piliouras, Marc Lanctot, and Karl Tuyls. From Poincaré Recurrence to Convergence in Imperfect Information Games: Finding Equilibrium via Regularization. 2020.

[4] Samuel Sokota, Ryan D'Orazio, Chun Kai Ling, David J. Wu, J. Zico Kolter, and Noam Brown. Abstracting Imperfect Information Away from Two-Player Zero-Sum Games. 2023.

[5] Julien Perolat, Bart De Vylder, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer, Paul Muller, Jerome T. Connor, Neil Burch, Thomas Anthony, Stephen McAleer, Romuald Elie, Sarah H. Cen, Zhe Wang, Audrunas Gruslys, Aleksandra Malysheva, Mina Khan, Sherjil Ozair, Finbarr Timbers, Toby Pohlen, Tom Eccles, Mark Rowland, Marc Lanctot, Jean-Baptiste Lespiau, Bilal Piot, Shayegan Omidshafiei, Edward Lockhart, Laurent Sifre, Nathalie Beauguerlange, Remi Munos, David Silver, Satinder Singh, Demis Hassabis, and Karl Tuyls. Mastering the game of Stratego with model-free multiagent reinforcement learning. *Science*. 2022, 378 (6623), 990-996. DOI 10.1126/science.add4679.

[6] Martin Schmid, Matej Moravčík, Neil Burch, Rudolf Kadlec, Josh Davidson, Kevin Waugh, Nolan Bard, Finbarr Timbers, Marc Lanctot, G. Zacharias Holland, Elnaz Davoodi, Alden Christianson, and Michael Bowling. Student of Games: A unified learning algorithm for both perfect and imperfect information games. *Science Advances*. 2023, 9 (46). DOI 10.1126/sciadv.adg3256.

[7] Pim Nijssen, and Mark H. M. Winands. Monte Carlo Tree Search for the Hide-and-Seek Game Scotland Yard. *IEEE Transactions on Computational Intelligence and AI in Games*. 2012, 4 (4), 282-294. DOI 10.1109/TCIAIG.2012.2210424.

[8] Sourabh Bhattacharya, Tamer Başar, and Maurizio Falcone. *Surveillance for Security as a Pursuit-Evasion Game*. In: Radha Poovendran, and Walid Saad, eds. *Decision and Game Theory for Security*. Cham: Springer International Publishing, 2014. 370–379. ISBN 978-3-319-12601-2.

[9] Alexander Alexopoulos, Benjamin Kirsch, and Essameddin Badreddin. *Realization of pursuit-evasion games with unmanned aerial vehicles*. In: *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2017. 797-805.

[10] J. Mikael Eklund, Jonathan Sprinkle, and S. Shankar Sastry. Switched and Symmetric Pursuit/Evasion Games Using Online Model Predictive Control With Application to Autonomous Aircraft. *IEEE Transactions on Control Systems Technology*. 2012, 20 (3), 604-620. DOI 10.1109/TCST.2011.2136435.

[11] Erik P. Blasch, Khanh Pham, and Dan Shen. *Orbital satellite pursuit-evasion game-theoretical control*. In: *2012 11th International Conference on Information Science, Signal Processing and their Applications (ISSPA)*. 2012. 1007-1012.

[12] Liran Zhao, Yulin Zhang, and Zhaohui Dang. PRD-MADDPG: An efficient learning-based algorithm for orbital pursuit-evasion game with impulsive maneuvers. *Advances in Space Research*. 2023, 72 (2), 211-230. DOI https://doi.org/10.1016/j.asr.2023.03.014.

[13] Naiming QI, Qilong SUN, and Jun ZHAO. Evasion and pursuit guidance law against defended target. *Chinese Journal of Aeronautics*. 2017, 30 (6), 1958-1973. DOI https://doi.org/10.1016/j.cja.2017.06.015.

[14] Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Paul Muller, Timo Ewalds, Ryan Faulkner, János Kramár, Bart De Vylder, Brennan Saeta, James Bradbury, David Ding, Sebastian Borgeaud, Matthew Lai, Julian Schrittwieser, Thomas Anthony, Edward Hughes, Ivo Danihelka, and Jonah Ryan-Davis. OpenSpiel: A Framework for Reinforcement Learning in Games. *CoRR*. 2019, abs/1908.09453

[15] Vojtěch Kovařík, Martin Schmid, Neil Burch, Michael Bowling, and Viliam Lisý. *Rethinking Formal Models of Partially Observable Multiagent Decision Making*. 2021.

[16] John Von Neumann, and Oskar Morgenstern. *Theory of Games and economic behavior*. 2007.

[17] John F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*. 1950, 36 (1), 48-49. DOI 10.1073/pnas.36.1.48.

[18] Y. Shoham, and K. Leyton-Brown. *Multiagent Systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2009 .

[19] Revan MacQueen. *A Proof that Coarse Correlated Equilibrium Implies Nash Equilibrium in Two-Player Zero-Sum Games*. 2023.

[20] Finbarr "Timbers, Nolan Bard, Edward Lockhart, Marc Lanctot, Martin Schmid, Neil Burch, Julian Schrittwieser, Thomas Hubert, and Michael" Bowling. "Approximate exploitability: Learning a best response in large games". 2020.

[21] Paul Muller, Shayegan Omidshafiei, Mark Rowland, Karl Tuyls, Julien Perolat, Siqi Liu, Daniel Hennes, Luke Marris, Marc Lanctot, Edward Hughes, Zhe Wang, Guy Lever, Nicolas Heess, Thore Graepel, and Remi Munos. *A Generalized Training Approach for Multiagent Learning*. 2020.

[22] Stephen McAleer, Gabriele Farina, Marc Lanctot, and Tuomas Sandholm. *ESCHER: Eschewing Importance Sampling in Games by Computing a History Value Function to Estimate Regret*. 2022.

[23] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. *Regret Minimization in Games with Incomplete Information*. In: J. Platt, D. Koller, Y. Singer, and S. Roweis, eds. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2007.
`https://proceedings.neurips.cc/paper_files/paper/2007/file/08d98638c6fcd194a4b1e6992063e944-Paper.pdf`.

[24] Sergiu Hart, and Andreu Mas-Colell. A Simple Adaptive Procedure Leading to Correlated Equilibrium. *Econometrica*. 2000, 68 (5), 1127-1150. DOI https://doi.org/10.1111/1468-0262.00153.

[25] Brendan McMahan. *Follow-the-Regularized-Leader and Mirror Descent: Equivalence Theorems and L1 Regularization*. In: Geoffrey Gordon, David Dunson, and Miroslav Dudík, eds. *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Fort Lauderdale, FL, USA: PMLR, 2011. 525–533.
`https://proceedings.mlr.press/v15/mcmahan11b.html`.

[26] Daniel Hennes, Dustin Morrill, Shayegan Omidshafiei, Remi Munos, Julien Perolat, Marc Lanctot, Audrunas Gruslys, Jean-Baptiste Lespiau, Paavo Parmas, Edgar Duenez-Guzman, and Karl Tuyls. *Neural Replicator Dynamics*. 2020.

[27] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. *IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures*. 2018.

[28] Gergely Neu, Anders Jonsson, and Vicenç Gómez. *A unified view of entropy-regularized Markov decision processes*. 2017.

Appendix A
Additional experimental results

A.1 Small boards

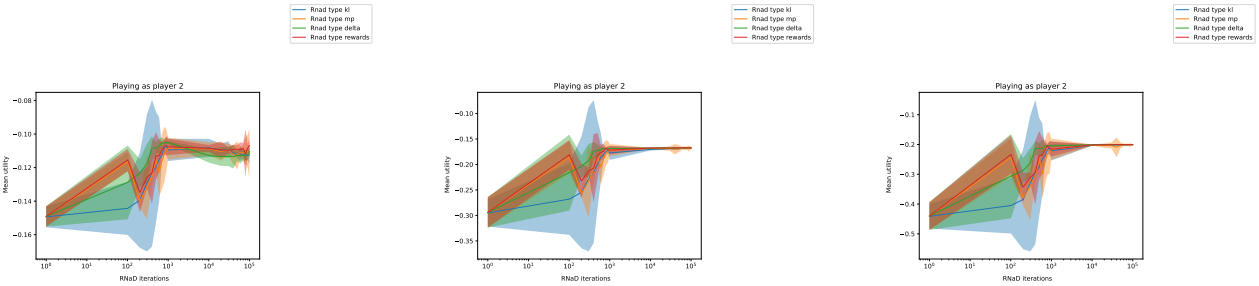


Figure A.1. The average utilities gained by the RNaDs when playing as player 2 in the three-player small board. From left to right are the results against random opponents, heuristic opponents, and CFR respectively.

A.2 Medium-sized board

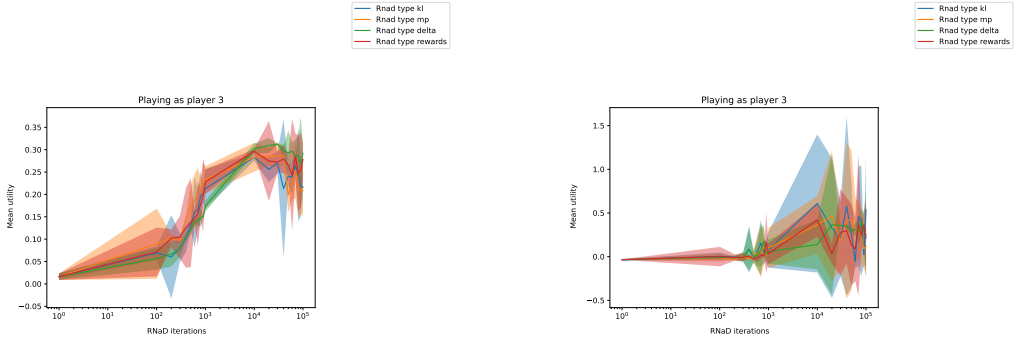


Figure A.2. The average utilities gained by the RNaDs when playing as player 3 in the three-player medium-sized board. Results against random opponents are on the left and results against heuristic opponents are on the right.

A.3 Large board zero-sum

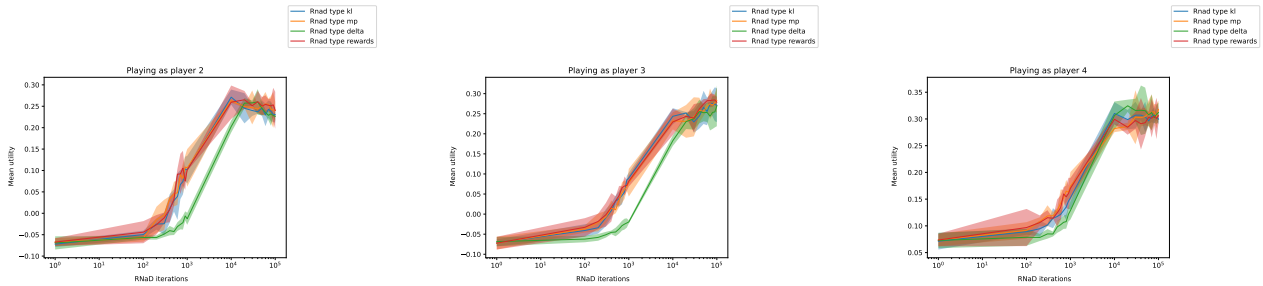


Figure A.3. The average utilities gained by the RNaDs against random opponents in the large zero-sum board. From left to right are the results when playing as player 2, player 3, and player 4 respectively.

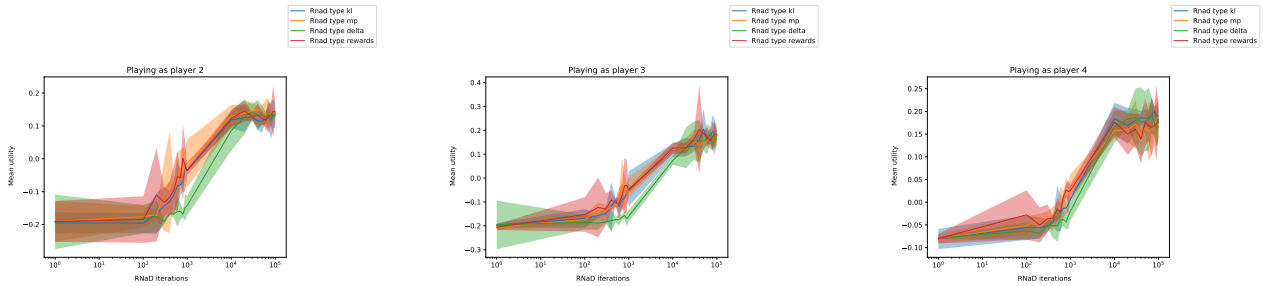


Figure A.4. The average utilities gained by the RNaDs against heuristic opponents in the large zero-sum board. From left to right are the results when playing as player 2, player 3, and player 4 respectively.

A.4 Large board general-sum

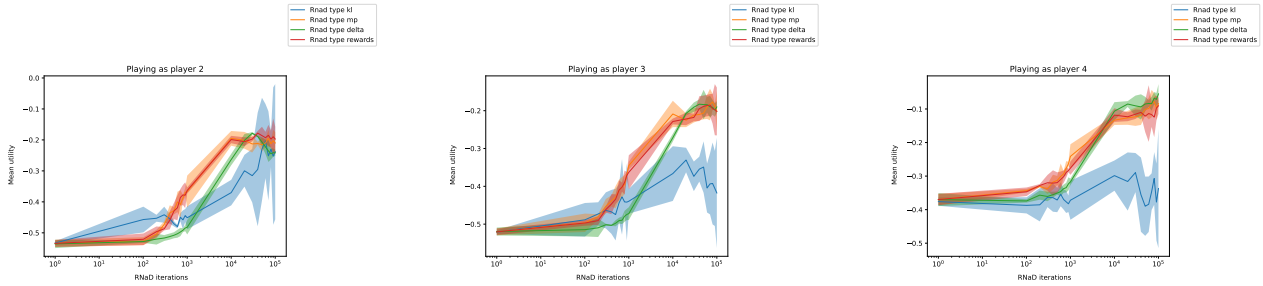


Figure A.5. The average utilities gained by the RNaDs against random opponents in the large general-sum board. From left to right are the results when playing as player 2, player 3, and player 4 respectively.

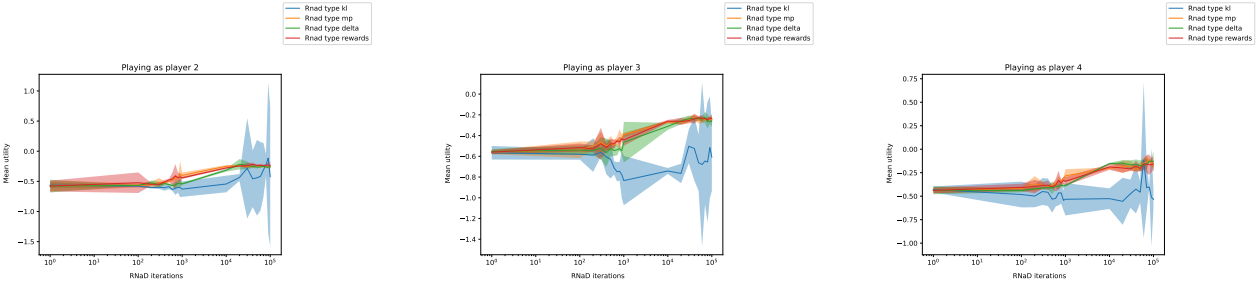


Figure A.6. The average utilities gained by the RNaDs against heuristic opponents in the large general-sum board. From left to right are the results when playing as player 2, player 3, and player 4 respectively.