CZECH TECHNICAL UNIVERSITY IN PRAGUE

F3

Faculty of Electrical Engineering
Department of Compute Science

Master's Thesis

Variance Reduction in One-Sided Partially Observable Stochastic Games

Ondřej Kubíček

20.05.2022
Supervisor: doc. Mgr. Branislav Bošanský, PhD.

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Kubíček Ondřej**	Personal ID number:	**474745**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Science**		
Study program:	**Open Informatics**		
Specialisation:	**Artificial Intelligence**		

II. Master's thesis details

Master's thesis title in English:

Variance Reduction in One-Sided Partially Observable Stochastic Games

Master's thesis title in Czech:

Redukce variance v jednostranně pozorovatelných stochastických hrách

Guidelines:

One-Sided Partially Observable Stochastic Games are dynamic games with infinite horizon where only one player has imperfect information and the opponent has full information. The original algorithm for solving this class of games has limited scalability. One possible way to improve the scalability is to replace the exact submethods for solving stage games with approximate iterative algorithms. However, usage of approximate methods (e.g., in combination with approximate representation of value functions) can lead to increased variance and slow convergence. The expected approach is as follows.
1. Get familiar with the algorithm PG-HSVI, analyze the variance in values in the current version of the algorithm.
2. Survey the existing variance-reduction techniques used for sampling-based game-theoretic algorithms.
3. Analyze compatibility of methods from the previous step, adapt, and implement selected methods into the HSVI for PG-HSVI.
4. Compare these methods and analyze the impact of these techniques on the speed of the convergence in POSGs.

Bibliography / sources:

[1] Horák, K., Bošanský, B., & Pěchouček, M. (2017). Heuristic Search Value Iteration for One-Sided Partially Observable Stochastic Games. In AAAI (pp. 558-564)
[2] Horák, K., Bošanský, B., Kovařík, V., & Kiekintveld, C. (2020). Solving Zero-Sum One-Sided Partially Observable Stochastic Games. arXiv preprint arXiv:2010.11243.
[3] Schmid, M., Burch, N., Lanctot, M., Moravcik, M., Kadlec, R., & Bowling, M. (2019, July). Variance reduction in monte carlo counterfactual regret minimization (VR-MCCFR) for extensive form games using baselines. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 33, No. 01, pp. 2157-2164).

Name and workplace of master's thesis supervisor:

doc. Mgr. Branislav Bošanský, Ph.D.   Artificial Intelligence Center FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **28.01.2022**   Deadline for master's thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

doc. Mgr. Branislav Bošanský, Ph.D.		prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgement

I would like to thank my supervisor, doc. Mgr. Branislav Bošanský, Ph.D., for his guidance throughout my studies and especially throughout the work on this thesis. I would also like to thank Bc. Jakub Brož and MSc. Dominik Andreas Seitz for their part in studying OS-POSGs and to Ing. Karel Horák, Ph.D., whose work started the research in these games. Furthermore, I would like to thank my family for their support during my studies. Lastly, I would like to thank Zichovec brewery for helping me with my studies.

Declaration

I declare that this thesis has been composed solely by myself and that it has not been submitted, in whole or in part, in any previous application for a degree. Except where stated otherwise by reference or acknowledgement, the work presented is entirely my own.

In Prague 20.5.2022

....

Abstrakt / Abstract

Hledání téměř optimálních strategií ve hrách s neúplnou informací není obecně jednoduché. Posilované učení lze použít pro hledání přibližných řešení v takto složitých hrách. Tyto metody jsou zatíženy nezanedbatelným rozptylem, který je způsobený vzorkováním herního stromu. Několik metod pro snižovaní rozptylu bylo vytvořeno, aby došlo k jeho snížení a tím ke zrychlení algoritmů posilovaného učení.

V této práci se snažíme ukázat jak jedna hodnotová funkce ovlivňuje rozptyl ve stochastických hrách. Dále zkoumáme jak dvě hodnotové funkce, použité jako dolní a horní odhad, ovlivňují rozptyl ve stochastických hrách pro dva hráče a jednostranných částečně pozorovatelných stochastických hrách.

Dále představujeme nový způsob pro řešení stochastických her pro dva hráče se simultánními pohyby a pro řešení jednostranných částečně pozorovatelných stochastických her pomocí minimalizace lítosti. Také představujeme vylepšenou verzi ekvilibria kvantové odpovědi pro řešení těchto her. Tyto přibližné metody na aproximace Nashova ekvilibra tvoří dodatečné chyby při výpočtech. Empiricky odhadujeme rozptyl, který tyto metody tvoří a zkoušíme jak jednotlivé techniky redukce rozptylu ovlivňují celkovou konvergenci algoritmů.

Klíčová slova: částečně pozorovatelné stochastické hry, častečně pozorovatelné Markovské rozhodovací procesy, redukce rozptylu, ekvilibrium kvantové odpovědi, Nashovo ekvilibrium, hodnotová funkce

Překlad titulu: Redukce variance v jednostranně pozorovatelných stochastických hrách

Finding near-optimal strategies in imperfect information games is generally intractable. Reinforcement learning is used to find approximate solutions in such complicated games. These methods often deal with significant variance caused by a sampling of the game tree. Multiple variance reduction techniques were developed to reduce the variance and speed up the convergence of reinforcement learning algorithms.

In this work, we show the effect of a single value function on variance in stochastic games. Furthermore, we investigate how two value functions, used as a lower and upper bound, affect variance in two-player stochastic games with simultaneous moves and one-sided partially observable stochastic games.

We introduce a new way to solve two-player stochastic games with simultaneous moves and one-sided partially observable stochastic games by regret minimization. We also refine the quantal response equilibrium method for solving these games. These different methods of approximating Nash equilibria introduce additional errors into the computation. We also empirically estimate the variance introduced by these errors and test if variance reduction techniques improve the overall convergence of the algorithms.

Keywords: partially observable stochastic games, one-sided partially observable stochastic games, partially observable Markov decision processes, variance reduction, quantal response equilibrium, Nash equilibrium, value function

Contents /

Tables / Figures

Chapter 1
Introduction

Sequential decision-making is an important part of artificial intelligence. Things like autonomous cars [1][2], robotic movement [3], and network security [4] all use sequential decision making for a long time planning. However, it is sometimes necessary to plan while considering other agents in the environment. These agents may have malicious intents, so it is necessary to act in such a manner that cannot be exploited by the opponent. Game theory provides a mathematical framework for studying these multi-agent environments, in which each agent maximizes its profit.

Often the agent does not have the full information about the environment, which could be dynamic, or affected by the other agents. Solving such problems in a multi-agent setting is notoriously difficult. One-sided partially observable stochastic games (OS-POSGs) [5][6] are a subset of such games, which are played by just two players, and one player gain is other players lost. Additionally, one of the agents has complete information about the environment, while the other has only partial information. This makes the solution of such a game the worst-case scenario for the agent with partial knowledge.

Modification of heuristic search value iteration initially developed for partially observable Markov decision processes may be used to solve OS-POSGs [5]. This algorithm uses lower and upper bound, and at each stage, it solves linear programs, which updates both of these bounds until the gap between bounds gets sufficiently small. Linear programming solves the game precisely, but it may require a lot of time and space. Therefore other approaches could be used to avoid one or both of these problems. Quantal response equilibrium was already tried on these games with a moderate success [7]. In this work, we also try different approaches based on regret minimization. These approximate methods do not find Nash equilibrium exactly, but only approximately, and therefore their convergence requires necessarily more iterations to reach the same precision.

Reinforcement learning suffers from similar issues as the approximate methods for solving OS-POSGs. It requires many iterations to converge somewhat closely to the optimal strategy. One technique that allows the reinforcement learning algorithm to converge more quickly is variance reduction. The variance which is being reduced by this technique is mainly caused by the sampling of the game tree instead of solving each trajectory in the tree.

Measuring variance in OS-POSGs becomes difficult because of the continuous belief space. Additionally, there is only one algorithm heuristic search value iteration for solving these games. Stochastic games with simultaneous moves provide full information to both players. Therefore it has discrete state space, making it easier to estimate the variance. There are multiple algorithms to solve stochastic games with simultaneous moves. This makes it possible to compare them based on convergence and variance.

This work focuses on how variance reduction may be used in both stochastic games with simultaneous moves and OS-POSGs. It also studies whether this technique would be helpful because, unlike the reinforcement learning techniques, the heuristic search value iteration already stores information about the whole belief space in the form of upper and lower bound. It also evaluates the variance for individual methods of solving the game.

1.1 Outline

The second chapter describes the basics of single-agent sequential decision making. The third chapter introduces concepts from game theory, which are essential throughout the whole thesis. The fourth chapter introduces the most used variance reduction technique of control variates. This chapter also presents a couple of ways control variates are used in reinforcement learning. Chapter 5 offers a way to use quantal response equilibrium and regret minimization to solve stochastic games. It also shows a new algorithm that is an intermediate step between value iteration and heuristic search value iteration and how it can be used with action baselines as a variance reduction technique. Throughout Chapter 6, we introduce the same notion for one-sided partially observable stochastic games. In the seventh chapter, we show the empirical evaluation of all the concepts presented in the previous sections.

Chapter 2
Partially Observable Markov Decision Processes

Decision making of a single agent under uncertainty is a well studied problem in artificial intelligence [8][9][10]. Markov decision processes (MDPs) [11] have been found quite useful in this field. However, MDPs assume that the environment has a fully observable state. This may not be the case even for single-agent applications [12]. Partially observable Markov decision processes (POMDPs) were developed to handle even environments without fully observable states [13]. We will formally define MDPs in this chapter and present the value iteration algorithm, which is capable of solving them to arbitrary precision. Then we will define POMDPs with the heuristic search value iteration algorithm, which is used to solve them.

2.1 Markov Decision Processes

Definition 2.1. [11]
Markov decision process (MDP) is a tuple $(S, A, T, R, s^{(0)}, S^{(\text{end})}, \gamma)$, where
S is a finite set of state.
A is a finite set of actions.
$T: S \times A \times S \to [0,1]$ is a transition function, where $T(s'|s,a)$ is a conditional probability that the game transitions from state $s \in S$ to state $s' \in S$ after agent chooses action $a \in A$.
$R: S \times A \to \mathbb{R}$ is a reward function, where $R(s,a)$ is a reward given to agent, if it chooses action $a \in A$ in state $s \in S$.
$s^{(0)} \in S$ is a initial state of the process.
$S^{(end)} \subseteq S$ are terminal states of the process.
$\gamma \in [0,1]$ is a discount factor, used to prefer earlier rewards.

Each Markov decision process starts in a state $s^{(0)}$ and at each step t in state $s^{(t)} \in S$, the agent chooses one action $a^{(t)} \in A$, which transfers it to the new state $s^{(t+1)} \in S$ based on $T(s^{(t+1)}|s^{(t)}, a^{(t)})$ and receives a reward $R(s^{(t)}, a^{(t)})$. Reward may also be defined as $R(s^{(t)}, a^{(t)}, s^{(t+1)})$ or $R(s^{(t)})$. We have decided to use $R(s^{(t)}, a^{(t)})$, due to similarities with OS-POSGs as defined in [5]. Transition function T depends only on the current state and the taken action. This is called Markov property [14], and it is a fundamental assumption for any model to be considered a Markov model. Because of the Markov property, we know that no matter when the agent reaches some state, it always expects the same payoff, therefore its optimal strategy is only dependent on the state. This state-dependent strategy is called policy

Definition 2.2.
Let $G = (S, A, T, R, s^{(0)}, S^{(\text{end})}, \gamma)$ be a MDP.
Policy $\pi\colon S \to [0,1]^{|A|}$ is a probability distribution over actions in state $s \in S$. $\pi(s)$ denotes a probability distribution vector over all actions in state $s \in S$ and $\pi(s,a)$ denotes a probability of choosing single action $a \in A$ in same state.

$$\sum_{a \in A} \pi(s,a) = 1 \quad \forall s \in S$$

Playing the same action in the same state may result in a different next state in MDP. Therefore employing the same policy may lead to vastly different results. To compare policies, we have to use the expected value

Definition 2.3.
Let $G = (S, A, T, R, s^{(0)}, S^{(\text{end})}, \gamma)$ be a MDP.
Expected value of policy π in state $s \in S$ is

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=1}^{\infty} \gamma^t R(s^{(t)}, a^{(t)})\right]$$

Optimal policy is such a policy π^* for which holds

$$\pi^*(s) = \operatorname*{argmax}_{\pi} V^{\pi}(s)$$

Optimal value is a expected value of optimal policy.

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

The expected value may be rewritten recursively, where current values from all possible future states are used to compute the value of the current state.

$$V^{\pi}(s) = \max_{a \in A} R(s,a) + \sum_{s' \in S} \gamma T(s'|s,a) V^{\pi}(s') \tag{2.1}$$

Equation (2.1) is called Bellman equation [11]. Due to the usage of max operator, it is not possible to simply solve the system of linear equations. However, it is possible to improve values to arbitrary precision iteratively, this approach is called value iteration Since the values are iteratively improved in each state, they may be initialized arbitrarily, but closer to the optimal value, fewer iterations are required to reach the target precision. The easiest initialization is to set each value to zero. Value iteration is described in Algorithm 1. Pure policy in each state may then be extracted by using the following equation

$$a^* = \operatorname*{argmax}_{a \in A} R(s,a) \sum_{s' \in S} + \gamma T(s'|s,a) V(s) \tag{2.2}$$

Algorithm 1 Value Iteration

```
function VALUE ITERATION($S, A, T, R, \gamma, \varepsilon$)
    $V(s) \leftarrow 0 \quad \forall s \in S$
    $\Delta \leftarrow \varepsilon$
    while $\Delta \geq \varepsilon$ do
        $\Delta \leftarrow 0$
        for $s \in S$ do
            $v \leftarrow \max_{a \in A} R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a)V(s')$
            $\Delta \leftarrow \max(\Delta, v - V(s))$
            $V(s) \leftarrow v$
        end for
    end while
end function
```

Value iteration could also be written as following LP, where we would recompute values for each state until we converge to the desired precision. This linear program allows mixed policies, which are not possible in the usual value iteration algorithm.

$$\max_{V(s),\pi} V(s) \tag{2.3a}$$

$$\text{s.t. } V(s) \leq \sum_{a \in A} \pi(s,a) \left[R(s,a) + \sum_{s' \in S} \gamma T(s'|s,a)V(s') \right] \tag{2.3b}$$

$$\sum_{a \in A} \pi(a) = 1 \tag{2.3c}$$

$$\pi(a) \geq 0 \qquad \forall a \in A \tag{2.3d}$$

2.2 Partially Observable Markov decision Processes

Giving the agent only partial information makes it significantly harder to find the optimal policy. The agent has to argue about multiple states that it might be in and act accordingly. Based on taken actions and received observations, the agent stores its belief about the current state, which is a probability distribution over all states.

Definition 2.4.
Let S be set of MDP or POMDP states.
Belief $b \rightarrow [0,1]^{|S|}$ is a probability distribution over all states $s \in S$

$$\sum_{s \in S} b(s) = 1$$

Each POMDP starts by sampling the initial state $s^{(0)} \sim b^{(0)}$ and at each step t in state $s^{(t)} \in S$ the agent chooses one action $a^{(t)} \in A$, which transfers it to the state $s^{(t+1)} \in S$ and gives it an observation $o^{(t)} \in O$ based on $T(s^{(t+1)}, o^{(t)}|s^{(t)}, a^{(t)})$. The Player also receives a reward $R(s^{(t)}, a^{(t)})$. Process terminates when agent reaches some terminal state $s^{(\text{end})} \in S^{(\text{end})}$. If $S^{(\text{end})} = \emptyset$, then the process never ends. If each observation is unique to one state, the POMDP simplifies into MDP. Therefore the belief is every time non-zero in exactly one state.

Definition 2.5.
Partially observable Markov decision process (POMDP) is a tuple $(S, A, O, T, R, b^{(0)}, S^{(\text{end})}, \gamma)$, where
S is a finite set of state.
A is a finite set of actions.
O is a finite set of observation.
$T\colon S \times A \times O \times S \to [0,1]$ is a transition function, where $T(s', o|s, a)$ is a conditional probability that the game transitions from state $s \in S$ to the state $s' \in S$ and the agent receives observation $o \in O$ after it chooses the action $a \in A$.
$R\colon S \times A \times \mathbb{R}$ is a reward function, where $R(s, a)$ is a reward given to agent, if it chooses an action $a \in A$ in a state $s \in S$.
$b^{(0)}\colon S \to \mathbb{R}$ is a initial belief agent has about the environment, where $b^{(0)}(\text{end})(s)$ is a probability that the POMDP is in state $s \in S$ at the beginning.
$S^{(\text{end})} \subseteq S$ are terminal states of the process.
$\gamma \in [0,1]$ is a discount factor, used to prefer earlier rewards.

Even when transitions between states are the same as in MDP, the agent also has to transition in its belief. After playing an action $a \in A$ and receiving an observation $o \in O$, the agent changes its belief in the following way

$$b'(s') = \tau(b, a, o)(s') = \frac{\sum_{s \in S} b(s) T(s', o|s, a)}{\sum_{s \in S} \sum_{s'' \in S} b(s) T(s'', o|s, a)} \tag{2.4}$$

Expected value, optimal policy, and optimal value are defined similarly to the MDP. However, belief is used instead of the state.

Definition 2.6.
Let $G = (S, A, O, T, R, b^{(0)}, S^{(\text{end})}, \gamma)$ be POMDP.
Expected value of policy π in a belief b is

$$V^{\pi}(b) = \mathbb{E}_{\pi,b}\left[\sum_{t=1}^{\infty} \gamma^t R(s^t, a^t)\right]$$

Optimal policy is such a policy π^* for which holds

$$\pi^*(b) = \operatorname*{argmax}_{\pi} V^{\pi}(b)$$

Optimal value is a expected value of optimal policy.

$$V^*(b) = \max_{\pi} V^{\pi}(b)$$

Rewriting the expected value to be computed recursively results in a Bellman equation for POMDPs.

$$V^{\pi}(b) = \max_{a \in A} \sum_{s \in S} b(s) R(s, a) + \gamma \sum_{o \in O} V^{\pi}(\tau(b, a, o)) \sum_{s \in S} \sum_{s' \in S} b(s) T(s', o|s, a) \tag{2.5}$$

States in MDP are defined as a discrete set, while belief in POMDP is a continuous distribution over all states. Therefore the expected value for POMDPs is also a continuous function.

The optimal Value function for POMDPs was proven to be convex [15].

2.3 Heuristic Search Value Iteration

The major disadvantage of the usual value iteration in MDPs is that many iterations are required to fully converge to optimal values, and we do not know how close to the optimal value we are. Heuristic search value iteration (HSVI) deals with this issue by maintaining upper and lower bound, which are getting more tight with each iteration. Full implementation of HSVI for POMDPs is shown in the Algorithm 3.

2.3.1 HSVI Value Functions

Since the optimal value function is convex, we could approximate it by using several linear functions. These functions are called α-vectors [15], and we could do value iteration in POMDPs by using these functions.

Definition 2.7. [15]
α-Vector is a linear function $\alpha\colon S \to \mathbb{R}$, where $\alpha(s)$ is a expected value in state $s \in S$.
Expected Value of α-vector α in belief b is then

$$\alpha(b) = \sum_{s\in S} b(s)\alpha(s)$$

Γ is a set of α-vectors.

$$\Gamma = \{\alpha_0, ..., \alpha_n\}$$

The value function with α-vectors is then defined as

$$V^{\Gamma}_{\text{LB}}(b) = \max_{\alpha\in\Gamma} \alpha(b) \tag{2.6}$$

With this usage of the α-vectors, the value function is restricted to be always less or equal to the optimal value function. Otherwise, adding the α-vector to the set Γ would not improve the value function without removing some α-vector from Γ. This suggests that the values may not be initialized arbitrarily because this property has to hold

$$V^{\Gamma}_{\text{LB}}(b) \leq V^*(b) \tag{2.7}$$

Γ is a lower bound of optimal solution.

When creating a new α-vector to update the value function, we will use Bellman equation (2.5), but with slight modification to use α-vectors. We will start with selecting the α-vector for each action and observation, which has the highest expected reward after the belief transition.

$$\alpha_{a,o} = \operatorname*{argmax}_{\alpha\in\Gamma} \sum_{s\in S} \alpha(s)b(s) \tag{2.8}$$

Bellman equation is then performed for each action

$$\alpha_a(s) = R(s,a) + \gamma \sum_{s'\in S}\sum_{o\in O} \alpha_{a,o}(s')T(s',o|s,a) \tag{2.9}$$

The new α-vector, which should be added to the value function, is selected by maximizing the expected reward.

$$\alpha^* = \operatorname*{argmax}_{\alpha_a} \sum_{s\in S} \alpha_a(s)b(s) \tag{2.10}$$

HSVI maintains upper bound as a lower convex hull of points in $|S|+1$-dimensional space.

Definition 2.8. [16]
Value point $(b, y) \in R^{|S|+1}$ is a $|S|+1$-dimensional point, for which first $|S|$ dimensions represents belief b and last dimension is a value y in this belief.
Υ is a set of value points.

$$\Upsilon = \{(b_0, y_0), ..., (b_n, y_n)\}$$

Value function with value points is defined as

$$V_{\mathrm{UB}}^{\Upsilon}(b) = \min \left\{ \sum_{i=1}^{n} \lambda_i y_i \middle| \lambda \in \mathbb{R}^n_{\geq 0} \colon \sum_{i=1}^{m} \lambda_i b_i = b \right\} \tag{2.11}$$

Similarly to the value function using α-vectors, it is necessary to ensure that the value of each point is always higher or equal to the optimal value.

$$V_{\mathrm{UB}}^{\Upsilon}(b) \geq V^*(b) \tag{2.12}$$

Now lower bound is improved by constructing a new α-vector and adding it to the Γ and the upper bound by constructing a new value point and adding it to the Υ. Both of them may be constructed by using Algorithm 2. Before updating the value function, there have to be some initial values for both the lower and upper bound.

Algorithm 2 Point-Based Update

```
1: function UPDATE($b$)
2:     $\alpha^{a,o} \leftarrow \arg\max_{\alpha \in \Gamma} \sum_{s' \in S} \tau(b, a, o)(s')\alpha(s')$        $\forall (a, o) \in A \times O$
3:     $\alpha^a(s) \leftarrow R(s, a) + \gamma \sum_{o \in O} \sum_{s' \in S} T(s', o|s, a)\alpha^{a,o}(s')$        $\forall (s, a) \in S \times A$
4:     $\Gamma \leftarrow \Gamma \cup \arg\max_{\alpha^a} \sum_{s \in S} b(s)\alpha^a(s)$
5:     $\Upsilon \leftarrow \Upsilon \cup \max_{a \in A} \left[ \sum_{s \in S} b(s)R(s, a) + \gamma \sum_{o \in O} V_{\mathrm{UB}}^{\Upsilon}(\tau(b, a, o)) \sum_{s \in S} \sum_{s' \in S} b(s)T(s', o|s, a) \right]$
6: end function
```

2.3.2 Presolving Value Functions

Lower bound initialization is done by constructing α-vector, which corresponds to playing a uniform strategy. First, we will define initial values in α-vector, which corresponds to receiving the smallest possible reward for each action. Because of the discount factor γ, this value is bounded if the rewards themselves are bounded.

$$\alpha^0(s) = \min_{s' \in S, a \in A} \frac{R(s', a)}{1 - \gamma} \qquad \forall s \in S \tag{2.13}$$

The uniform strategy for an agent is used to update each coordinate in α-vector until the changes between each iteration become less than some $\varepsilon > 0$.

$$\alpha^{t+1}(s) = \sum_{a \in A} \frac{1}{|A|} \left[R(s, a) + \gamma \sum_{o \in O} \sum_{s' \in S} T(s', o|s, a)\alpha^t(s') \right] \tag{2.14}$$

If the uniform strategy is optimal, then the lower bound is presolved to the optimal value. Otherwise, the value function has to be lower than the optimal one.

While presolving the upper bound, the partial observability is removed from the POMDP. MDP techniques like Value Iteration may then be used for the presolve. The points constructed by this fully observable MDP are in the vertices of the belief simplex. The property (2.12) has to hold after the presolve part. The value iteration has to start with greater values than the optimal ones to ensure this. A similar approach to the lower bound in (2.13) is used for the upper bound.

$$y = \max_{s' \in S, a \in A} \frac{R(s', a)}{1 - \gamma} \tag{2.15}$$

Since this is done for each state, there are $|S|$ points after the presolve part. If the POMDP has the same solution as MDP, then after the presolve these points correspond to the optimal solution. Otherwise, the values are higher than the optimal ones because adding information to the agent can't make the value worse. It is possible to use Algorithm 1 or linear program (2.3a)-(2.3d) to presolve upper bound as MDP.

2.3.3 Heuristic Exploration

Solving MDPs by value iteration requires solving each state separately. In POMDPs, we are dealing with continuous belief space. Therefore we would have to solve the game in infinitely many beliefs. It is possible to leverage the fact that the POMDPs dynamics are known in advance, as well as the initial belief, and solve only in those beliefs which are reachable from the initial belief. This does not resolve the issue because the process may still be infinitely long, but we may restrict only to those beliefs that are reachable with at most t steps. The heuristic can be used to solve only relevant parts of the belief space. This could speed up convergence in the initial belief, but it does not guarantee that the gap between the lower and upper bound is less than ε in each belief.

The heuristic chooses the action with the highest expected utility in the upper bound.

$$a^* = \operatorname*{argmax}_{a \in A} \sum_{s \in S} b(s) R(s, a) + \gamma \sum_{o \in O} V_{\mathrm{UB}}^{\Upsilon}(\tau(b, a, o)) \sum_{s \in S} \sum_{s' \in S} b(s) T(s', o | s, a) \tag{2.16}$$

In order to only explore those parts of belief space, which are not yet sufficiently explored, the excess gap is defined as

$$e_t(\tau(b, a, o)) = V_{\mathrm{UB}}^{\Upsilon}(\tau(b, a, o)) - V_{\mathrm{LB}}^{\Gamma}(\tau(b, a, o)) - \varepsilon \gamma^{-t} \tag{2.17}$$

Additionally, the gap for each observation is weighted by the probability of receiving such observation after action a^*.

$$o^* = \operatorname*{argmax}_{o \in O} e_{t+1}(\tau(b, a^*, o)) \sum_{s \in S} \sum_{s' \in S} b(s) T(s', o | s, a) \tag{2.18}$$

The exploration is finished when the excess is negative for the future belief after picking the action-observation pair (a^*, o^*).

$$e_t(\tau(b, a^*, o^*)) < 0 \tag{2.19}$$

Algorithm 3 Heuristic Search Value Iteration

```
1: Initialize $V^{\Gamma}_{\text{LB}}$, $V^{\Upsilon}_{\text{UB}}$
2: while $V^{\Gamma}_{\text{LB}}(b^0) - V^{\Upsilon}_{\text{UB}}(b^0) > \varepsilon$ do
3:     EXPLORE$(b^0, \varepsilon, t)$
4: end while
5:
6: function EXPLORE$(b, \varepsilon, t)$
7:     if $V^{\Gamma}_{\text{LB}}(b) - V^{\Upsilon}_{\text{UB}}(b) > \gamma^{-t}\varepsilon$ then
8:         UPDATE$(b)$
9:         $a^*, o^* \leftarrow$ SELECTAO$(b)$
10:        EXPLORE$(\tau(b, a^*, o^*), \varepsilon, t+1)$
11:        UPDATE$(b)$
12:    end if
13: end function
```

Chapter 3
Game Theory

Game theory is a mathematical framework used for sequential decision making in a multiagent environment called a game by rational agents called players. Each player employs a strategy and receives utility based on its and other player strategies. Each player aims to maximize its utility while also taking into account the rationality of other players.

Definition 3.1. [17]
Normal-form game $\mathcal{G}$ is a tuple $(\mathcal{N}, \mathcal{S}, u)$, where
$\mathcal{N}$ is a set of players $i \in \mathcal{N} = \{1, ..., n\}$, $-i$ denotes all players except i.
$\mathcal{S}$ is a set of all pure strategies $\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2 \times ... \times \mathcal{S}_n$.
u is the utility function $u_i : \mathcal{S} \to \mathbb{R}$, $\forall i \in \mathcal{N}$.

Normal-form games are often described using n-dimensional matrices, where each dimension corresponds to one player's strategy. Each player picks a strategy and then plays according to it. Playing just the pure strategy may not be desirable for some players. Imagine a game of rock-paper-scissors playing just the pure strategy could easily be exploited. We introduce the notion of mixed strategies, where the player alternates between multiple strategies with some probability.

Definition 3.2. [17]
Let $\mathcal{S}_i$ be all pure strategies for player $i \in \mathcal{N}$.
Mixed strategy σ_i is a probability distribution over $\mathcal{S}_i$.

$$\sigma_i = \{p^i \in \mathbb{R}^{|\mathcal{S}_i|} | \sum_{j=1}^{|\mathcal{S}_i|} p^i_j = 1 \wedge p^i_j \geq 0\}$$

Strategy profile σ is then defined as $(\sigma_1, \sigma_2, ..., \sigma_n)$.

The player is incentivized to play the best strategy to maximize utility. This strategy is called a best response. We use $u_i(\sigma_i, \sigma_{-i})$ to denote that player $i \in \mathcal{N}$ expects to receive utility u_i, if it plays mixed strategy σ_i, while other players play mixed strategy σ_{-i}.

Definition 3.3. [17]
Let u be utility function in game $\mathcal{G}$ and σ_{-i} be a strategy for each player except i.
Best response σ^*_i is such strategy, that for player i and $\forall \sigma_i$ holds

$$u_i(\sigma^*_i, \sigma_{-i}) \geq u_i(\sigma_i, \sigma_{-i})$$

When all players play the best response against all other policies, we get a stable solution in which neither player can improve its utility by changing strategy. Such a solution is called Nash equilibrium

Definition 3.4. [18]
Let u be utility function in game $\mathcal{G}$
Nash equilibrium is such strategy profile σ^*, for which holds

$$u_i(\sigma_i^*, \sigma_{-i}^*) \geq u_i(\sigma_i, \sigma_{-i}^*) \quad \forall \sigma_i, \forall i \in \mathcal{N}$$

ε-Nash equilibrium is such strategy profile σ^*, for which holds

$$u_i(\sigma_i^*, \sigma_{-i}^*) \geq u_i(\sigma_i, \sigma_{-i}^*) - \varepsilon \quad \forall \sigma_i, \forall i \in \mathcal{N}, \varepsilon > 0$$

John Nash has proved that each game has at least 1 Nash equilibrium if mixed strategies are allowed [18].

Theorem 3.1. Nash's existence theorem [18]
For each game $\mathcal{G}$, there exists at least 1 strategy profile σ^*, which is Nash equilibrium.

3.1 Example: Battle of sexes

The wife and her husband decide whether they go to ballet or soccer. Man prefers the soccer match, while woman prefers the ballet. However, they will not go anywhere if they do not agree on an evening activity, which is something neither of them wants. Utilities for both of them are presented in Table 3.1, where the first value always describes utility for the wife and the second describes utility for the husband [19].

	Ballet	Soccer
Ballet	3;2	0;0
Soccer	0;0	2;3

Table 3.1. Utilities for both players in battle of sexes

Battle of sexes has 2 pure Nash equilibria and 1 mixed Nash equilibrium.

$$\sigma_1^* = (1, 0) \qquad \sigma_2^* = (1, 0) \qquad u^* = (3, 2) \tag{3.1a}$$
$$\sigma_1^* = (0, 1) \qquad \sigma_2^* = (0, 1) \qquad u^* = (2, 3) \tag{3.1b}$$
$$\sigma_1^* = (0.6, 0.4) \qquad \sigma_2^* = (0.4, 0.6) \qquad u^* = (1.2, 1.2) \tag{3.1c}$$

where $\sigma_i^* = (\mathbb{P}(s_i = \text{Ballet}), \mathbb{P}(s_i = \text{Soccer}))$ and $u^* = (u_1^*(\sigma_1^*, \sigma_2^*), u_2^*(\sigma_1^*, \sigma_2^*))$. We can see that each Nash equilibrium results in a different utility for each player and that only the mixed equilibrium has the same expected utility for both players.

3.2 Zero-sum Games

We mainly focus on a subset of games, namely two-player zero-sum games. In zero-sum games for any mixed strategy, the sum of utilities is zero.

Definition 3.5. [17]
Let $\mathcal{G} = (\mathcal{N}, \mathcal{S}, u)$ be a game.
Zero-sum game is a game for which holds

$$\sum_{i \in \mathcal{N}} u_i(\sigma_i, \sigma_{-i}) = 0 \quad \forall \sigma_i, \sigma_{-i}$$

In two-player zero-sum games, the utility of the first player is the negative utility of the second player. Therefore maximizing utility for the first player is the same as minimizing utility for the second player.

$$u_1(\sigma_1, \sigma_2) + u_2(\sigma_2, \sigma_1) = 0 \Rightarrow u_1(\sigma_1, \sigma_2) = -u_2(\sigma_2, \sigma_1) \tag{3.2}$$

Computing Nash equilibrium in general-sum games is proven to be a PPAD-complete problem [20], but in zero-sum games, it is possible to use the notion of the maxmin and minmax strategies to compute Nash equilibrium in polynomial time by using the linear programming.

Definition 3.6. [17]
Let $\mathcal{G} = (\mathcal{N}, \mathcal{S}, u)$ be a game.
Maxmin strategy for player $i \in \mathcal{N}$ is

$$\operatorname*{argmax}_{\sigma_i} \min_{\sigma_{-i}} u_i(\sigma_i, \sigma_{-i})$$

Definition 3.7. [17]
Let $\mathcal{G} = (\mathcal{N}, \mathcal{S}, u)$ be a game.
Minmax strategy for player $i \in \mathcal{N}$ is

$$\operatorname*{argmin}_{\sigma_i} \max_{\sigma_{-i}} u_i(\sigma_i, \sigma_{-i})$$

In a two-player zero-sum setting, it was proven by John von Neumann [21] that each Nash equilibrium is equal to maxmin and minmax value.

Theorem 3.2. Minimax theorem [21]
In any finite, two-player zero-sum game, in any Nash equilibrium each player receives a payoff that is equal to both its maxmin value and its minmax value.

It is possible to rewrite 3.6 as linear program in a following way

$$\max_{V_i,\sigma_i} V_i \tag{3.3a}$$
$$\text{s.t. } V_i \leq \sum_{s_i \in \mathcal{S}_i} \sigma_i(s_i) u(s_i, s_{-i}) \qquad \forall s_{-i} \in \mathcal{S}_{-i} \tag{3.3b}$$
$$\sum_{s_i \in \mathcal{S}_i} \sigma_i(s_i) = 1 \tag{3.3c}$$
$$\sigma_i(s_i) \geq 0 \qquad \forall s_i \in \mathcal{S}_i \tag{3.3d}$$

In this linear program, value V_i is the expected utility for player $i \in \mathcal{N}$ and σ_i denotes a mixed strategy for the same player. Since we are in the zero-sum setting, expected utility for the other player is $V_i = -V_{-i}$.

3.3 Two-Player Stochastic Games with Simultaneous moves

Before stepping out to partially observable two-player stochastic games, we will focus on games where both players have full information about the state of the game. These games bear a resemblance to Markov decision processes.

Definition 3.8. [22]
Two-player stochastic game with simultaneous moves (SG) is a tuple
$\mathcal{G} = (S, A_1, A_2, T, R, s^{(0)}, \gamma)$ where
S is a finite set of state.
A_1 is a finite set of actions for first player.
A_2 is a finite set of actions for second player.
$T: S \times A_1 \times A_2 \times S \to [0,1]$ is a transitions function, where $T(s'|s, a_1, a_2)$ is a conditional probability that the game transitions from state $s \in S$ to state $s' \in S$ after players play actions $a_1 \in A_1$ and $a_2 \in A_2$.
$R: S \times A_1 \times A_2 \to \mathbb{R}$ is a reward function of the first player, where $R(s, a_1, a_2)$ is a reward given to first player after players play actions $a_1 \in A_1$ and $a_2 \in A_2$ in state $s \in S$.
$s^{(0)} \in S$ is a initial state of the game.
$\gamma \in [0,1]$ is a discount factor of the game.

Each game starts in state $s^{(0)}$ and at each stage t in state $s^{(t)}$ both players choose actions $a_1^{(t)} \in A_1, a_2^{(t)} \in A_2$ simultaneously and the game transitions to the new state $s^{(t+1)}$ based on $T(s^{(t+1)}|s^{(t)}, a_1^{(t)}, a_2^{(t)})$. First player then receives reward $R(s^{(t)}, a_1^{(t)}, a_2^{(t)})$ and second player receives $-R(s^{(t)}, a_1^{(t)}, a_2^{(t)})$.

3.3.1 Value Iteration

Value iteration in MDP updates the expected value for each state in each iteration. The expected value for games has to take into account the policy of both players [23].

$$V^{\pi_1,\pi_2}(s) = \sum_{a_1 \in A_1} \sum_{a_2 \in A_2} \pi_1(s, a_1)\pi_2(s, a_2)\left(R(s, a_1, a_2) + \gamma \sum_{s' \in S} T(s'|s, a_1, a_2)\right) \tag{3.4}$$

With the notion of maxmin and minmax strategies, the resulting Bellman update is

$$\max_{\pi_1}\min_{\pi_2} V^{\pi_1,\pi_2}(s) = \min_{\pi_1}\max_{\pi_2} V^{\pi_1,\pi_2}(s) \tag{3.5}$$

This may be rewritten as a linear program, which updates the value of a single state. Both players may use the same LP to update its value. The only real difference is that the second player receives reward $-R(s, a_1, a_2)$.

$$\max_{V,\pi_1} V(s) \tag{3.6a}$$

$$\text{s.t.}\ \ V(s) \leq \sum_{a_1 \in A_1} \pi_1(s,a_1)R(s,a_1,a_2) + \gamma \sum_{a_1 \in A_1}\sum_{s' \in S} \pi_1(s,a_1)T(s',o|s,a_1,a_2)\hat{V}(s') \quad \forall a_2 \in A_2 \tag{3.6b}$$

$$\sum_{a_1 = A_1} \pi_1(s,a_1) = 1 \tag{3.6c}$$

$$\pi_1(s,a_1) \geq 0 \quad \forall a_1 \in A_1 \tag{3.6d}$$

Value iteration uses the LP (3.6a)-(3.6d) for each state for i iterations or until the largest change between values in each iteration is less then some $\epsilon > 0$. The second variant is shown in Algorithm 4

Algorithm 4 Value Iteration for SGs

```
1:  function VALUE ITERATION($\varepsilon$)
2:      Initialize $V(s)$
3:      $\Delta \leftarrow \varepsilon$
4:      while $\Delta \geq \varepsilon$ do
5:          $\Delta \leftarrow 0$
6:          for $s \in S$ do
7:              $v, \pi_1, \pi_2 \leftarrow$ SOLVE($s$)
8:              $\Delta \leftarrow \max(\Delta, v - V(s))$
9:              $V(s) \leftarrow v$
10:         end for
11:     end while
12: end function
```

3.3.2 Heuristic Search Value Iteration

HSVI was mainly developed for POMDPs. However, it could be slightly modified for solving MDPs and fully observable two-player games. The main idea to keep upper and lower bound remains the same. The initial lower bound and the upper bound is set to the smallest and the highest reward given to the player indefinitely.

$$V_{LB}(s) = \min_{(s,a_1,a_2) \in S \times A_1 \times A_2} \frac{R(s,a_1,a_2)}{1-\gamma} \tag{3.7}$$

$$V_{UB}(s) = \max_{(s,a_1,a_2) \in S \times A_1 \times A_2} \frac{R(s,a_1,a_2)}{1-\gamma} \tag{3.8}$$

HSVI uses excess gap to decide where to explore the state space and whether the exploration should continue or not. This excess gap ensures that in depth i, the gap between the lower and upper bound is not already sufficiently small. The HSVI for the stochastic games with simultaneous moves defines the excess gap same way as the HSVI for OS-POSGs [6].

$$e_t(s) = V_{\mathrm{UB}}(s) - V_{\mathrm{LB}}(s) - \rho(t) \tag{3.9}$$

$$\rho(0) = \varepsilon \tag{3.10}$$

$$\rho(t+1) = \frac{\rho(t) - 2\delta D}{\gamma} \tag{3.11}$$

where $D \in (0, \frac{(1-\gamma)\varepsilon}{2\delta})$ is a neighbourhood parameter. If $D = 0$, the resulting excess is the same as with the HSVI for POMDPs and larger D results in the algorithm exploring deeper. If $D \geq \frac{(1-\gamma)\varepsilon}{2\delta}$, then it may happen that $\rho(t+1) \leq \rho(t)$, which could cause the algorithm to explore indefinitely.

The action a^* of player 1 and the next state s^*, which should be explored from state s is found as

$$(a^*, s^*) = \underset{(a_1, s') \in A_1 \times S}{\operatorname{argmax}} \; \pi_1(a_1) \sum_{a_2 \in A_2} e_t(s') \pi_2(a_2) T(s'|s, a_1, a_2) \tag{3.12}$$

The exploration is stopped when

$$\pi_1(a^*) \sum_{a_2 \in A_2} e_t(s^*) \pi_2(a_2) T(s^*|s, a^*, a_2) \leq 0 \tag{3.13}$$

Full implementation is in Algorithm 5.

Algorithm 5 Heuristic Search Value Iteration for SGs

```
1:  Initialize $V_{\text{LB}}$, $V_{\text{UB}}$
2:  while $V_{\text{LB}}(s^{(0)}) - V_{\text{UB}}(s^{(0)}) > \varepsilon$ do
3:      EXPLORE($s^{(0)}, \varepsilon, t$)
4:  end while
5:
6:  function EXPLORE($s, \rho, t$)
7:      $\pi_1^{\text{LB}}, \pi_2^{\text{LB}}, v^{\text{LB}} \leftarrow$ SOLVELB($s$)
8:      $\pi_1^{\text{UB}}, \pi_2^{\text{UB}} v^{\text{UB}} \leftarrow$ SOLVEUB($s$)
9:      $V_{\text{LB}}(s) = v^{\text{LB}}$
10:     $V_{\text{UB}}(s) = v^{\text{UB}}$
11:     $a^*, s' \leftarrow$ SELECTAS(s)
12:     $g \leftarrow e_t(s') \sum_{a_2 \in A_2} \pi_1^{\text{UB}}(a^*) \pi_2^{\text{LB}}(a_2) T(s'|s, a^*, a_2)$
13:     if $g > 0$ then
14:         $\rho \leftarrow \frac{\rho - 2\delta D}{\gamma}$
15:         EXPLORE($s', \rho, t+1$)
16:         $\pi_1^{\text{LB}}, \pi_2^{\text{LB}}, v^{\text{LB}} \leftarrow$ SOLVELB($s$)
17:         $\pi_1^{\text{UB}}, \pi_2^{\text{UB}} v^{\text{UB}} \leftarrow$ SOLVEUB($s$)
18:         $V_{\text{LB}}(s) = v^{\text{LB}}$
19:         $V_{\text{UB}}(s) = v^{\text{UB}}$
20:     end if
21: end function
```

3.4 One-Sided Partially Observable Stochastic Games

Solving general partially observable stochastic games with two players becomes difficult because each player has to argue its strategy based on the opponents' belief, but this belief is affected based on the belief of the other player. This cycle of arguing about opponents' knowledge is infinite. Therefore it is challenging to compute reasonable strategies. This problem is called the nested beliefs problem [24]. We will focus on a subclass of two-player zero-sum games named one-sided partially observable stochastic games. In these games, one player has imperfect information about the game's current state, while the other one has complete information. We will call them imperfect information player and perfect information player, respectively. Thanks to this property, the perfect information player knows exactly the opponent's belief, and the imperfect information player knows its opponent has this knowledge. Such games may be used to model several security problems [6]. Based on the similarities between one-sided partially observable stochastic games and POMDPs, a variant of the HSVI algorithm was developed for solving these games [5].

Definition 3.9. [5][6]
One-sided partially observable stochastic game (OS-POSG) is a tuple $\mathcal{G} = (S, A_1, A_2, O, T, R, b^{(0)}, \gamma)$ where
S is a finite set of state.
A_1 is a finite set of actions for imperfect information player.
A_2 is a finite set of actions for perfect information player.
O is a finite set of observation for imperfect information player.
$T: S \times A_1 \times A_2 \times O \times S \to [0, 1]$ is a transitions function, where $T(s', o|s, a_1, a_2)$ is a conditional probability that the game transitions from state $s \in S$ to state $s' \in S$ and imperfect information player receives observation $o \in O$ after players play actions $a_1 \in A_1$ and $a_2 \in A_2$.
$R: S \times A_1 \times A_2 \to \mathbb{R}$ is a reward function of imperfect information player, where $R(s, a_1, a_2)$ is a reward given to imperfect information player after players play actions $a_1 \in A_1$ and $a_2 \in A_2$ in state $s \in S$.
$b^{(0)}: S \to \mathbb{R}$ is a initial belief of imperfect information player, where $b^{(0)}(s)$ is probability that the game is in state $s \in S$ at the beginning of the game.
$\gamma \in [0, 1]$ is a discount factor of the game.

Each game starts by sampling single state $s^{(0)} \sim b^{(0)}$. At each stage of the game t both players choose actions $a_1^{(t)} \in A_1$, $a_2^{(t)} \in A_2$ simultaneously and the game transitions to the new state $s^{(t+1)}$, which is shown to the perfect information player, and imperfect information only receives observation $o^{(t)} \in O$ based on $T(s^{(t+1)}, o^{(t)}|s^{(t)}, a_1^{(t)}, a_2^{(t)})$. There is also a reward $R(s^{(t)}, a_1^{(t)}, a_2^{(t)})$ given to the imperfect information player.

Since the belief of an imperfect information player is known to the perfect information player, both of them have their strategies dependent on belief. Imperfect information player has policy $\pi_1(a_1)$ for each of its actions $a_1 \in A_1$. Perfect information player knows the true state of the game, so its policy has to be dependent on the state. We will use $\pi_2(a_2|s)$ to denote the conditional probability of playing $a_2 \in A_2$ if the player is in state $s \in S$. This does not mean there are only $|S|$ policies for a perfect information player based on state.

Because there is one player, which has full information, only the imperfect information player has to have a belief about the environment. In POMDPs, the belief update is dependent only on the dynamics of the environment. However, in OS-POSGs, it depends also on the policy played by the perfect information player.

$$b'(s') = \tau(b, a_1, \pi_2, o)(s') = \frac{\sum_{s \in S} \sum_{a_2 \in A_2} b(s)\pi_2(a_2|s)T(s', o|s, a_1, a_2)}{\sum_{s \in S} \sum_{a_2 \in A_2} \sum_{s'' \in S} b(s)\pi_2(a_2|s)T(s'', o|s, a_1, a_2)} \tag{3.14}$$

3.4.1 Bellman Update

As with the original HSVI for POMDPs, the HSVI for OS-POSGs approximates the solution by tightening the gap between the lower and upper bound. It also updates the value function by adding a point to the upper bound and α-vector to the lower bound after using the Bellman equation to update the values. In OS-POSGs, the expected value has to take into account the strategies of each player.

$$\begin{aligned} V^{\pi_1,\pi_2}(b) = &\sum_{a_1\in A_1}\sum_{a_2\in A_2}\sum_{s\in S}\pi_1(a_1)\pi_2(a_2|s)b(s)R(s,a_1,a_2) \\ &+\gamma\sum_{a_1\in A_1}\sum_{o\in O}\pi_1(a_1)V^{\pi}(\tau(b,a_1,\pi_2,o))\sum_{s\in S}\sum_{a_2\in A_2}\sum_{s'\in S}\pi_2(a_2|s)b(s)T(s',o|s,a_1,a_2) \end{aligned} \tag{3.15}$$

With the notion of maxmin and minmax strategies 3.6 the Bellman update for OS-POSGs is

$$\max_{\pi_1}\min_{\pi_2} V^{\pi_1,\pi_2}(b) = \min_{\pi_2}\max_{\pi_1} V^{\pi_1,\pi_2}(b) \tag{3.16}$$

With the usage of α-vectors, it is possible to perform the Bellman update by using linear program. It still holds that the value function is the lower bound solution

$$\max_{V,\pi_1,\hat{\alpha},\lambda} \sum b(s)V(s) \tag{3.17a}$$

$$\text{s.t. } V(s) \le \sum_{a_1\in A_1}\pi_1(a_1)R(s,a_1,a_2)+\gamma\sum_{a_1\in A_1}\sum_{o\in O}\sum_{s'\in S}T(s',o|s,a_1,a_2)\hat{\alpha}^{a_1,o}(s') \quad \forall(s,a_2)\in S\times A_2 \tag{3.17b}$$

$$\hat{\alpha}^{a_1,o}(s') = \sum_{i=1}^{|\Gamma|}\lambda_i^{a_1,o}\alpha_i(s') \quad \forall(a_1,o,s')\in A_1\times O\times S \tag{3.17c}$$

$$\sum_{i=1}^{|\Gamma|}\lambda_i^{a_1,o} = \pi_1(a_1) \quad \forall(a_1,o)\in A_1\times O \tag{3.17d}$$

$$\sum_{a_1=A_1}\pi_1(a_1) = 1 \tag{3.17e}$$

$$\pi_1(a_1)\ge 0 \quad \forall a_1\in A_1 \tag{3.17f}$$

$$\hat{\lambda}_i^{a_1,o}\ge 0 \quad \forall(a_1,o)\in A_1\times O, 1\le i\le|\Gamma| \tag{3.17g}$$

Dual form of this linear program is

$$\min_{V,\pi_2,\hat{V},\tau} V \tag{3.18a}$$

$$\text{s.t. } V \ge \sum_{s\in S}\sum_{a_2\in A_2}\pi_2(s\wedge a_2)R(s,a_1,a_2)+\gamma\sum_{o\in O}\hat{V}(a_1,o) \quad \forall a_1\in A_1 \tag{3.18b}$$

$$\hat{V}(a_1,o) \ge \sum_{s'\in S}\tau(a_1,o)(s')\alpha_i(s') \quad \forall(a_1,o)\in A_1\times O, 1\le i\le|\Gamma| \tag{3.18c}$$

$$\tau(a_1,o)(s') = \sum_{s\in S}\sum_{a_2\in A_2}T(s',o|s,a_1,a_2)\pi_2(s\wedge a_2) \quad \forall(a_1,o,s')\in A_1\times O\times S \tag{3.18d}$$

$$\sum_{a_2=A_2}\pi_2(s\wedge a_2) = b(s) \quad \forall s\in S \tag{3.18e}$$

$$\pi_2(s\wedge a_2)\ge 0 \quad \forall(s,a_2)\in S\times A_2 \tag{3.18f}$$

These presented linear programs are using α-vectors. However, the upper bound is a lower convex hull of points. This next linear program shows how to receive value for each belief from the value function using another linear program, which also ensures that the upper bound is δ-Lipschitz continuous.

$$\begin{aligned}
V_{\mathrm{UB}}^{\Upsilon}(b) = \min_{\lambda,\Delta,b'} & \sum_{i=1}^{|\Upsilon|} \lambda_i y_i + \delta \sum_{s\in S} \Delta_s && && (3.19a)\\
\text{s.t} & \sum_{i=1}^{|\Upsilon|} \lambda_i b_i(s) = b'(s) && \forall s \in S && (3.19b)\\
& \Delta_s \geq b'(s) - b(s) && \forall s \in S && (3.19c)\\
& \Delta_s \geq b(s) - b'(s) && \forall s \in S && (3.19d)\\
& \sum_{i=1}^{|\Upsilon|} \lambda_i = 1 && && (3.19e)\\
& \lambda_i \geq 0 && 1 \leq i \leq |\Upsilon| && (3.19f)
\end{aligned}$$

This linear program serves as a basis to formulate a dual formulation to handle points instead of α vectors. The δ is a Lipschitz constant specific to each game and it is computed as

$$R_{\min} = \min_{(s,a_1,a_2)\in S\times A_1\times A_2} R(s,a_1,a_2) \tag{3.20}$$

$$R_{\max} = \max_{(s,a_1,a_2)\in S\times A_1\times A_2} R(s,a_1,a_2) \tag{3.21}$$

$$\delta = \frac{R_{\max} - R_{\min}}{2(1-\gamma)} \tag{3.22}$$

Replacing the condition (3.18c) by slightly modified version of linear program (3.19a)-(3.19f) results in dual LP for Bellman update using belief points

$$\begin{aligned}
\min_{V,\pi_2,\hat{V},\tau} \ & V && && (3.23a)\\
\text{s.t.} \ & V \geq \sum_{s\in S}\sum_{a_2\in A_2} \pi_2(s\wedge a_2)R(s,a_1,a_2) + \gamma \sum_{o\in O} \hat{V}(a_1,o) && \forall a_1 \in A_1 && (3.23b)\\
& \hat{V}^{a_1,o} = \sum_{i=1}^{|\Upsilon|} \lambda_i^{a_1,o} y_i + \delta \sum_{s\in S} \Delta^(a_1,o)_s && \forall (a_1,o) \in A_1 \times O && (3.23c)\\
& \sum_{i=1}^{|\Upsilon|} \lambda_i^{a_1,o} b_i(s') = \hat{b}^{a_1,o}(s') && \forall (a_1,o,s') \in A_1 \times O \times S && (3.23d)\\
& \Delta_s^{a_1,o} \geq \hat{b}^{a_1,o}(s') - \tau(a_1,o)(s') && \forall (a_1,o,s') \in A_1 \times O \times S && (3.23e)\\
& \Delta_s^{a_1,o} \geq \tau(a_1,o)(s') - \hat{b}^{a_1,o}(s') && \forall (a_1,o,s') \in A_1 \times O \times S && (3.23f)\\
& \tau(a_1,o)(s') = \sum_{s\in S}\sum_{a_2\in A_2} T(s',o|s,a_1,a_2)\pi_2(s\wedge a_2) && &&\\
& && \forall (a_1,o,s') \in A_1 \times O \times S && (3.23h)\\
& \sum_{i=1}^{|\Upsilon|} \lambda_i = 1 && && (3.23g)\\
& \sum_{a_2=A_2} \pi_2(s\wedge a_2) = b(s) && \forall s \in S && (3.23i)\\
& \pi_2(s\wedge a_2) \geq 0 && \forall (s,a_2) \in S \times A_2 && (3.23j)\\
& \lambda_i \geq 0 && 1 \leq i \leq |\Upsilon| && (3.23k)
\end{aligned}$$

These Bellman updates may then be used to create a new α-vector and new point in each stage of exploration. We will call these stages stage games.

3.4.2 Presolving

Before solving the game, there have to be some values as upper and lower bounds to our solution. The same idea as with the original HSVI for POMDPs applies for OS-POSGs, and we initialize a single α-vector and $|S|$ belief points, where each belief point is a pure belief of a single state.

$$\alpha(s) = R_{\text{min}} \quad \forall s \in S \tag{3.24}$$

$$y_i = R_{\text{max}} \tag{3.25}$$

Now the bounds are improved by solving the fully observable variant of a given game, using value iteration (3.6a)-(3.6d) in the upper bound and by playing uniform strategy as an imperfect information player in the lower bound. The better the solution from the presolve part, the less computationally expensive the HSVI itself is.

3.4.3 Exploration

The exploration of HSVI for OS-POSGs is similar to the exploration defined for SGs. The excess gap is computed for belief rather then state, but the ρ parameter and its update remains the same as (3.10), (3.11)

$$e_t(b) = V_{\text{UB}}(b) - V_{\text{LB}}(b) - \rho(t) \tag{3.26}$$

Exploration is dependent on the policies of both players. It uses the policy of imperfect information player from the upper bound and the policy of perfect information player from the lower bound. The optimal action and observation maximize the excess gap weighted by the probability of reaching such belief.

$$(a_1^*, o^*) = \underset{(a_1,o)\in A\times O}{\text{argmax}}\ \pi_1(a_1)e_t(\tau(b, a_1, \pi_2^{\text{LB}}, o)) \sum_{s\in S} \sum_{a_2\in A_2} \sum_{s'\in S} \pi_2(a_2|s)b(s)T(s', o|s, a_1, a_2) \tag{3.27}$$

The exploration is again stopped when

$$\pi_1(a_1^*)e_t(\tau(b, a_1^*, \pi_2^{\text{LB}}, o^*)) \sum_{s\in S} \sum_{a_2\in A_2} \sum_{s'\in S} \pi_2(a_2|s)b(s)T(s', o^*|s, a_1^*, a_2) \leq 0 \tag{3.28}$$

Algorithm 6 Heuristic Search Value Iteration for OS-POSGs

```
1:  Initialize $V_{\text{LB}}^{\Gamma}$, $V_{\text{UB}}^{\Upsilon}$
2:  while $V_{\text{LB}}^{\Gamma}(b^{(0)}) - V_{\text{UB}}^{\Upsilon}(b^{(0)}) > \varepsilon$ do
3:      EXPLORE($b^{(0)}, \varepsilon, t$)
4:  end while
5:
6:  function EXPLORE($b, \rho, t$)
7:      $\pi_1^{\text{LB}}, \pi_2^{\text{LB}}, \alpha \leftarrow$ SOLVELB($b$)
8:      $\pi_1^{\text{UB}}, \pi_2^{\text{UB}}, y \leftarrow$ SOLVEUB($b$)
9:      UPDATE($b, \alpha, y$)
10:     $a^*, o^* \leftarrow$ SELECTAO($b$)
11:     $g \leftarrow \pi_1^{\text{UB}}(a_1^*)e_t(\tau(b, a_1^*, \pi_2^{\text{LB}}, o^*)) \sum_{s\in S} \sum_{a_2\in A_2} \sum_{s'\in S} \pi_2^{\text{LB}}(a_2|s)b(s)T(s', o^*|s, a_1^*, a_2)$
12:     if $g > 0$ then
13:         $\rho \leftarrow \frac{\rho - 2\delta D}{\gamma}$
14:         EXPLORE($\tau(b, a^*, \pi_2^{\text{LB}}, o^*), \rho, t+1$)
15:         $\pi_1^{\text{LB}}, \pi_2^{\text{LB}}, \alpha \leftarrow$ SOLVELB($b$)
16:         $\pi_1^{\text{UB}}, \pi_2^{\text{UB}}, y \leftarrow$ SOLVEUB($b$)
17:         UPDATE($b, \alpha, y$)
18:     end if
19: end function
```

3.5 Quantal Response Equilibrium

Nash equilibrium gives us the expected reward for a player and optimal strategy while playing against a rational and optimally playing opponent. However, knowing the exact policy may not always be necessary, so we could use a less computationally demanding methods, which are only suboptimal. One of these methods is called quantal response equilibrium (QRE) [25].

While computing QRE, a player updates its strategy based on expected utilities, and it also assumes that other players act in a similar fashion. Quantal response equilibrium is a fixed point of this process.

Let us assume that player $i \in \mathcal{N}$ does observe utility with some error for each of its pure strategies $s_i \in \mathcal{S}_i$. Therefore the utility becomes

$$\hat{u}_i(\sigma) = \hat{u}_i(s_i, \sigma^*_{-i}) = u_i(s_i, \sigma^*_{-i}) + \varepsilon_{s_i} \tag{3.29}$$

where ε_{s_i} is a error for player $i \in \mathcal{N}$ when playing strategy s_i and opponents play a best response σ^*_{-i}. Furthermore ε_i is a vector containing ε_{s_i} for each pure strategy s_i and is distributed according to joint distribution with density function $f_i(\varepsilon_i)$. Function f is called admissible if marginal distribution of f_i exists for each ε_{s_i} and expected value is zero $\mathbb{E}(\varepsilon_{s_i}) = 0$ for each player $i \in \mathcal{N}$. We assume that player i will choose a best pure strategy s^*_i, which yields in $\hat{u}_i(s^*_i, \sigma^*_{-i}) \geq \hat{u}_i(s_i, \sigma^*_{-i}), \forall s_i \in \mathcal{S}_i$.

We define region $R_{s^*_i}(u) \subseteq \mathbb{R}^{|S_i|}$, that specifies a region of errors which has s^*_i as a strategy that should be played by player $i \in \mathcal{N}$ if the expected utilities are u.

$$R_{s^*_i}(u) = \{\varepsilon_i \in \mathbb{R}^{|S_i|} | u_i(s^*_i, \sigma^*_{-i}) + \varepsilon_{s^*_i} \geq u_i(s_i, \sigma^*_{-i}) + \varepsilon_{s_i} \quad \forall s_i \in S_i\} \tag{3.30}$$

We may then define $\varsigma_{s_i}(u)$, which is a probability that player i will choose pure strategy s_i, if the expected utilities are u.

$$\varsigma_{s_i}(u) = \int_{R_{s_i}(u)} f(\varepsilon) d\varepsilon \tag{3.31}$$

We use σ_{s_i} to denote probability of playing strategy $s_i \in \mathcal{S}_i$ in strategy profile σ by player $i \in \mathcal{N}$.

Definition 3.10. [25]
Let $\mathcal{G} = (\mathcal{N}, \mathcal{S}, u)$ be a normal-form game and let f be admissible.
Quantal response equilibrium (QRE) is any mixed strategy profile σ for which holds

$$\sigma_{s_i} = \varsigma_{s_i}(u(\sigma)) = \int_{R_{*_i}(u)} f(\varepsilon) d\varepsilon$$

In QRE, the expected utility of strategy s_i is positively correlated with probability of playing that strategy.

Theorem 3.3. Existence of quantal response equilibrium [25]
In any normal-form game $\mathcal{G} = (\mathcal{N}, \mathcal{S}, u)$ and admissible f, there exists a quantal response equilibrium.

To compute QRE it is necessary to define admissible function f. We will use logit equilibrium in this work, which is parameterized by a single parameter $\lambda \geq 0$. If for each player the following equation holds, then we have logit QRE [25].

$$\sigma_{s_i}(u) = \frac{e^{\lambda u_{s_i}(\sigma)}}{\sum_{s_i' \in S_i} e^{\lambda u_{s_i'}(\sigma)}} \tag{3.32}$$

If $\lambda = 0$, the result is a uniform mixed strategy, if $\lambda = \infty$ the result is a uniform mixed strategy between actions with the highest expected utility.

Since QRE is a fixed point, it is possible to iteratively update strategies $\sigma^{(t)}$ at each step t out of expected values computed from $\sigma^{(t-1)}$, until convergence. We initialize strategy to the uniform one for each player.

$$\sigma_{s_i}^{(1)} = \frac{1}{|S_i|} \tag{3.33}$$

Then the new mixed strategy is computed from utility based on (3.32).

$$\sigma_{s_i}^{\prime(t+1)} = \frac{e^{\lambda u_{s_i}(\sigma^{(t)})}}{\sum_{s_i' \in S_i} e^{\lambda u_{s_i'}(\sigma^{(t)})}} \tag{3.34}$$

To ensure the strategy convergence, the strategies are summed by using cummulative moving average [7].

$$\sigma_{s_i}^{(t+1)} = \frac{t\sigma_{s_i}^{(t)} + \sigma_{s_i}^{\prime(t+1)}}{t+1} \tag{3.35}$$

This process continues for T iterations or until the difference between steps becomes smaller than some $\varepsilon_{\mathrm{QRE}} > 0$.

$$|\sigma_{s_i}^{(t+1)} - \sigma_{s_i}^{(t)}| < \varepsilon_{\mathrm{QRE}} \qquad \forall s_i \in \mathcal{S}_i,\ \forall i \in \mathcal{N} \tag{3.36}$$

3.6 Regret Minimization

When computing both Nash and quantal response equilibrium, the goal is to maximize the expected value for each player against the best response of the opponent. However opponent may not play according to a best response, but it may play entirely randomly. It may be better for the agent to minimize how much it can lose by playing according to some strategy instead of maximizing how much it gains by playing according to the same strategy.

Definition 3.11. [17]
Let $\mathcal{G} = (\mathcal{N}, \mathcal{S}, u)$ be a normal-form game.
Regret for playing strategy σ_i if the remaining players play σ_{-i} is defined as

$$r_i = \max_{s_i \in S_i} u_i(s_i, \sigma_{-i}) - u_i(\sigma_i, \sigma_{-i})$$

The regret says how much we could gain by playing the pure strategy s_i instead of strategy σ_i against the fixed strategy σ_{-i} of all the other players.

Suppose an online learning algorithm, which computes a mixed strategy profile $\sigma^{(t)}$ at each step $t \in \{1, \ldots, T\}$, the external average regret is defined as

$$R_i^{(T)} = \frac{1}{T} \max_{\sigma_i^*} \sum_{t=1}^{T} u_i(\sigma_i^*, \sigma_{-i}^{(t)}) - u_i(\sigma^{(t)}) \tag{3.37}$$

If the average regret converges to 0 for each player $i \in \mathcal{N}$, then the algorithm is called no-regret or regret minimization algorithm.

$$\lim_{T \to \infty} R_i^{(T)} = 0 \tag{3.38}$$

Chapter 4
Variance Reduction in Reinforcement Learning

Reinforcement learning is a machine learning method that aims to find the best possible sequence of decisions in an environment. Reinforcement learning does not need to have the exact model of the environment, but it learns the policy from simulating the game multiple times. Therefore the agent may view the environment as a black box and acts just according to the observations and rewards it receives. The fundamental problem in reinforcement learning is when to explore the environment to get better knowledge and when to use that knowledge to maximize the reward. This is called *exploration-exploitation dilemma.* If the agent only focuses on exploration, it simply chooses random action, and the expected reward has a substantial variance. On the other hand, if the agent only focuses on the exploitation and plays such actions with the highest expected reward, it may easily miss a better solution because we do not explore some parts of the game that could be more rewarding.

To reduce the variance from the naive random exploration, some algorithms stores a value function like Q-learning or SARSA, which uses so-called Q-values that assign value to each state-action pair based on expected reward [26], [27]. Other algorithms use Monte Carlo sampling based on the current policy of an agent [28]. Both of these approaches still suffer from high variance, and several methods were introduced to reduce it.

This chapter will introduce the essential variance reduction technique of control variates. Furthermore, we will describe multiple already used variance reduction in reinforcement learning and argue about its usage in HSVI for OS-POSGs.

4.1 Control Variates

The naive way to reduce variance is by using more samples. However, this increases the runtime linearly with the number of samples. Control variates can reduce variance without increasing sample size and with only slight computational overhead by exploiting the information about the error from samples.

Let X be a random variable, which has a mean μ, which we try to estimate from n samples as

$$\mathbb{E}[X] = \mu_X \approx \hat{\mu}_X = \frac{1}{n}\sum_{i=1}^{n} X_i \tag{4.1}$$

We use X_i to denote the value of X in the i-th sample. Let us suppose that we additionally have a different random variable Y, which is jointly distributed with X, and we know its mean.

$$\mathbb{E}[Y] = \mu_Y \tag{4.2}$$

The main idea of control variates is to use the error between the known mean μ_Y and estimated mean $\hat{\mu}_Y$, which is computed from samples of Y. We now construct a new random variable with additional parameter $\lambda \in \mathbb{R}$.

$$\bar{X} = X + \lambda(Y - \mu_Y) \tag{4.3}$$

We now show that the estimated mean is the same for both X and $\bar{X}$.

$$\mu_{\bar{X}} = \mathbb{E}[\bar{X}] = \mathbb{E}[X+\lambda(Y-\mu_Y)] = \mathbb{E}[X]+\lambda(\mathbb{E}[Y]-\mathbb{E}[\mu_Y]) = \mu_X+\lambda(\mu_Y-\mu_Y) = \mu_X \quad (4.4)$$

We further investigate the variance of $\bar{X}$.

$$\operatorname{Var}(\bar{X}) = \operatorname{Var}(X) + \lambda^2 \operatorname{Var}(Y) + 2\lambda \operatorname{Cov}(X, Y) \quad (4.5)$$

We may see that the resulting variance depends on the parameter λ, and if we choose it poorly, the variance of $\bar{X}$ may be greater than the variance of X. Finding the best choice of λ is a simple optimization task.

$$\lambda^* = \operatorname*{argmin}_{\lambda} \operatorname{Var}(X) + \lambda^2 \operatorname{Var}(Y) + 2\lambda \operatorname{Cov}(X, Y) \quad (4.6)$$

This may be solved by differentiation.

$$\frac{d \operatorname{Var}(\bar{X})}{d\lambda} = 2\lambda^* \operatorname{Var}(Y) + 2 \operatorname{Cov}(X, Y) = 0 \rightarrow \lambda^* = -\frac{\operatorname{Cov}(X, Y)}{\operatorname{Var}(Y)} \quad (4.7)$$

4.1.1 Example: Estimating Definite Integral

Let us suppose we want to estimate the definite integral of $\sin(x)$ from 0 to 1 via Monte Carlo integration.

$$\int_0^1 \sin(x)dx \quad (4.8)$$

We will use 1000 uniformly sampled values from interval $[0, 1]$ to estimate the value of this integral with regular Monte Carlo integration and then with Monte Carlo integration with control variate. Usual Monte Carlo integration is

$$\int_0^1 \sin(x)dx \approx \frac{1}{n}\sum_{i=1}^{n} \sin(x_i) \quad (4.9)$$

We will use the identity function for our control variate.

$$f(x) = x \quad (4.10)$$

$$\int_0^1 x = \frac{1}{2} \quad (4.11)$$

Our control variate estimation is then

$$\int_0^1 \sin(x)dx \approx I = \frac{1}{n}\sum_{i=1}^{n} \left(\sin(x_i) + \lambda(x_i - \frac{1}{2}) \right) \quad (4.12)$$

We will try to estimate this definite integral with four different settings of λ. $\lambda = -0.85578$ is an estimation of optimal parameter setting.

λ	I	Variance
0	0.452	0.0612
1	0.4432	0.2871
-2	0.4696	0.1097
-0.85578	0.4596	0.0003

Table 4.1. Results of Monte Carlo integration with control variates

The real value is

$$\int_0^1 \sin(x)dx \approx 0.4597 \quad (4.13)$$

4.2 Reducing Reward Variance

If the rewards in the environment are corrupted or stochastic, the reward may be seen as a random variable. In games, this may occur even if the rewards are deterministic because the rewards depend on other players' strategies. In [29] it was suggested to keep the reward estimate, which is then used to compute the value function, instead of the immediate reward given to the player. It was proven that this approach reduces variance in the tabular domain, and it was also empirically shown that even in longer MDPs with stochastic rewards, the variance is reduced.

4.3 Baselines for values

It was suggested by [30] to improve the algorithm Monte Carlo counterfactual regret minimization (MCCFR) for solving two-player imperfect information games. This algorithm samples playthroughs of the game (one route in the tree) and then updates each visited state based on the received reward. In the original implementation of MC-CFR [28], the update to state is done only by the single action, which was played in the current playthrough. Variance reduction is employed by taking into account values from all possible actions in a given state based on previous visits of this state.

We will define how the baselines in [30] would work in fully observable setting, while the original implementation is for imperfect information games. At each iteration n of MCCFR the single trajectory $s^{(1)}a_1^{(1)}a_2^{(1)}s^{(2)}\dots a_1^{(T-1)}a_2^{(T-1)}s^{(T)}$ is sampled and then for each state $s^{(t)}$ from state $s^{(T-1)}$ to $s^{(1)}$, the algorithm computes utility from trajectory and expected utility for each action in a following way

$$u_i(s^{(t)}, a_i^{(t)}) = \sum_{i=t}^{T-1} \gamma^{i-t} R(s^{(t)}, a_1^{(t)}, a_2^{(t)}) \tag{4.14}$$

$$\hat{u}_i^{(n)}(s^{(t)}, a_i) = \begin{cases} b^{(n)}(s^{(t)}, a_i) + \frac{u(s^{(t)}, a_i^{(t)}) - b^{(n)}(s^{(t)}, a_i)}{\bar{\pi}_i(s^{(t)}, a_i)} & \text{if } a_i = a_i^{(t)} \\ b^{(n)}(s^{(t)}, a_i) & \text{else} \end{cases} \tag{4.15}$$

where $\bar{\pi}_i$ is a sampling policy of a player i, this policy is not the same as the players policy, and $b(s^{(t)}, a_i)$ is a baseline, which is updated each time that given state-action pair is sampled.

$$b^{(n+1)}(s^{(t)}, a_i^{(t)}) = (1-\alpha) b^{(n)}(s^{(t)}, a_i^{(t)}) + \alpha u(s^{(t)}, a_i^{(t)}) \tag{4.16}$$

The real expected value, which would be known if we sampled all the possible actions, is approximated by baseline b. Therefore the optimal baseline would be the actual expected reward after picking such action.

Chapter 5
Two-Player Stochastic Games with Simultaneous moves

Two-player stochastic games with simultaneous moves (SGs) are closely related OS-POSGs, but without partial observability, it is much easier to solve them. This means that if the significant variance is present in these games, it is most likely present in OS-POSGs. This chapter will present a slightly modified version of value iteration for SGs, which uses stochastic exploration to update state values. We will also show how to avoid linear programming while computing Nash equilibrium in each state and instead use two iterative approaches based on quantal response equilibrium and regret minimization. In the last part of this chapter, we will show how the variance reduction baselines may be used with our version of regret minimization.

5.1 Value Iteration with Stochastic Exploration

Usual value iteration updates each state in each iteration. However, some states may be reached only with a small probability. Such states only slightly affect the initial state value and therefore do not have to be approximated as closely as some other states. In order to solve the game in such states, which are more likely to be reached from the initial state, we could sample the actions according to the policies of both players. If some action has zero or near-zero probability in policy, it is never sampled. We offset the sampling policy to ensure such actions may be sampled. One way of doing so is ϵ-offset.

$$\hat{\pi}_i(s, a_i) = (1-\epsilon)\pi_i(s, a_i) + \epsilon\frac{1}{|A_i|} \qquad \forall a + i \in A_i \tag{5.1}$$

where π_i is a policy for player i.

The algorithm then starts in the initial state and samples actions for both players based on (5.1) and transitions until the terminal state or maximum depth is reached. Then the value is updated similarly to the usual value iteration. We call this algorithm value iteration with stochastic exploration (VISE) and it is presented in Algorithm 7.

5.2 Quantal Response Equilibrium

In the previous section, we have presented two variants of value iteration and one variant of HSVI. Usually, these algorithms solve the game in a given state by solving linear program (3.6a)-(3.6d), which computes the Nash equilibrium. Computing Nash equilibrium via linear programs may be computationally demanding both time-wise and space-wise. Quantal response equilibrium is a different type of equilibrium, which is computed by an iterative approach. Therefore it avoids constructing and solving the linear program.

Algorithm 7 Value Iteration with Stochastic Exploration for SGs

```
function VALUE ITERATION STOCHASTIC EXPLORE($T, D_{\max}, \epsilon$)
    Initialize $V(s)$
    for $t = \{1, 2, 3, \dots, T\}$ do
        STOCHASTICEXPLORE($s^0, \epsilon, 1, D_{\max}$)
    end for
end function

function STOCHASTICEXPLORE($s, \epsilon, D, D_{\max}$)
    if $D < D_{\max}$ then
        $s_{\text{new}} \leftarrow$ Sample according to $\epsilon$-offset policies
        STOCHASTICEXPLORE($s_{\text{new}}, \epsilon, D+1, D_{\max}$)
        $v, \pi_1, \pi_2 \leftarrow$ SOLVE($s$)
    end if
end function
```

When reached a state $s \in S$ for t-th time, each player computes the expected value against the opponents current policy as follows

$$Q^{(t)}(s, a_1) = \sum_{a_2 \in A_2} \pi_2^{(t)}(s, a_2) \left(R(s, a_1, a_2) + \gamma \sum_{s' \in S} T(s'|s, a_1, a_2) V(s') \right) \quad \forall a_1 \in A_1 \tag{5.2}$$

$$Q^{(t)}(s, a_2) = -\sum_{a_1 \in A_1} \pi_1^{(t)}(s, a_1) \left(R(s, a_1, a_2) + \gamma \sum_{s' \in S} T(s'|s, a_1, a_2) V(s') \right) \quad \forall a_2 \in A_2 \tag{5.3}$$

Since the value V is stored only for player 1, thanks to the zero-sum property, it is enough to use the negative value $-V$ to compute value Q for the second player.

The initial policy is set as a uniform distribution over all actions.

$$\pi_i^{(1)}(s, a_i) = \frac{1}{|A_i|} \tag{5.4}$$

New policy increment is then computed as quantal response equilibrium in the following way

$$\bar{\pi}_i(s, a_i) = \frac{e^{\lambda Q^{(t)}(s, a_i)}}{\sum_{a_i' \in A_i} e^{\lambda Q^{(t)}(s, a_i')}} \tag{5.5}$$

New average policy is computed by moving average.

$$\pi_i^{(t+1)}(s, a_i) = \frac{t\pi_i^{(t)}(s, a_i) + \bar{\pi}_i(s, a_i)}{t+1} \tag{5.6}$$

Parameter λ affects how the differences between values are amplified. High λ results in a higher policy increment for action, which has a maximal expected value. If $\lambda = \infty$, then the resulting average policy converges to the Nash equilibrium.

5.3 Regret Minimization

Instead of approximating Nash equilibrium by maximizing the value, it could be achieved by minimizing the regret. Regret minimization is also an iterative approach similar to QRE. With QRE, it is necessary for each state to store the value, the average policy, and how many times this state was already solved. Regret minimization requires additional information about regret for each action in each state. Cumulative regret at the beginning of the solving stage is defined as

$$R^{(0)}(s, a_i) = 0 \qquad \forall s \in S,\ \forall a_i \in A_i,\ \forall i \in \{1, 2\} \tag{5.7}$$

Unlike the QRE, where the average opponent policy is used to compute expected values, the regret minimization employs a method called regret matching [31], which computes the policy in the following way

$$\bar{\pi}_i^{(t)}(s, a_i) = \begin{cases} \frac{R^{(t)}(s,a_i)}{\sum_{a_i' \in A_i} R^{(t)}(s,a_i')} & \text{if } \sum_{a_i' \in A_i} R^{(t)}(s, a_i') > 0 \\ \frac{1}{|A_i|} & \text{else} \end{cases} \tag{5.8}$$

Regret of deviating from this regret matching strategy is then computed as

$$r_i(s, a_i) = \sum_{a_i' \in A_i} \bar{\pi}_i^{(t)}(s, a_i') Q^{(t)}(s, a_i') - Q^{(t)}(s, a_i) \tag{5.9}$$

To avoid negative policy in (5.8), the cumulative regret has to be non-negative. To ensure this, we clamp the cumulative regret to 0 if it is negative.

$$R^{(t)}(s, a_i) = \begin{cases} R^{(t-1)}(s, a_i) + r_i(s, a_i) & \text{if } R^{(t-1)}(s, a_i) + r_i(s, a_i) > 0 \\ 0 & \text{else} \end{cases} \tag{5.10}$$

Policy update remains the same as in (5.6).

5.3.1 Regret Minimization with baselines

Action baselines significantly improved the original MCCFR [30] by reducing the variance in regrets. We tried similair idea with our regret minimization algorithm, which is based on value iteration. Unlike the MCCFR, which does only hold the regrets for each state, the value iteration holds expected value associated to the state. We will use the baseline in a similar fashion to compute the regrets and update the expected value according to the policy found by regret minimization. The baseline affected value is defined the same as for the MCCFR.

$$\hat{Q}^{(t)}(s, a_i) = \begin{cases} b^{(t)}(s, a_i) + \frac{Q^{(t)}(s,a_i) - b^{(t)}(s,a_i)}{\bar{\pi}_i(s,a_i)} & \text{if } a_i \text{ was sampled} \\ b^{(t)}(s, a_i) & \text{else} \end{cases} \tag{5.11}$$

As with the MCCFR the sampling policy $\bar{\pi}_i$ is different then the actual policy of player i. The value $Q^{(t)}$ is the same as in (5.2) and (5.3). These values are then used instead of $Q^{(t)}$ to compute regret in (5.9).

The MCCFR always updates the baseline of sampled action, because that is the only value available. However in the value iteration we have values already available for each state. We leverage this additional information and update baseline for each action in state s.

$$b^{(t+1)}(s, a_i) = (1 - \alpha) b^{(t)}(s, a_i) + \alpha Q^{(t)}(s, a_i)$$

Chapter 6
One-Sided Partially Observable Stochastic Games

While solving one-sided partially observable stochastic games to arbitrary precision with HSVI, the most computationally expensive part is constructing and solving linear programs. To possibly speed up the convergence of HSVI it is necessary to find different approaches to approximate Nash equilibria, which do not use linear programs. However these approximations increases variance in both value function and policies. This chapter presents a way to remove linear programs from the HSVI algorithm and instead use regret minimization or the computation of quantal response equilibrium to approximate Nash equilibrium.

6.1 Iterative Stage Games Solving

Quantal response equilibrium and regret minimization may both be used to solve OS-POSGs. When using these methods while solving SGs, each state had a separate value, the average policy, visits, and regret, but OS-POSGs have continuous belief space instead of states. Solving imperfect information games with reinforcement learning techniques, like counterfactual regret minimization [31], deals with this issue through the concept of information sets. Each information set consists of indistinguishable states by the player, actions the player played, and observations it received. However, this approach is not scalable enough since the amount of information sets grows exponentially with the game's length.

We have decided to store these values separately for the lower and upper bound. Since the lower bound is constructed by α-vectors, for each belief, we may assign α-vector based on

$$\alpha^* = \operatorname*{argmax}_{\alpha \in \Gamma} \sum_{s \in S} b(s)\alpha(s) \tag{6.1}$$

The policies and regrets are then defined for each α-vector α and player.

$$\pi_1^{(t)}(\alpha, a_1) \qquad \pi_2^{(t)}(\alpha, s, a_2) \tag{6.2}$$

$$R_1^{(t)}(\alpha, a_1) \qquad R_2^{(t)}(\alpha, s, a_2) \tag{6.3}$$

Each time the stage game in belief b is about to be solved we start with policy $\pi_i^{(t)}$, regret $R_i^{(t)}$ and visits t from the α-vector α^*. After solving the stage game with any of the approximate method, the new α-vector is constructed with policy $\pi_i^{(t+1)}$, regret $R_i^{(t+1)}$ and visits $t+1$.

The upper bound has to be handled differently because the value function is not entirely made from α-vector but just from points. We decided to discretize the belief space and always store policy, regret, and visits in the nearest rounded belief. The policies and regrets are defined for each discretized belief point b^* and player.

$$\pi_1^{(t)}(b^*, a_1) \qquad \pi_2^{(t)}(b^*, s, a_2) \tag{6.4}$$

$$R_1^{(t)}(b^*, a_1) \qquad R_2^{(t)}(b^*, s, a_2) \tag{6.5}$$

After updating values $\pi_i^{(t+1)}$ and $R_i^{(t+1)}$ we add new α-vector with these values, but the values in the old α-vector remains the same. However when adding new point, these values are updated in the discretized belief point. When using the iterative algorithms for stage game solving, the expected value for each action is defined differently for each player based one belief.

$$\begin{aligned} Q(b, a_1) = & \sum_{a_2 \in A_2} \sum_{s \in S} \pi_2(a_2|s) b(s) R(s, a_1, a_2) \\ & + \gamma \sum_{o \in O} V(\tau(b, a_1, \pi_2, o)) \sum_{s \in S} \sum_{a_2 \in A_2} \sum_{s' \in S} \pi_2(a_2|s) b(s) T(s', o|s, a_1, a_2) \end{aligned} \tag{6.6}$$

$$\begin{aligned} Q(b, s, a_2) = & \sum_{a_1 \in A_1} \pi_1(a_1) R(s, a_1, a_2) \\ & + \gamma \sum_{a_1 \in A_1} \sum_{o \in O} V(\tau(b, a_1, \pi_2, o)) \sum_{s' \in S} \pi_1(a_1) T(s', o|s, a_1, a_2) \end{aligned} \tag{6.7}$$

$V(b)$ is either $V_{\text{UB}}^{\Upsilon}(b)$ or $V_{\text{LB}}^{\Gamma}(b)$ for upper bound and lower bound respectively. Computation of quantal response equilibrium and regret minimization in OS-POSGs is then done the same way as in SGs in (5.5) and (5.8).

6.1.1 Example: Adding new α-vector and belief point to the value function

Let us suppose we have a value functions Γ and Υ as in 6.1 and we are about to solve the stage game with belief

$$b'(s_1) = 0.61 \quad b'(s_2) = 0.39 \tag{6.8}$$

Before we start to solve the lower bound stage game, we find the α-vector α^* according to

$$\alpha^* = \operatorname*{argmax}_{\alpha \in \Gamma} \sum_{s \in S} b'(s) \alpha(s) \tag{6.9}$$

In this case it is α_3. Out of α^3 we get $\pi_1^{(t)}(\alpha_3, a_1)$, $R_1^{(t)}(\alpha_3, a_1)$ $\forall a_1 \in A_1$ and $\pi_2^{(t)}(\alpha_3, s, a_2)$, $R_2^{(t)}(\alpha_3, s, a_2)$ $\forall (s, a_2) \in S \times A_2$. These values are then used to solve the lower bound stage game. The result is a new α-vector α_5, which is computed using new policies $\pi_1^{(t+1)}(\alpha_5, a_1)$, $\pi_2^{(t+1)}(\alpha_5, s, a_2)$. Regrets $R_1^{(t+1)}(\alpha_5, a_1)$, $R_2^{(t+1)}(\alpha_5, s, a_2)$ are also saved with this vector. Value function is simply updated as

$$\Gamma_{\text{new}} = \Gamma \cup \alpha_5. \tag{6.10}$$

Updated lower bound value function with α_5 is shown in 6.2.

For upper bound also find nearest discretized belief as in 6.3, which is

$$b^*(s_1) = 0.6 \quad b^*(s_2) = 0.4 \tag{6.11}$$

This discretized belief is not the closest neighbour to any point in value function. Therefore it is initialized as

$$\pi_1^{(1)}(b^*, a_1) = \frac{1}{|A_1|} \qquad \forall a_1 \in A_1 \tag{6.12}$$

$$\pi_2^{(1)}(b^*, s, a_2) = \frac{1}{|A_2|} \qquad \forall (s, a_2) \in S \times A_2 \tag{6.13}$$

$$R_1^{(1)}(b^*, a_1) = 0 \qquad \forall a_1 \in A_1 \tag{6.14}$$

$$R_2^{(1)}(b^*, s, a_2) = 0 \qquad \forall (s, a_2) \in S \times A_2 \tag{6.15}$$

Similarly we end up with new point y, computed from new policies $\pi_1^{(2)}(b, a_1)$, $\pi_2^{(2)}(b, s, a_2)$ and regrets $R_1^{(2)}(b, a_1), R_2^{(2)}(b, s, a_2)$. New belief point is added the the value function as shown in 6.4.

$$\Upsilon_{\text{new}} = \Upsilon \cup (b, y)$$

The policy and regret are updated in discretized belief space as $\pi_1^{(2)}(b^*, a_1)$, $\pi_2^{(2)}(b^*, s, a_2)$, $R_1^{(2)}(b^*, a_1), R_2^{(2)}(b^*, s, a_2)$. Note that policies and regrets were computed in belief b, but they are updated in value function in belief b^*. Updated lower and upper bound are shown in 6.5.

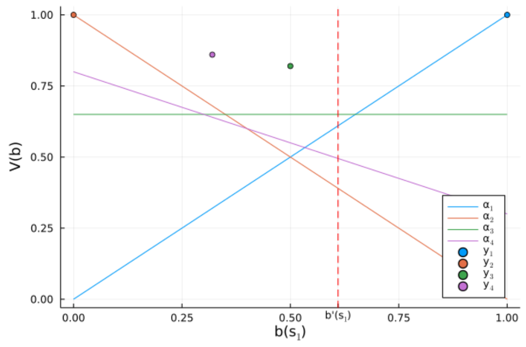

Figure 6.1. Value function when stage game is reached.

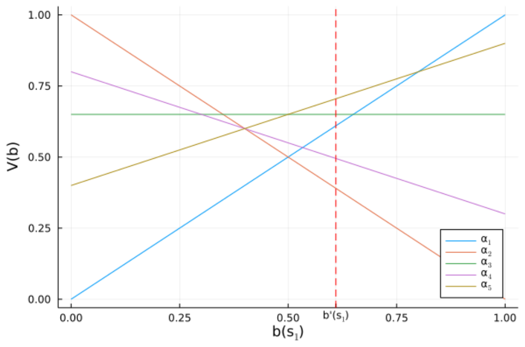

Figure 6.2. Lower bound after adding α-vector α_5.

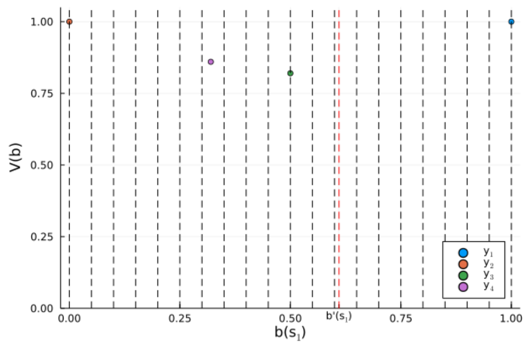

Figure 6.3. Discretized upper bound before solving the stage game.

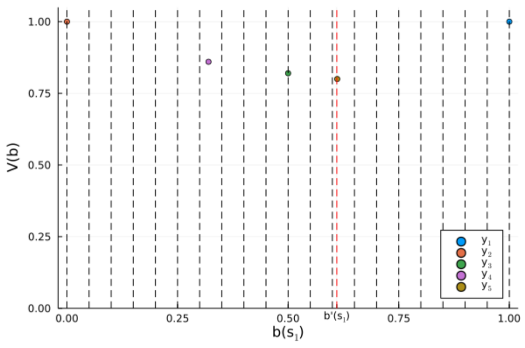

Figure 6.4. Discretized upper bound after adding new point y_5.

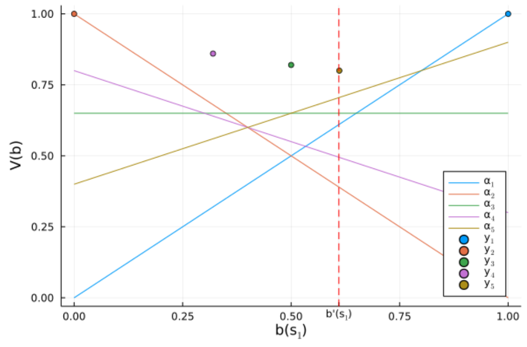

Figure 6.5. Updated value function after the stage game is successfully solved.

6.2 Removing Linear Programs

Linear programs (3.17a)-(3.17g) and (3.23a)-(3.23k) may be used to construct new α-vector and point even when the policies π_1 and π_2 are fixed. However the LPs are not avoided and therefore the other approaches cannot be more time nor space efficient then the regular HSVI for OS-POSGs. Constructing new α-vector and point without the need of the linear programming is the first step in making HSVI with iterative stage game solving faster. Point computation may altogether avoid LPs if the policy π_2 is fixed from some iterative approach and $\hat{V}(a_1, o)$ is computed without requiring the solution of LP (3.19a)-(3.19f). Computing lower convex hull in n-dimensional space is notoriously difficult task [32]. However, we do not necessarily need the complete lower convex hull, but it is sufficient to know a single $|S| + 1$ dimensional coordinate on this convex hull. Since the belief for which we try to compute the value is known from the belief update (3.14), we require only the single value to be computed. Using neural network with $|S|$ inputs and single output may be used to approximately find this upper bound value [7]. This neural network is trained on all points $(b, y) \in \Upsilon$, however some beliefs may be solved multiple times, with different values. This makes multiple inputs, having the different output in training. Reaching target loss with such configuration may not be possible. To avoid this, pruning is employed, where if the new belief point (b, y) is about to be added into Υ, we first find all points (b_i, y_i) for which the following conditions hold.

$$||b - b_i||_\infty < \varepsilon_{UB} \tag{6.16}$$

$$y_i > y \tag{6.17}$$

All points which satisfy both conditions are removed from the Υ. If there is at least one point for which (6.16) holds but (6.17) does not, then the new point (b, y) is not added into Υ. This neural network allows us to completely avoid LPs in the computation of the new upper bound point if we know the policy π_2. Still, the results suffer from the error made by the neural network, and it is necessary to periodically retrain the neural network when new points are created.
Constructing new α-vector with known policy π_1 requires the knowledge of $\hat{\alpha}^{a_1,o}(s')$ in (3.17b) for each (a_1, o) pair. It is not possible to compute such convex combination of α-vectors, which maximize (3.17a) for each action of the opponent without the use of the LP. To simplify this computation we decided to fix $\hat{\alpha}^{a_1,o}$ to be single α-vector which maximizes the value after belief update.

$$\hat{\alpha}^{a_1,o} = \operatorname*{argmax}_{\alpha \in \Gamma} \sum_{s \in S} \tau(b, a_1, \pi_2, o)(s) b'(s) \tag{6.18}$$

This α-vector $\hat{\alpha}^{a_1,o}$ is then used in (3.17b) instead of the LP variable, and such computation may be done algorithmically without the need for linear program. A newly constructed α-vector may result in a worse value $\sum_{s \in S} b(s)\alpha(s)$ than the one created by the LP, but it completely avoids any usage of linear programs.

6.3 Exploration

When solving the games by LP with such exploration, it is guaranteed that at least one stage game in any taken trajectory tightens the gap between the Lower and Upper bound. However, this is not ensured when using the approximate iterative algorithms to solve the stage games. The algorithm would always choose the same trajectory because of its deterministic nature without improving the value function. To suppress this effect, when using iterative algorithms for solving the stage games, the action observation pair is instead sampled based on the weighted excess gap (3.27). The higher the weighted excess gap, the higher the probability of (a_1, o)-pair to be sampled.

Chapter 7
Experiments

In previous chapters, we have presented new ways to solve fully observable two-player stochastic games and OS-POSGs using quantal response equilibrium and regret minimization. This chapter will study how using lower, and upper bound in HSVI affects the convergence compared to only a single value function. Additionally, we will examine the variance of the regular value iteration, value iteration with stochastic exploration, and HSVI with iterative stage game solving for fully observable games. We will also show how the action baselines affect both the variance and the values in the value function. Lastly, we will focus on how the setting of the λ parameter in quantal response equilibrium affects the variance and convergence of the HSVI for OS-POSGs.

All algorithms were implemented and then tested with Julia 1.6.2 [33]. The experiments were run on a desktop computer with Intel Core i5-4690K CPU and 16 GB DDR3 RAM. All of the plots are also created within Julia 1.6.2 with package Plots.jl [33].

7.1 Game Domains

We did all of our experiments on the variants of pursuit-evasion games (PEGs) [34]. These games are played by two players, where both of them move on the graph at once with multiple agents. One player is called the pursuer, and the second is the evader. The goal of the pursuer is to catch the evader by either entering the same graph vertex at the same time or moving through the same edge at once.

The basic version of PEGs is entirely deterministic, and the reward given to the pursuer is constant. We decided to use two different variants of this game. The first one removes the deterministic element from the game by introducing a non-zero probability that some action may fail. If the action fails, the agent does not move at all. The second different variant changes reward. In this version, the reward is dependent on where the agent was caught and the previous position of both agents. The reward is either 0.1 with 90% probability or 1 with 10% probability. These two variants were used to boost the variance. The stochastic transitions increase the variance in game solutions by adding multiple possible future states after each transition, so there are much more trajectories in the game tree. Using different values for reward causes both players to sometimes mix their strategies. It may not be beneficial for the pursuer to catch the evader as soon as possible because it could catch him later with a higher reward. The same goes for the evader because it may sometimes prefer to be caught earlier to deny possibly huge reward to the pursuer.

7.2 Fully Observable Stochastic Games

In a fully observable environment, we can easily examine the effect of the lower and upper bound from HSVI on convergence and variance because there are only finitely many

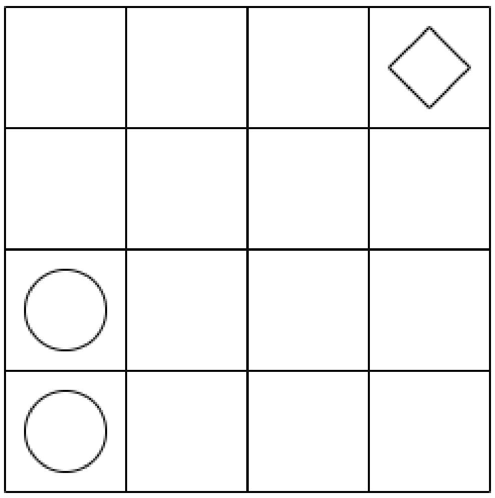

Figure 7.1. Example of Pursuit Evasion Game on a 4×4 grid with two pursuers and one evader.

states to investigate. We cannot investigate global variance because each algorithm is making a different amount of iterations based on the way it explores the state space. This would cause an unfair advantage for value iteration with stochastic exploration because it has shorter samples. This means that a lot of states remain unchanged for several iterations. Therefore, the variance would be close to zero, just because the states are often not visited.

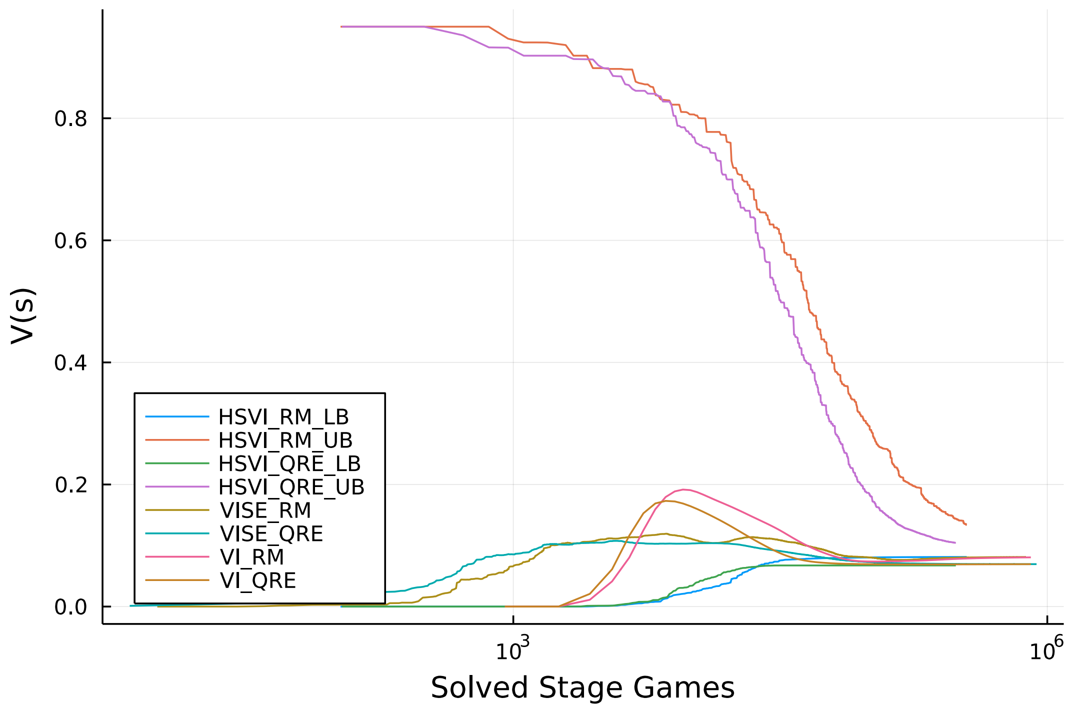

Figure 7.2. Expected value in the initial state in a 4×4 PEG with usual value iteration, value iteration with stochastic exploration and heuristic search value iteration.

Firstly we will focus on PEG with two pursuers and one evader on a 4×4 grid as shown in 7.1, which has randomized rewards but deterministic transitions. This game has 897 states, 577 pursuer actions, 49 evader actions, and 24001 transitions. The convergence of the expected value in the initial state is presented in Figure 7.2. The value iteration was run with 900 iterations, VISE with 75000 iterations, and HSVI with 6000 iterations. These numbers were selected so that the number of solved stage games is comparable between the algorithms.

Figure 7.2 shows that both value iteration and value iteration with stochastic exploration converge much faster than the HSVI upper bound. This is possibly caused by the terminal state not having initialized value to 0 in the upper bound, which is the true value of this state. It also may be caused by the exploration. In contrast, VISE has roughly sampled ten states per iteration, and the HSVI samples approximately 100 states. Therefore, the algorithm explores many states which are not easily reachable from the initial state and do not affect the expected value there. Usual value iteration could then be faster because it updates states close to the initial one more often than the HSVI. Notice that there are two different values to which algorithms converge. The first one is roughly 0.0614, and the second is 0.0788. All of the algorithms which use QRE to solve a stage game converge to the lower one. This is due to setting of $\lambda = 100$. Higher λ should approximate a Nash equilibrium more closely. We will further show the effect of this parameter in later sections.

Figure 7.3 shows the expected value in two additional states in the game chosen randomly. The expected value in these states is similar to the initial state, but the expected value changes only when the state is visited. Since these states are not visited in each iteration as the initial state, the expected value does not change that often as in 7.2.

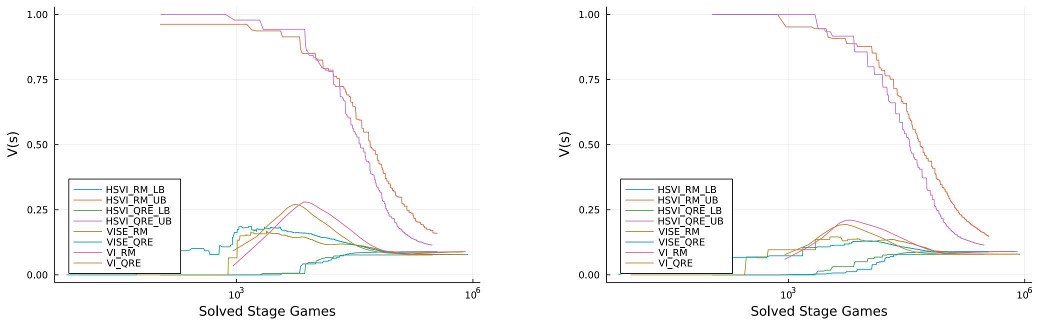

Figure 7.3. Expected value in two different states in a 4×4 PEG with usual value iteration, value iteration with stochastic exploration and heuristic search value iteration.

These figures show that the expected value for PEGs with randomized reward does not have significant variance. Therefore we will focus on policy in the same states to see if the variance is present there. The variance in the initial state, which corresponds to the Figure 7.2 is shown in Figure 7.4. It was computed with a moving window that computes the mean variance of each action in the last 100 visits of a given state. So the first sample is done after 100 visits, second after 101 visits etc. The regret minimization approaches seem to have a significantly higher variance for both players than the QRE approaches. When comparing the convergence speed, the QRE seems to converge slightly faster. This suggests that the higher variance in regret minimization causes it to converge slightly slower.

We will again show the variance in 2 more states in Figure 7.5. These figures show that regardless of the algorithm, the variance converges to 0. This is due to computing the average strategy by moving average, which causes each new increment to change the overall policy less. However, in one of the states, the policy variance of the pursuer in the HSVI lower bound with QRE was substantially greater than any other algorithm.

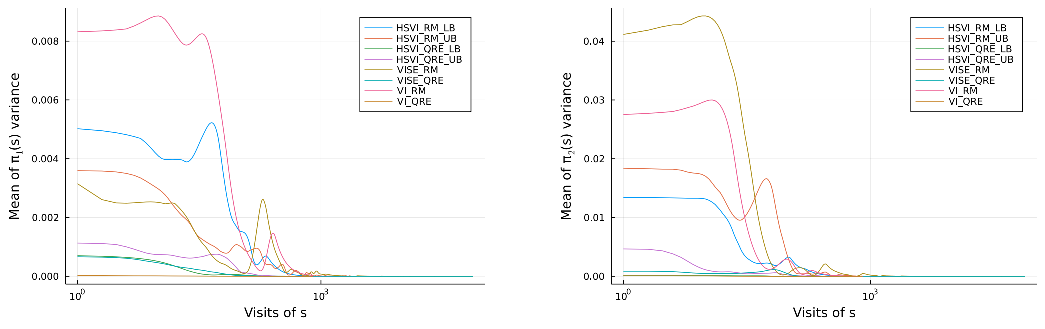

Figure 7.4. Policy variance across all actions every 100 visits of the initial state in a 4×4 PEG with usual value iteration, value iteration with stochastic exploration, and heuristic search value iteration. The left plot is a variance for pursuer actions and the right for evader actions. The x-axis is scaled logarithmically.

This was caused because one action had a higher expected value in the first couple of iterations, so it was prioritized. However, in the later iterations, it was found that this action would result in worse value than any other action. So the policy dramatically changes throughout couple more iterations to lower the probability of playing this action. This suggests that the randomness from the exploration causes the variance. It also shows that higher variance may be beneficial because this worse action should not be played if it results in a worse value. Hence, decreasing the probability of playing this action in policy is favorable.

We present the individual policies in the Appendix A.1 and A.2.

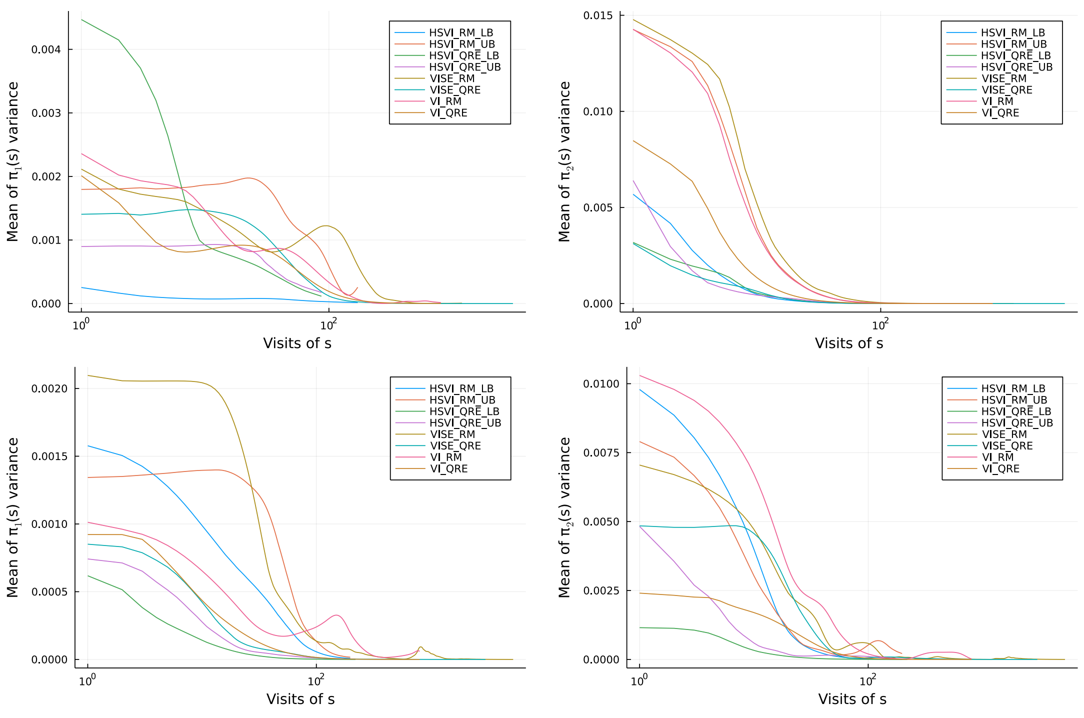

Figure 7.5. Policy variance across all actions every 100 visits in a $4{\times}4$ PEG with usual value iteration, value iteration with stochastic exploration, and heuristic search value iteration. Left plots are a variance for pursuer actions, and right plots are for evader actions. The x-axis is scaled logarithmically.

We have done the same experiment for the PEG on a 4×4 grid with constant reward but stochastic transitions. This game consisted of 1922 states, 1178 pursuer actions, 50 evader actions, and 363506 transitions. The probability that the action would fail was 10%. The value iteration was run with 500 iterations, VISE with 75000 iterations, and HSVI with 8000 iterations. The Expected value is shown in Figure 7.6. Out of the tested algorithms, the HSVI converges the slowest. VI and VISE both converge almost equally fast. Again the QRE does not converge to the same value as regret minimization. Still, the stochastic exploration value iteration converges the fastest. In comparison, the HSVI converges more slowly than the other algorithms.

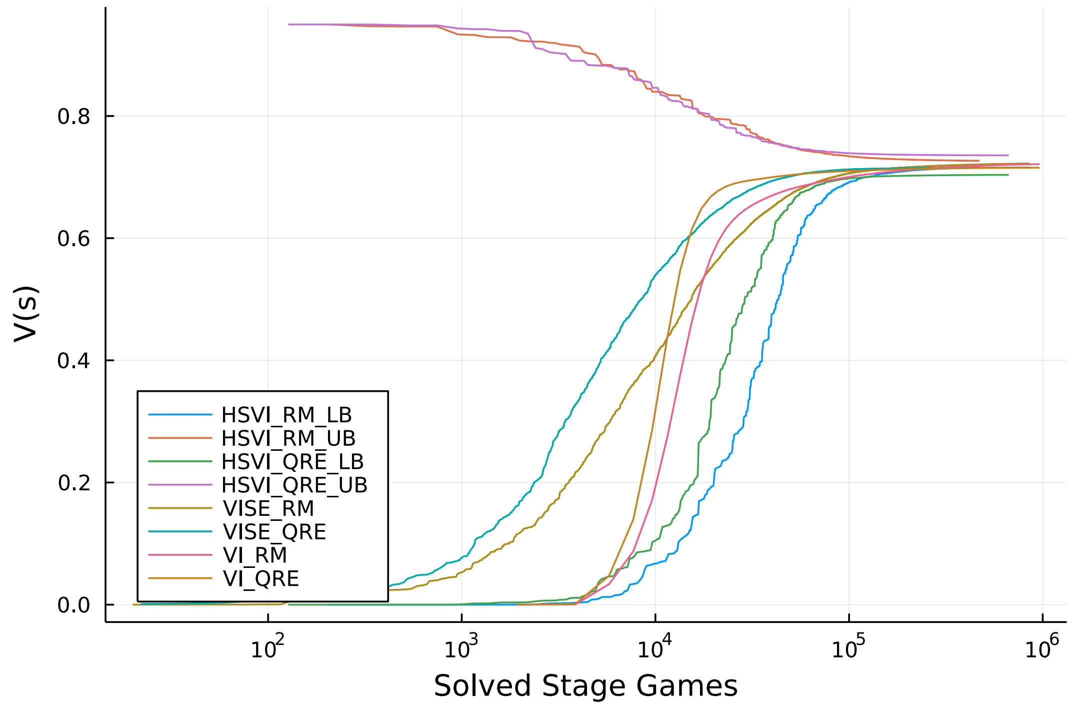

Figure 7.6. Expected value in the initial state in a 4×4 stochastic PEG with usual value iteration, value iterations with stochastic exploration and heuristic search value iteration.

We further show expected value in two randomly selected states in Figure 7.7. These results are again similar to the value in the initial state. This suggests that both variants of PEGs are best solved by the value iteration with stochastic exploration, slightly worse with the usual value iteration, and the worst algorithm for a fully observable case is HSVI. It is important to keep in mind that HSVI solves each stage game twice in each iteration, once in the forward pass and the second time in the backward pass.

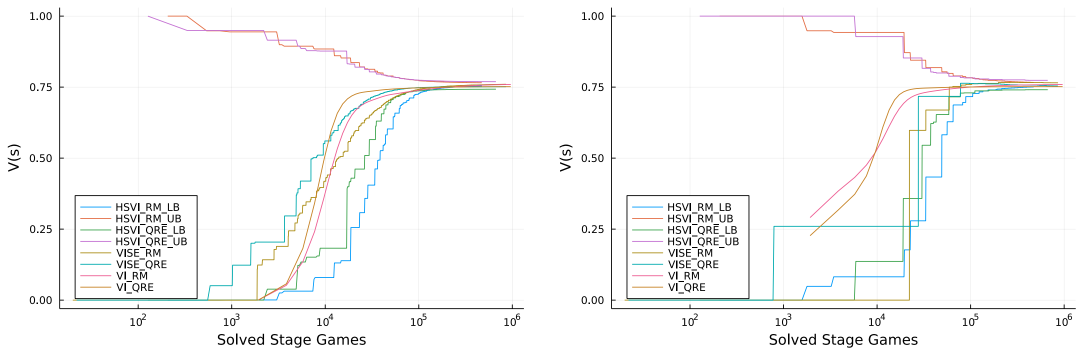

Figure 7.7. Expected value in two different states in a 4×4 PEG with usual value iteration, value iteration with stochastic exploration and heuristic search value iteration.

Policy variance in the initial and one other state is then shown in 7.8. As with the deterministic version, the variance when algorithm uses QRE is smaller than when the algorithm uses regret minimization. We again provide the individual policies in Appendix A.3 and A.4.

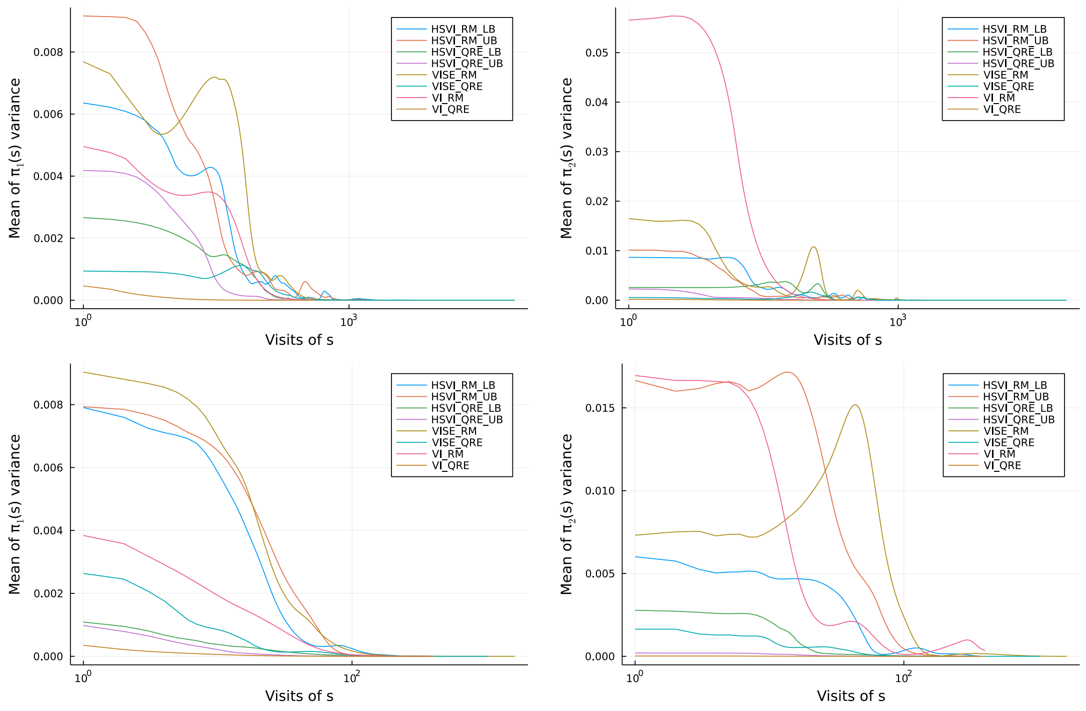

Figure 7.8. Policy variance across all actions every 100 visits in a 4×4 PEG with usual value iteration, value iteration with stochastic exploration, and heuristic search value iteration. Left plots are a variance for pursuer actions, and right plots are for evader actions. The x-axis is scaled logarithmically.

We ran the same experiment for two other PEGs. One was on a 5×4 grid with 1801 states, 63 evader actions, 962 pursuer actions, and 53321 transitions in the deterministic version. The stochastic version had 3802 states, 1955 pursuer actions, 64 evader actions, and 816706 transitions. The second was on a 5×5 grid, where the deterministic version had 3589 states, 1601 pursuer actions, 81 evader actions, and 117281 transitions. The stochastic version had 7502 states, 3242 pursuer actions, 82 evader actions, and 1812514 transitions. The results were mostly the same, so we leave them in Appendix A.5.

7.3 Regret Minimization with baselines

In the previous section, we have shown how the variance in policies affects the convergence of the algorithms. This section will focus only on VISE with regret minimization, and we will compare the results when the baselines are used and when they are not. We will again focus not only on the variance in policy but also on the expected value. We will use the same 4×4 PEGs which were presented in the previous section.

We have run 20 tests with baselines and 20 without them. We computed policy variance for each run and computed the confidence interval on these policies. The results are shown in 7.9. The left plot represents pursuers policy variance, and the right is evaders policy variance. Pursuer's policy variance seems roughly the same with and without baselines. However, the confidence intervals are tighter early on, but later they become wider. Evader's policy variance is always better without baselines. This suggests that our implementation of baselines can increase variance, and Figure 7.10 also shows that the convergence of expected value is also worse with baselines than without them.

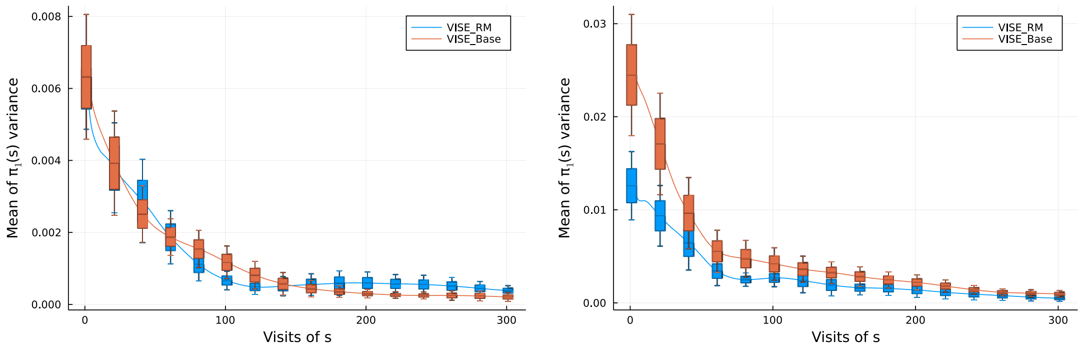

Figure 7.9. Variance of policies for each player in a 4×4 PEG. Left is a policy variance of pursuer and right is a policy variance of evader.

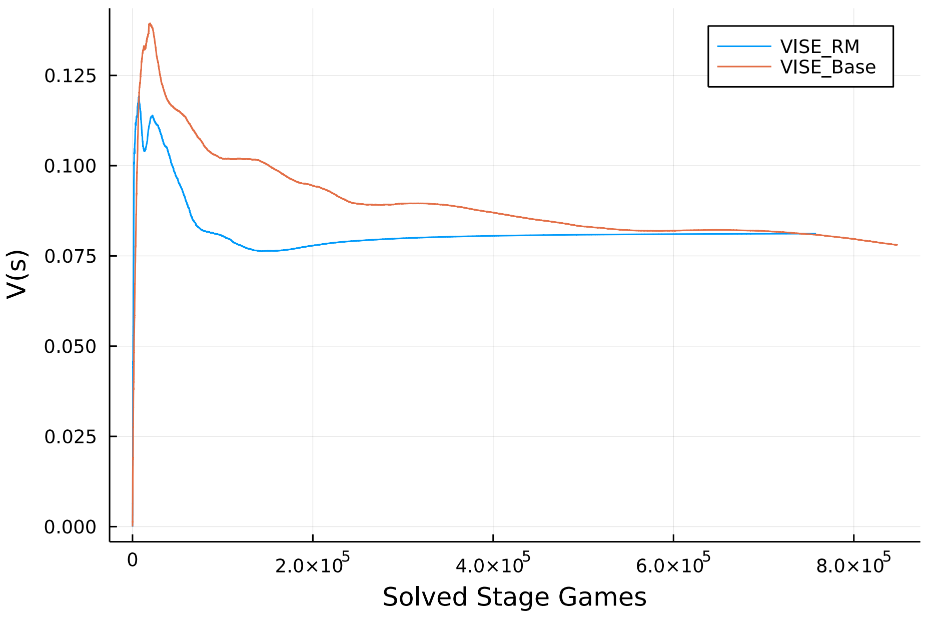

Figure 7.10. Expected value in the initial state in 4×4 PEG while solving with or without baselines.

We again investigated the effect of baselines in multiple other states. Each state seems to behave similarly to the initial one. The policies are often comparable, and the baseline policies converge slower or they differ, but the expected value is almost the same. We provide more policies in this and other states in the Appendix A.6.

Next, we will focus on the PEGs with stochastic transitions. The expected value in the initial state is shown in Figure 7.11. The resulting variances of policies for a PEG on a 4×4 grid are shown in Figure 7.12.

Again the policy for the evader has lower variance and better confidence intervals without baselines. On the other hand, the pursuer's policy variance is roughly the same regardless of the baselines. With the baselines, the confidence intervals are slightly

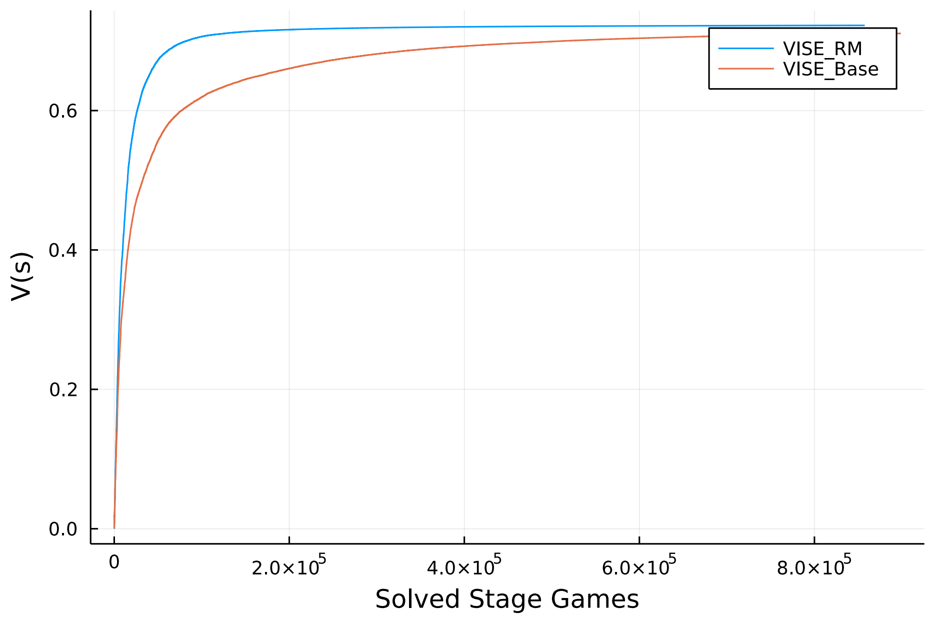

Figure 7.11. Expected value in the initial state in 4×4 PEG with stochastic transitions while solving with or without baselines.

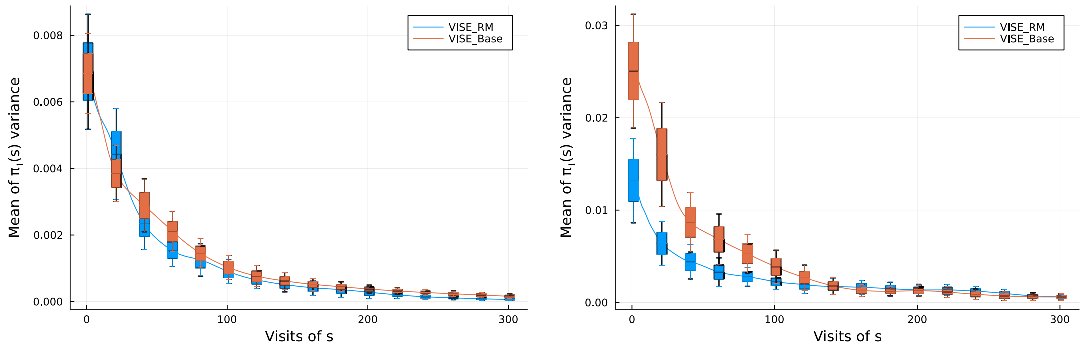

Figure 7.12. Variance of policies for each player in a 4×4 PEG with stochastic transitions. Left is a policy variance of pursuer and right is a policy variace of evader.

larger. Again, we will keep the additional information about other states and types of games in the Appendix A.6 and A.7.

7.4 Effect of λ parameter in QRE

The setting of the λ in logit QRE affects how closely the Nash equilibrium is approximated by QRE. If $\lambda = \infty$, then the QRE should give us Nash equilibrium in the limit. We will show how the three different settings of $\lambda = 100$, $\lambda = 500$, $\lambda = 5000$ changes the convergence and the variance in policies. We also show results when using regret minimization to make a clear comparison.

We start by testing QRE with different λ on the same 4×4 PEG with non-constant rewards. The expected value in initial state is shown in Figure 7.13. As expected the higher λ, the closer to the optimal value algorithms converge. The regret minimization seems to converge to roughly the same expected values as the QRE with high λ. However the QRE seems to converge faster. When using VI and VISE the regret

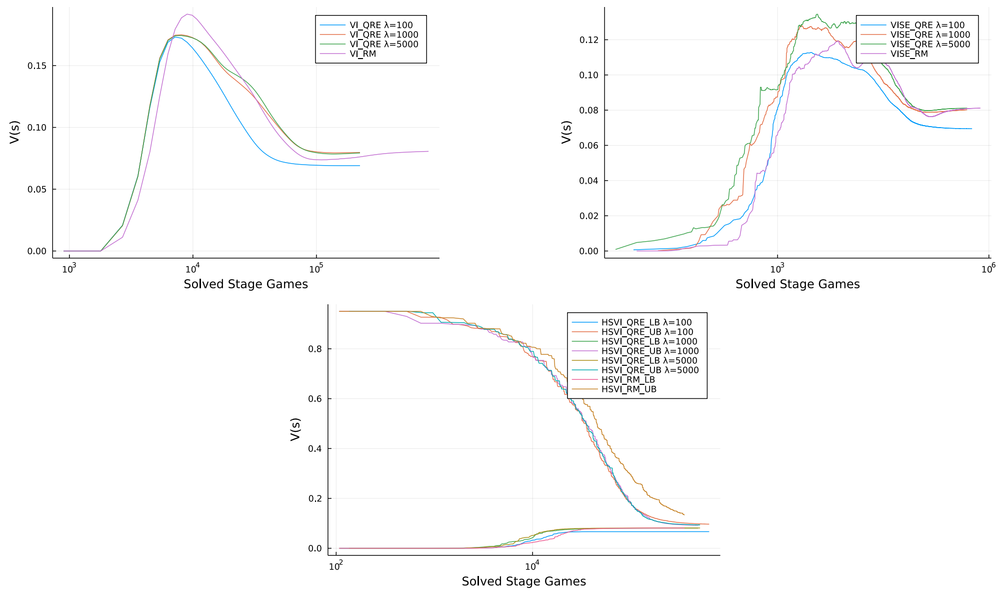

Figure 7.13. Expected value with different settings of QRE λ. First plot is while using value iteration, second is while using VISE and the last one is while using HSVI.

minimization overshoots the true expected value twice, but the QRE overshoots it only once.

We will further show the variance of policy based on the QRE λ. The results are shown in Figure 7.15. The variance is higher with higher λ. On the other hand, except for the policy of the pursuer with VISE and HSVI, regret minimization always has a higher variance. This suggests that higher variance caused by higher λ results in faster convergence, but the regret minimization converges slower even with higher variance. It seems that variance may be both beneficial and disadvantageous.

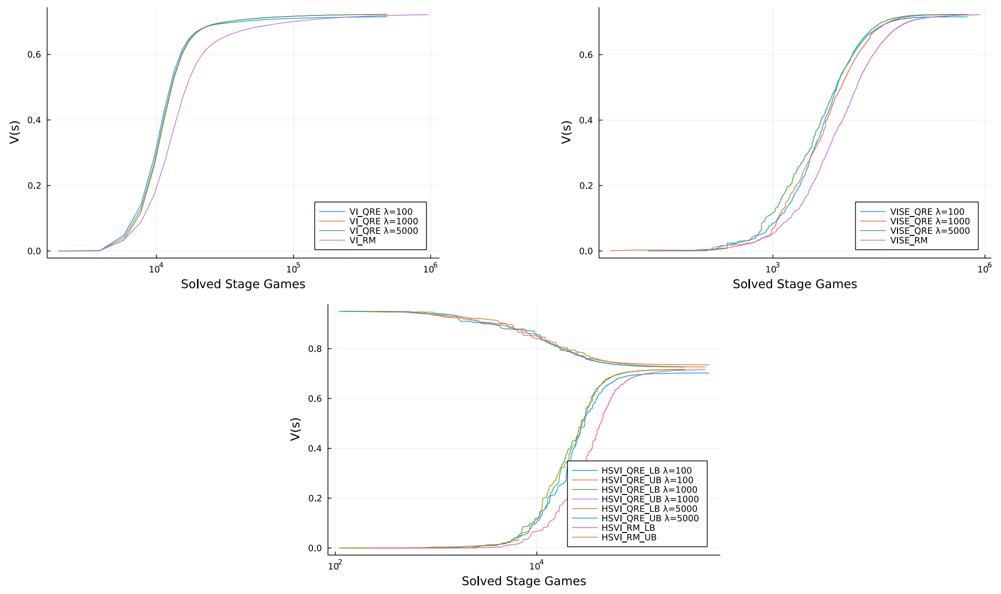

Figure 7.14. Expected value with different settings of QRE λ. First plot is while using value iteration, second is while using VISE and the last one is while using HSVI.

We ran the same experiment for the stochastic version on a 4×4 grid. The results are in Figures 7.14. Unlike the version with multiple rewards, the stochastic version does not overshoot the value, to which the algorithm converges. When setting the $\lambda = 100$, the algorithms almost converge as fast as with the higher λ, even when the variance is much smaller.

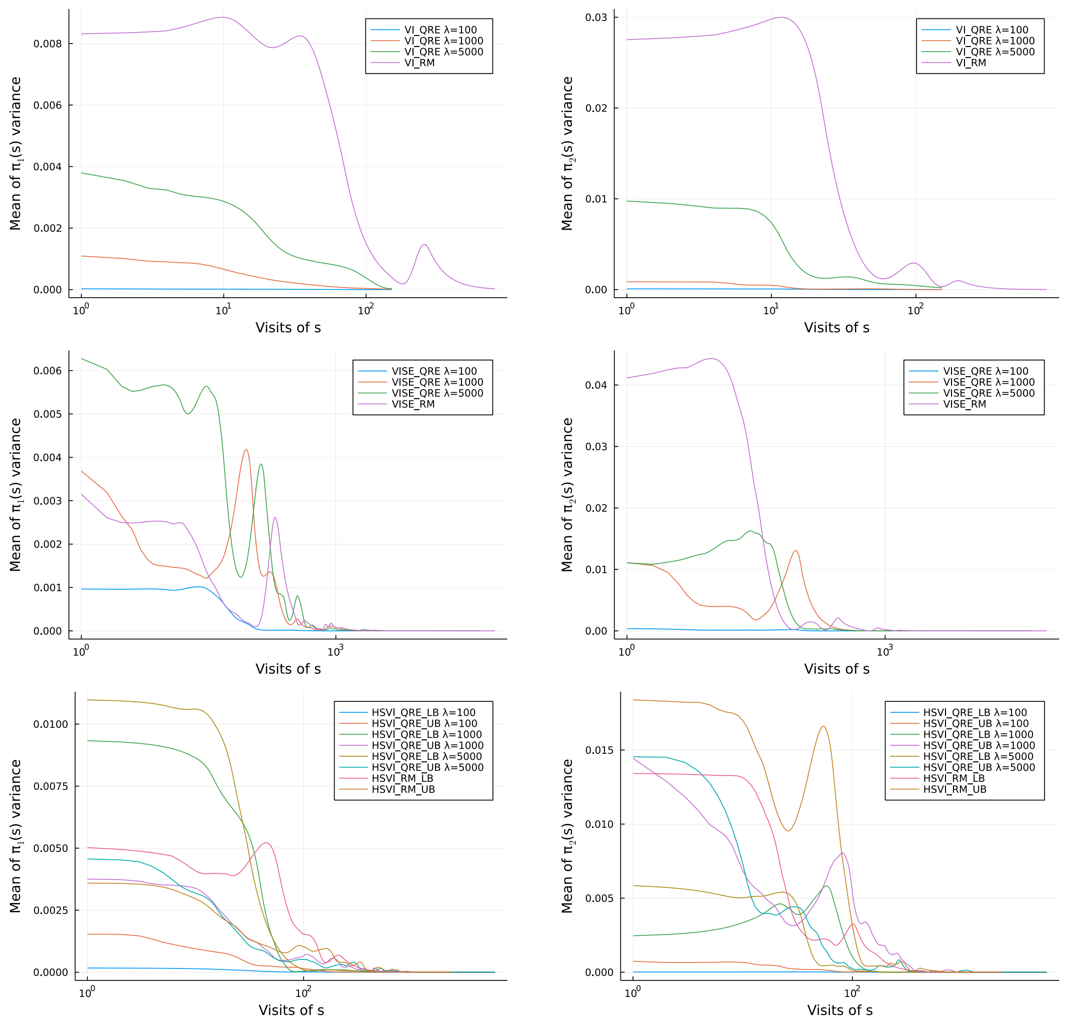

Figure 7.15. Policy variance with different settings of QRE λ in the PEG with non-constant rewards. First plot is while using value iteration, second is while using VISE and the last one is while using HSVI.

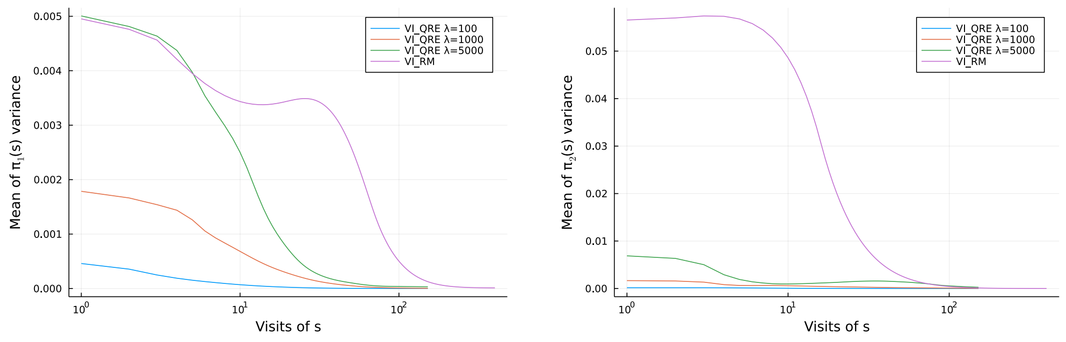

Figure 7.16. Policy variance with different settings of QRE λ in the PEG with stochastic transitions. First plot is while using value iteration, second is while using VISE and the last one is while using HSVI.

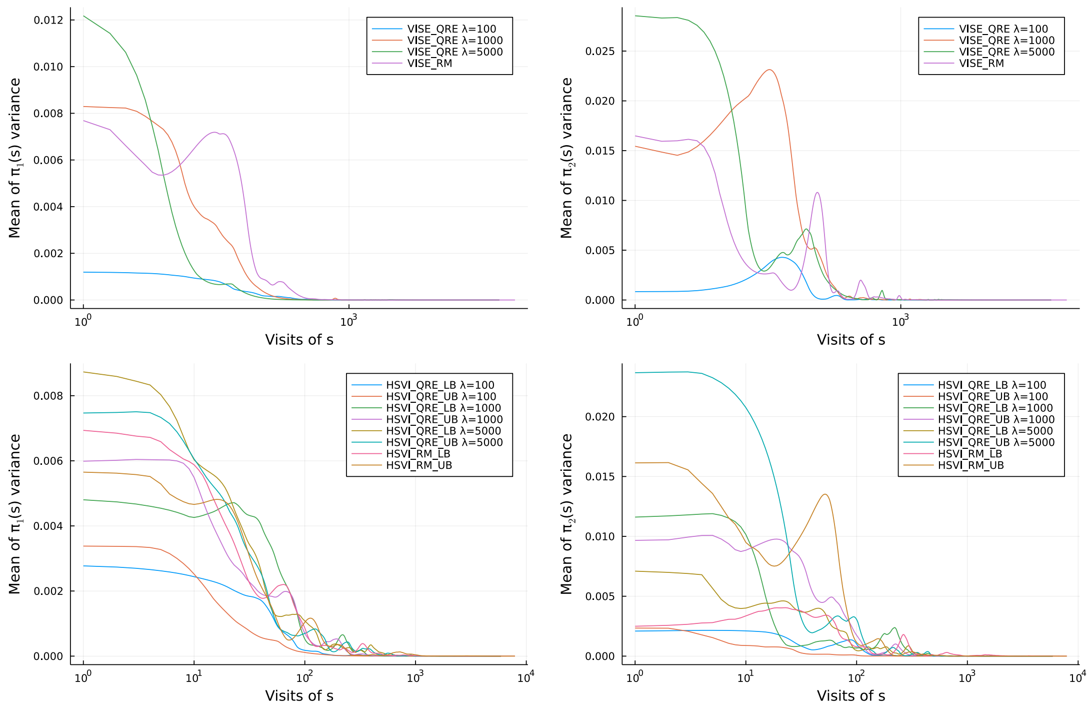

Figure 7.17. Policy variance with different settings of QRE λ in the PEG with stochastic transitions. First plot is while using value iteration, second is while using VISE and the last one is while using HSVI.

7.5 One-Sided Partially Observable Stochastic Games

Previous sections dealt with the convergence in SGs. This section is going to focus on convergence within the OS-POSGs. Unlike the SGs, the usual value iteration cannot be used for these games. Therefore, we will use only the HSVI and compare how the regret minimization compares with QRE. We will also set different values to the QRE parameter λ to ensure that findings from SGs are also applicable for OS-POSGs.

OS-POSGs are much more complicated games than the usual SGs, so we compared algorithms on just a 3×3 grid, and we used the deterministic rewards. The pursuer has only partial information in this version, while the evader knows everything about the game. Such a game has 144 states, 145 pursuer actions, 13 evader actions, 3 observations, and 2672 transitions. We ran each of the experiments for two hours. In Figure 7.18, we see that even after three hours, all algorithms closed the gap to roughly a width of 0.18. The regret minimization converges slightly slower than the QRE in both lower and upper bounds. Upper bound convergence for all algorithms is much worse than lower bound convergence. This may be caused because upper bound presolve is a much more sophisticated method and approximates the actual value better than the lower bound. It could also be caused by the upper bound value being computed from the perfect information policy player. This may suggest that computing a better policy for a perfect information player is for the iterative approach much more difficult. Figure 7.18 shows that regardless of the λ, the QRE converges to almost the same expected value, but when $\lambda = 100$, the later iterations do not improve the lower bound as with the higher λ. This suggests that high λ in QRE with OS-POSGs does not speed up the convergence. However, it should still converge closer to the optimal policy with enough time.

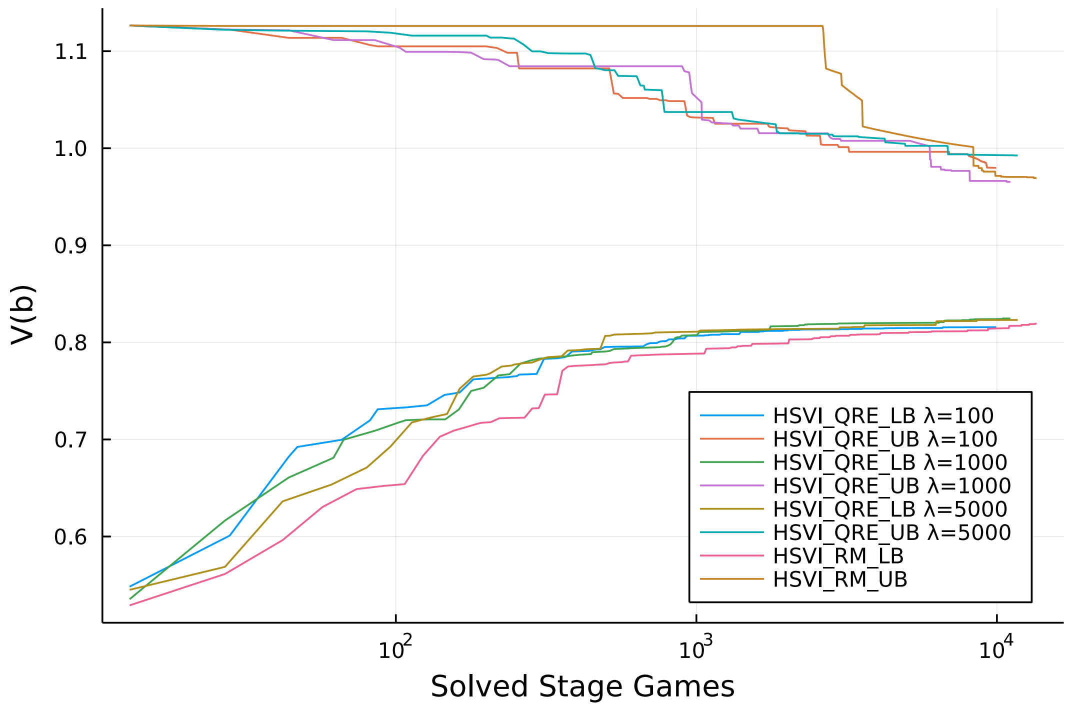

Figure 7.18. Expected value in the initial belief in a 3×3 partially observable PEG.

We also present the effect of parameter λ on variance in policies. The results are shown in Figure 7.19. In the partially observable setting, higher λ does not suggest higher variance like in the fully observable setting. On the other hand, when $\lambda = 1000$, the upper bound variance increases in the first 100 iterations until it decreases. Yet the expected value converges fastest.

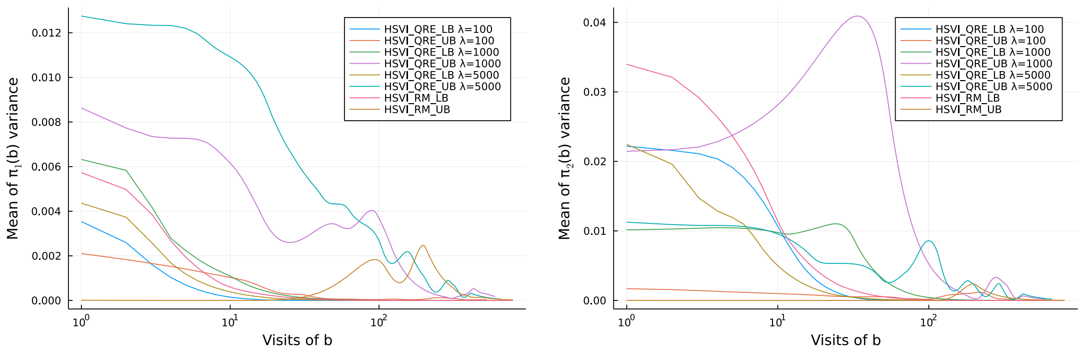

Figure 7.19. Policy variance across all actions every 100 visits of the initial belief in a 3×3 partially observable PEG. The left plot is a variance for pursuer actions and the right for evader actions. The x-axis is scaled logarithmically.

We again tried to solve the same game with stochastic transitions. The stochastic version also has 144 states, 145 pursuer actions, 22 evader actions, and 3 observations, but it has 9980 transitions. Figure 7.20 shows the expected value in the initial belief after three hours of runtime. The upper bound does not change the value throughout the whole run for $\lambda = 5000$ and for regret minimization. We did not manage to find the core of this issue. In the stochastic version, the lower λ resulted in better convergence, but not significantly. This further suggests that the convergence is more dependent on exploration than the parameter λ.

Policy variance is shown in Figure 7.21. Both players have high variance with regret minimization. High variance is also caused by the QRE with $\lambda = 1000$. The pursuer has high variance in the lower bound policy with this setting, while the evader has high variance in the upper bound. The difference within the expected value between regret minimization and QRE with $\lambda = 1000$ is quite significant even when the variance is similar. Both QRE with $\lambda = 100$ and $\lambda = 5000$ have small variance, yet there is a difference between their upper bound convergence. This suggests that small variance does not imply faster convergence.

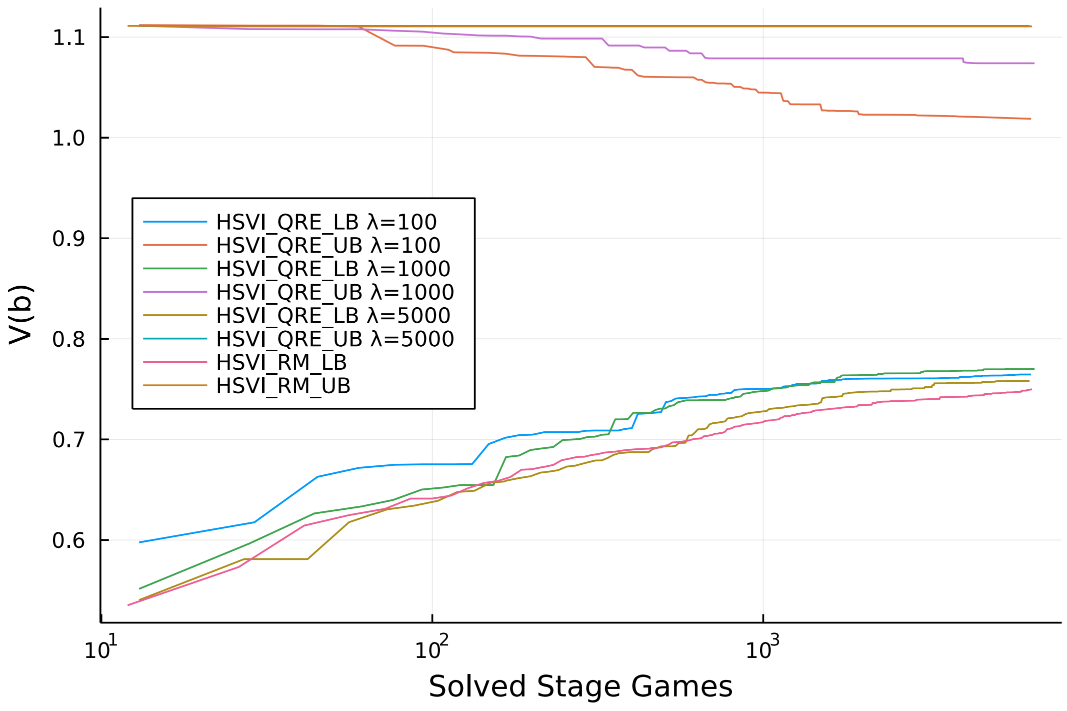

Figure 7.20. Expected value in the initial belief in a 3 × 3 partially observable PEG with stochastic transitions.

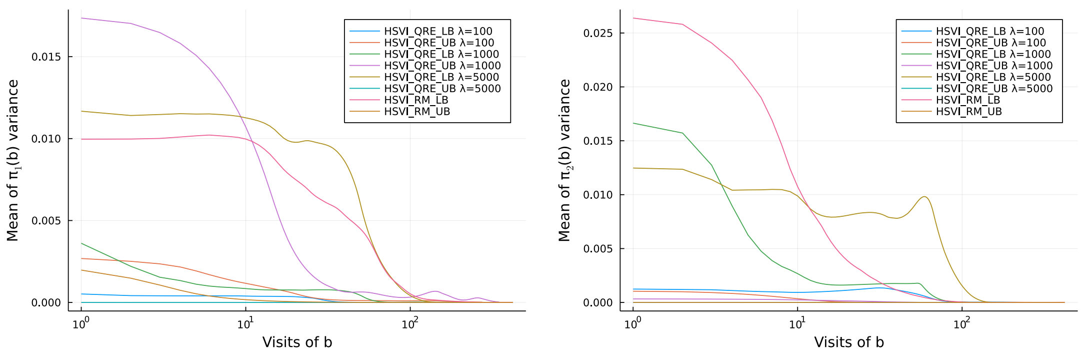

Figure 7.21. Policy variance across all actions every 100 visits of the initial belief in a 3×3 partially observable PEG with stochastic transitions. The left plot is a variance for pursuer actions and the right for evader actions. The x-axis is scaled logarithmically.

Chapter 8
Conclusion

In this work, we have decided to study variance in stochastic games and One-Sided Partially Observable Stochastic Games (OS-POSGs). We wanted to see if we could effectively use any of the variance reduction methods from reinforcement learning for these games. Solving and monitoring the OS-POSGs is tricky because of the continuous belief space. Our goal was to show that either the variance reduction techniques are applicable in HSVI for OS-POSGs or that the variance present in these games is necessary to converge to the optimal strategy effectively.

We did not try to employ variance reduction for the linear programming solutions of the stage games because the result of the linear program is always the Nash Equilibrium of the current stage game. Therefore, it improves the value function as much as possible when solving the stage game. Different exploration heuristics may speed up the overall convergence. Still, the goal of this work was to retain the same exploration because variance reduction in reinforcement learning speeds up the convergence on the same samples.

Different approaches to solving the stage game without linear programming cannot solve each stage game perfectly. Therefore the overall convergence is slowed down. In this work, we refined an already developed approach for solving the stage games via quantal response equilibrium [7]. Additionally, we developed a regret minimization technique to approximate a solution in each stage game. Since these algorithms approximate a Nash equilibrium, we do not get the optimal solution each time, so the variance reduction could improve the overall convergence while retaining the same samples.

Since using the value function itself is a variance reduction technique, we wanted to show the effect of two value functions serving as lower and upper bounds in HSVI on the variance and algorithm convergence. However, HSVI is the only algorithm currently developed for the OS-POSGs, so we decided to show this effect in a fully observable setting. We used regular value iteration and the version of HSVI for fully observable games, but the common value iteration deterministically updates all game states. We have developed a new version of value iteration called value iteration with stochastic exploration (VISE), which does not update each state, but samples the path from the initial state to the terminal and then updates only the states along this path. This version is more closely related to the HSVI than the regular value Iteration.

We empirically show that value iteration with stochastic exploration converges faster to the optimal value in a fully observable setting than the HSVI. This is primarily due to HSVI exploration, which depends on the upper bound policy, and the trajectories in HSVI are much longer than those from VISE. It also solves each stage game twice, once in the forward pass and the second time in the backward pass. However, value iteration does not provide information about how closely the solution approximates the optimal value. We could not show whether HSVI or VISE has a higher variance on average. Multiple runs all resulted in different variances for these algorithms. This suggests that the policy variance mainly depends on the exploration, which is non-deterministic.

We managed to show that updating policies with regret minimization has a higher variance than updating with QRE regardless of the algorithm. The QRE approach also converges faster than regret minimization. However, trying multiple settings of λ shows that even when the variance is increased, the convergence is not significantly affected. Regret Minimization is often used for finding optimal strategies in extensive-form games [35][31], so it is surprising that the QRE works better with value iteration methods in SGs.

Using baselines similar to those in Monte Carlo Counterfactual Regret Minimization [30] does not reduce variance in VISE, nor does it reduce the effect of stochastic exploration on the expected value. The VISE with baselines converges slower than without them. Using baselines altogether in the VISE does not improve anything. This suggests that the value function itself is a better technique for estimating the expected value than baselines. Baselines were developed for MCCFR, but the HSVI compares the policy against the best response, which is more similar to CFR-BR. To the best of our knowledge, there are no variance reduction techniques employed for CFR-BR. All of the CFR algorithms are mainly used to solve Poker's variants, whereas we use the HSVI for vastly different domains in the form of pursuit-evasion games. Poker is a game that has much higher stochasticity than PEGs. On the other hand, PEGs can be infinitely long, and there are much more actions applicable in each state.

We have also shown that the higher the value of the parameter λ in a fully observable setting, the closer to the optimal value the algorithms do converge. However, higher λ makes the algorithm less numerical stable because the λ is used as an exponent. When using smaller λ, all of the algorithms can converge to a fixed point similarly fast. Therefore using smaller λ is better if it is not necessary to approximate the solution well because reducing the effect of numerical instabilities lets us use faster data types. It also makes sense to increase the λ throughout the run in order to achieve more precise results in the end.

In the OS-POSGs, the setting of λ does not improve the speed of convergence. It is to be expected that, as with the SGs, the higher λ would allow the algorithm to get closer to the optimal value, but it has not been shown in our tests. Solving the upper bound in the OS-POSGs seems to be a much harder task than the lower bound, which is especially noticeable in the stochastic version of PEG. Regret minimization in OS-POSGs converges significantly slower than the quantal response equilibrium computation, but the regret minimization does not have any numerical instabilities like the QRE. Also, it works without specifying any additional parameters.

Even when regret minimization in OS-POSGs does converge slower and has a higher variance, we could not show that lower variance speeds up the convergence. When comparing the QRE, we have a larger variance for some settings, yet the algorithm converges faster. The higher variance at some stage is an advantage since it reacts to the changes in value function quicker. Nevertheless, sometimes these rapid changes favor the action, which may later be proved to be worse. Overall the higher variance could sometimes be helpful.

8.1 Future work

Our work found that the variance reduction techniques for HSVI for OS-POSGs with the usage of linear programming cannot improve the convergence. Even when the other approximate approaches cause a significant variance, the algorithms seem to converge the same or even better. This suggests that further work in variance reduction within these games is pointless.

The algorithm could further be improved by a different method of solving the stage games and different exploration heuristics. This could introduce a variance that may then be reduced by one of the usual variance reduction techniques.

In the presented version, the average policy is computed by moving average, which gives each policy increment the same weight. Still, newer policies may be valued more than the old ones. The different methods of computing average policy, like increasing weight linearly, may improve the overall convergence.

We showed that both SGs and OS-POSGs converge slightly better with QRE than with regret minimization. The research in different games, like extensive-form games, which often use regret minimization, could also focus on QRE to evaluate if it also works better in different settings.

Appendix A
Additional plots

A.1 Policies in the initial state in deterministic PEG

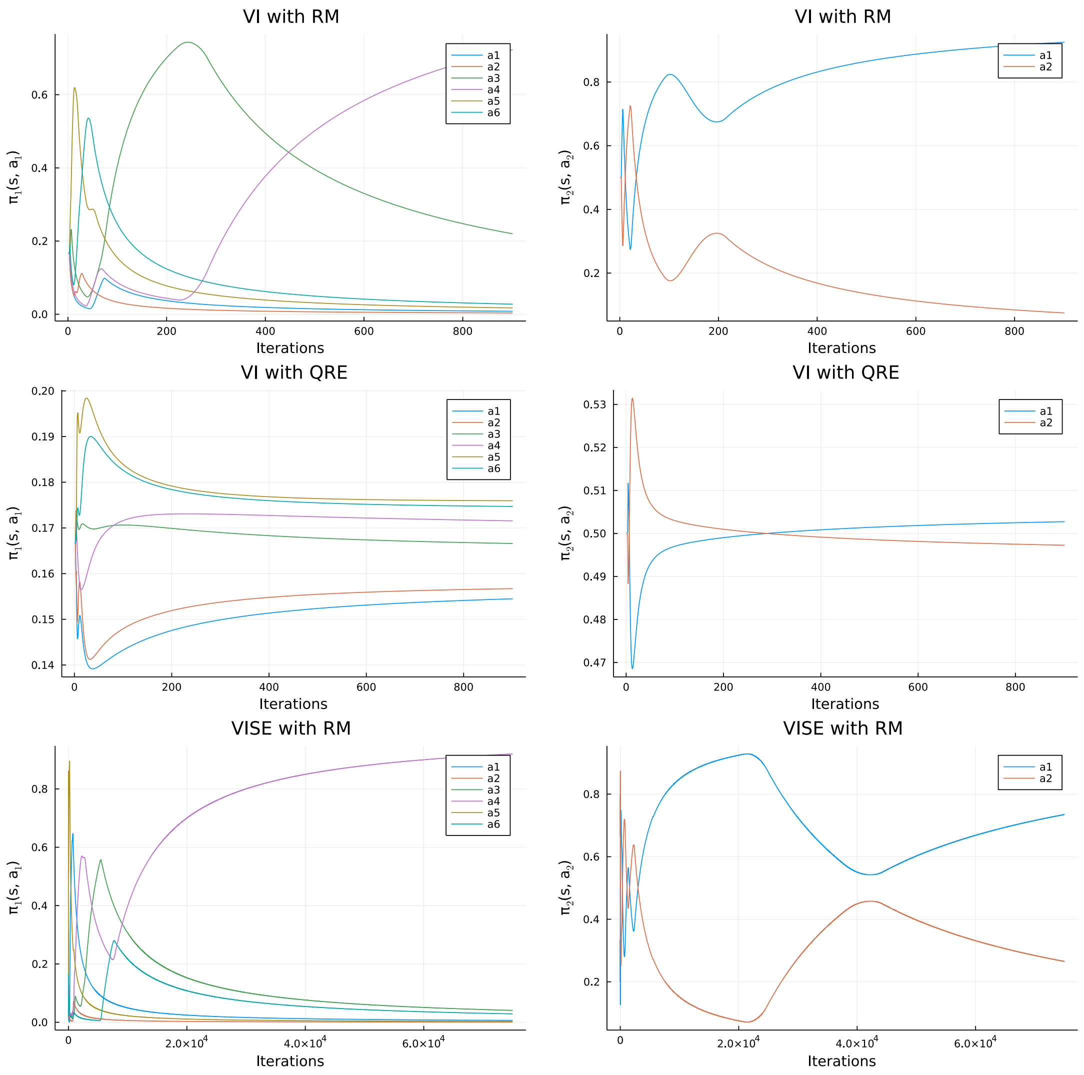

Figure A.1. Policies for each player in a 4×4 PEG in the initial state. Left is a policy of pursuer, and right is a policy of evader.

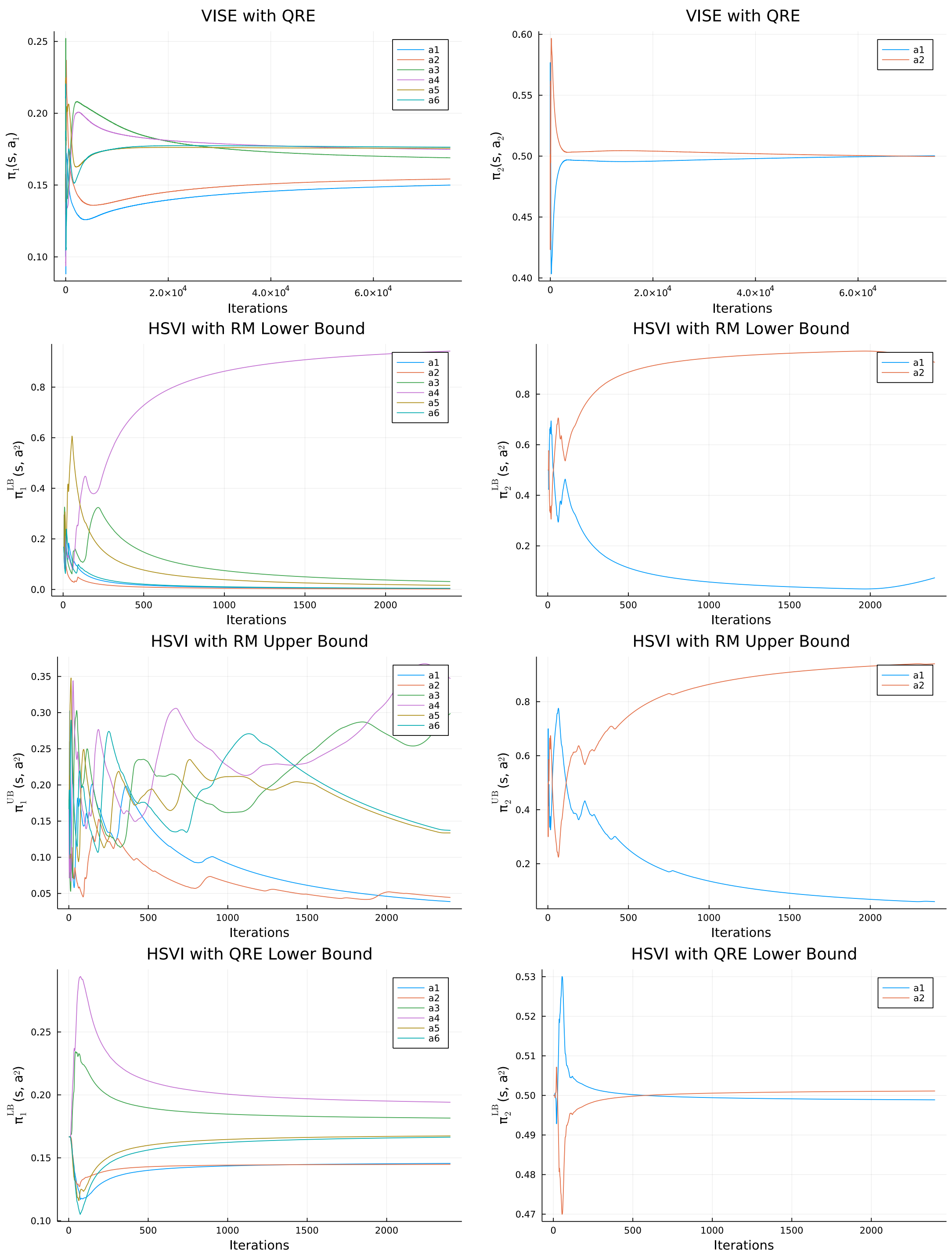

Figure A.2. Policies for each player in a 4×4 PEG in the initial state. Left is a policy of pursuer, and right is a policy of evader.

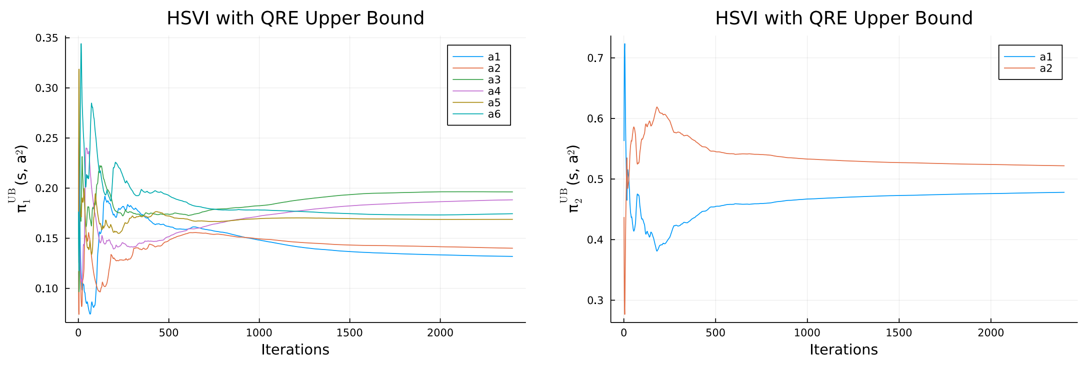

Figure A.3. Policies for each player in a 4×4 PEG in the initial state. Left is a policy of pursuer, and right is a policy of evader.

A.2 Policies in two non-initial states in deterministic PEG

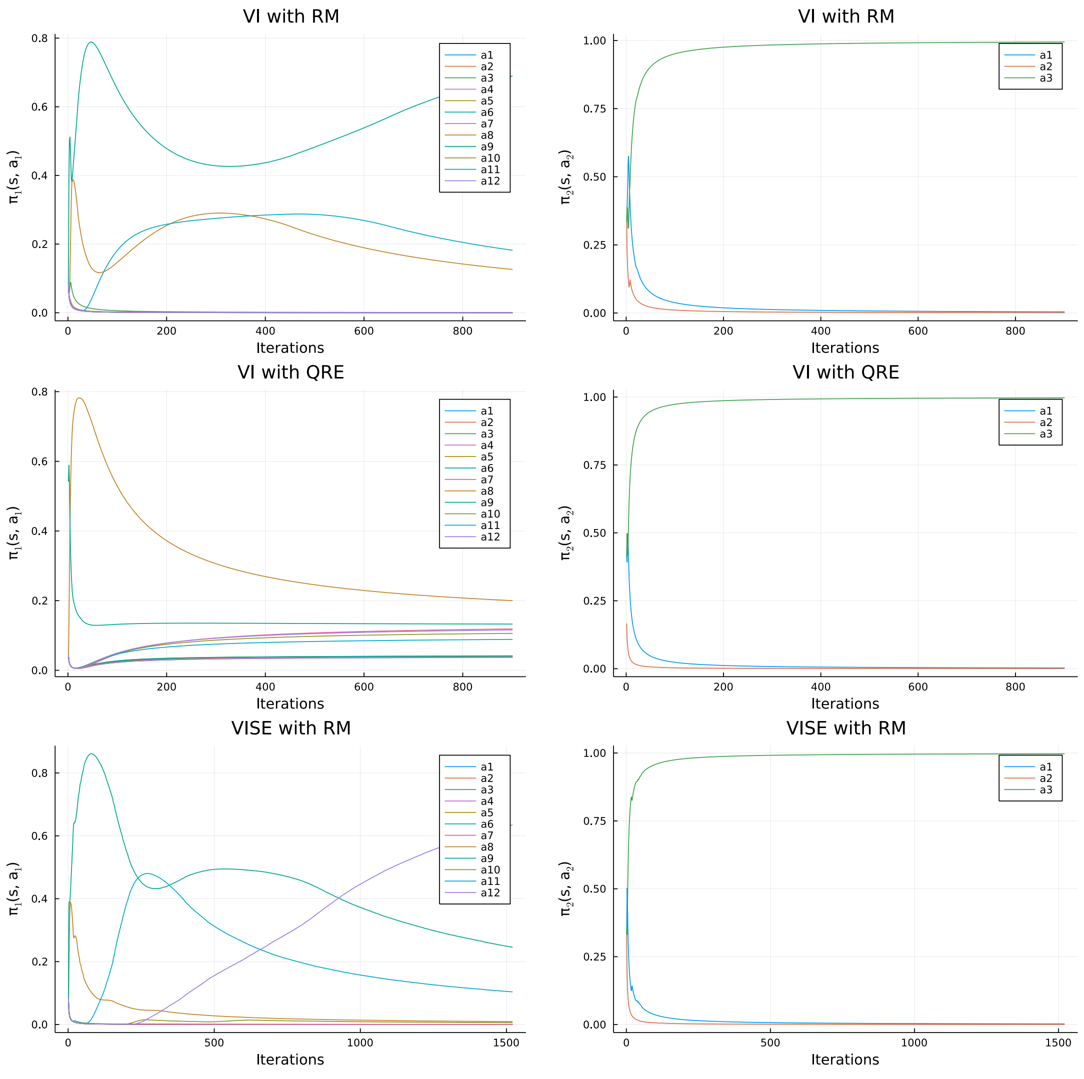

Figure A.4. Policies for each player in a 4×4 PEG in the non-initial state. Left is a policy of pursuer, and right is a policy of evader.

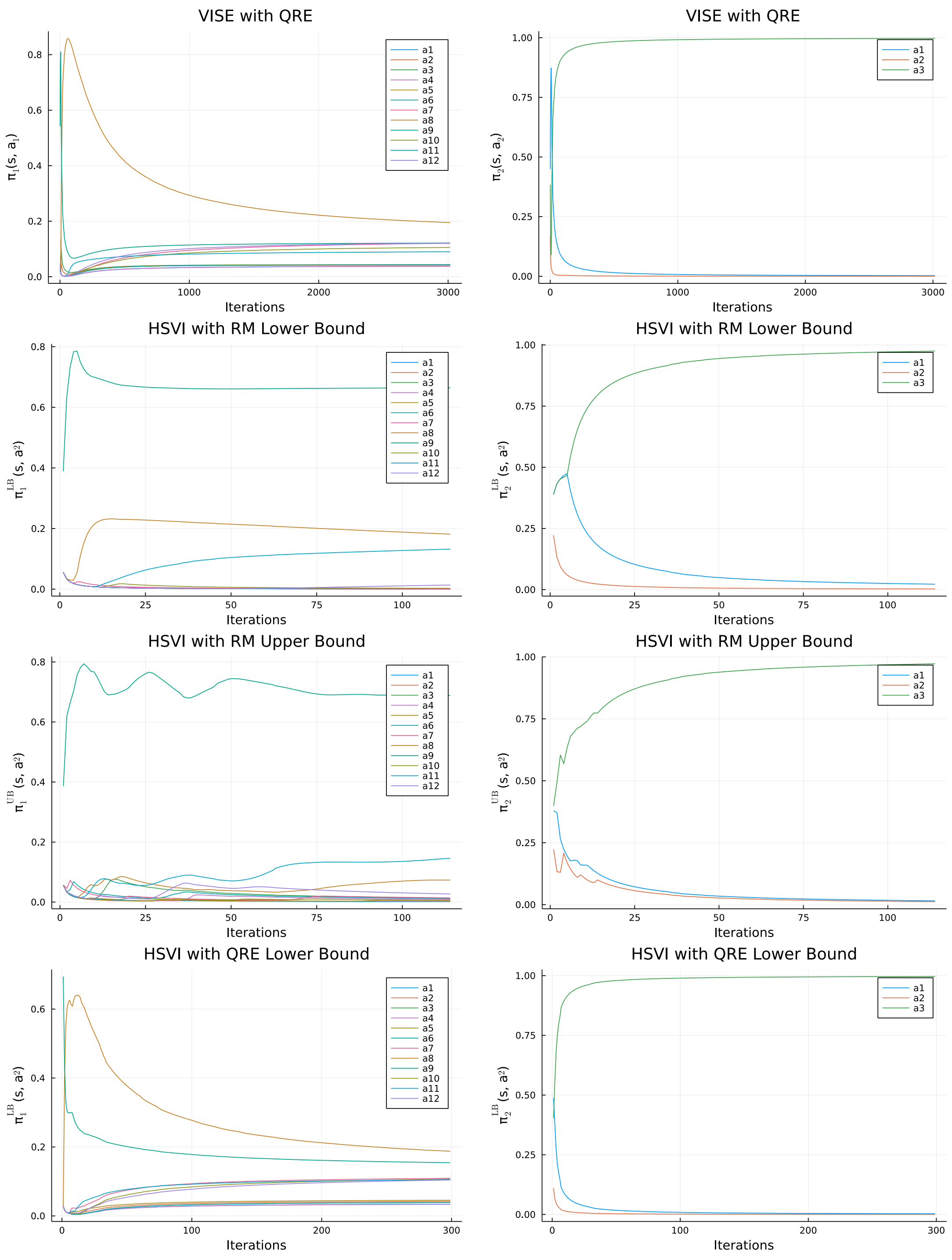

Figure A.5. Policies for each player in a 4×4 PEG in the non-initial state. Left is a policy of pursuer, and right is a policy of evader.

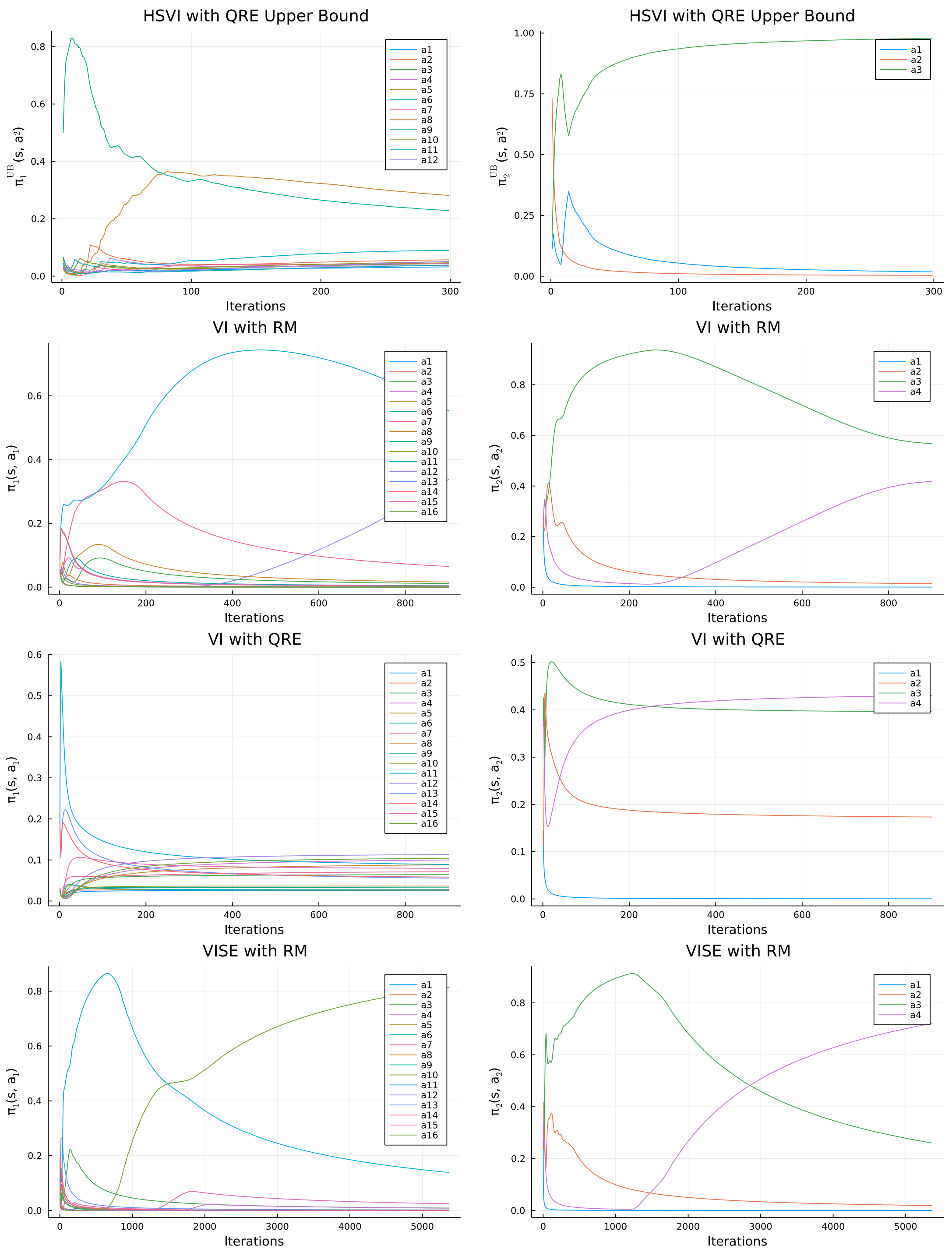

Figure A.6. Policies for each player in a 4×4 PEG in the non-initial state. Left is a policy of pursuer, and right is a policy of evader.

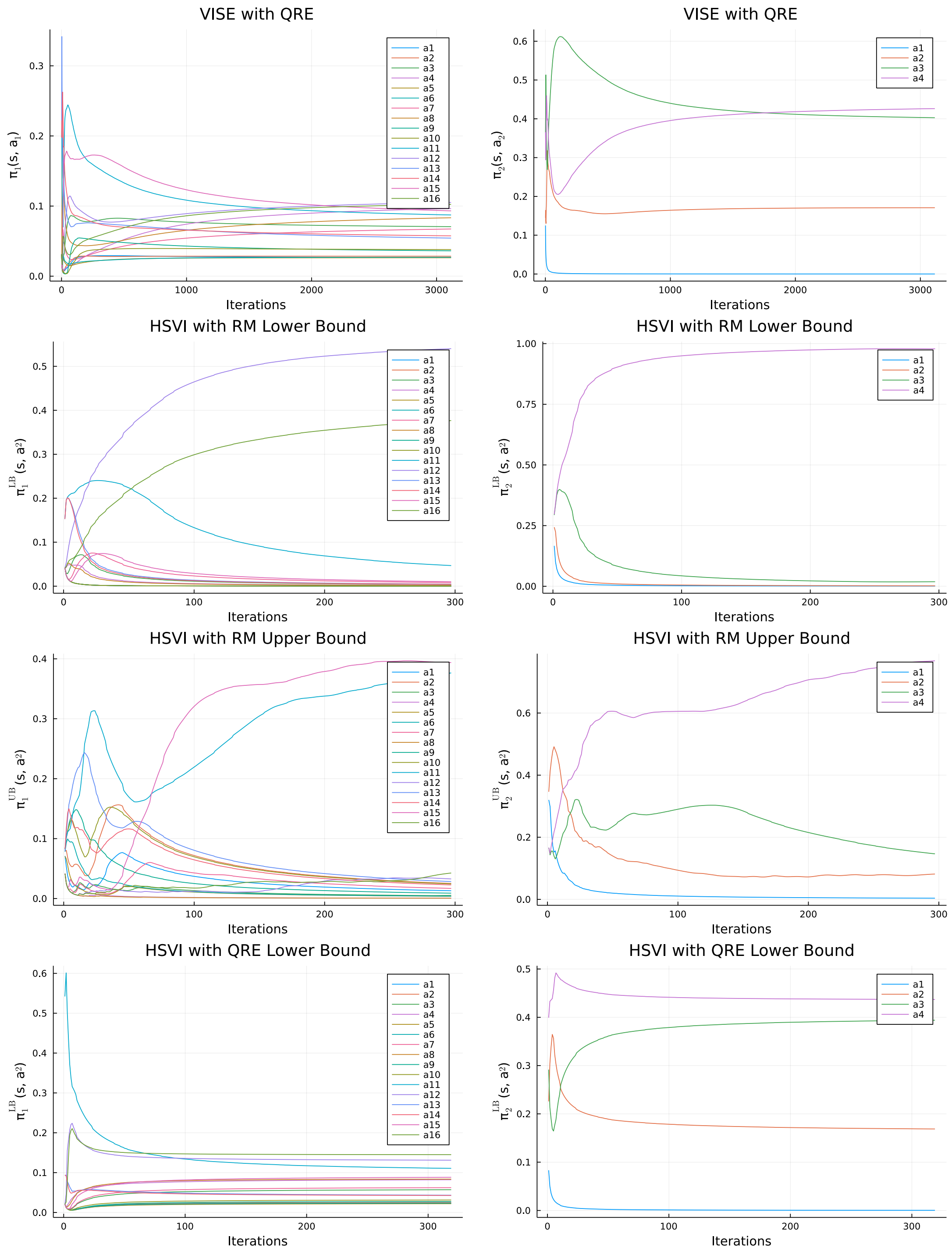

Figure A.7. Policies for each player in a 4×4 PEG in the non-initial state. Left is a policy of pursuer, and right is a policy of evader.

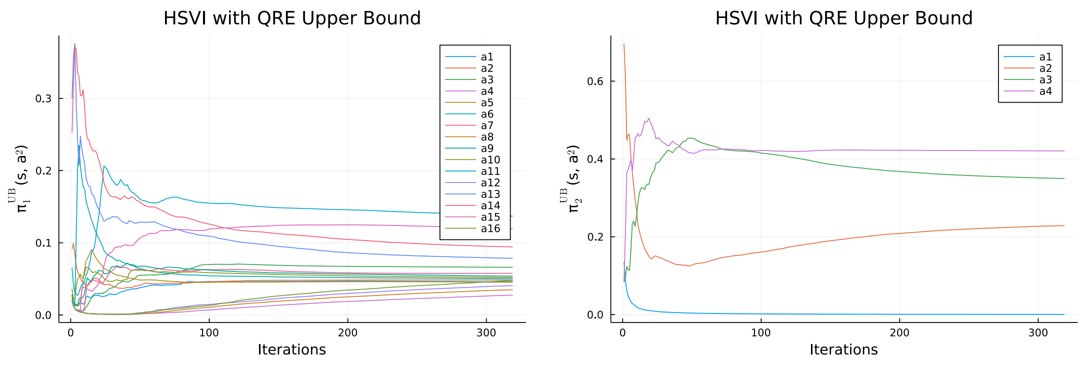

Figure A.8. Policies for each player in a 4×4 PEG in the non-initial state. Left is a policy of pursuer, and right is a policy of evader.

A.3 Policies in the initial state in stochastic PEG

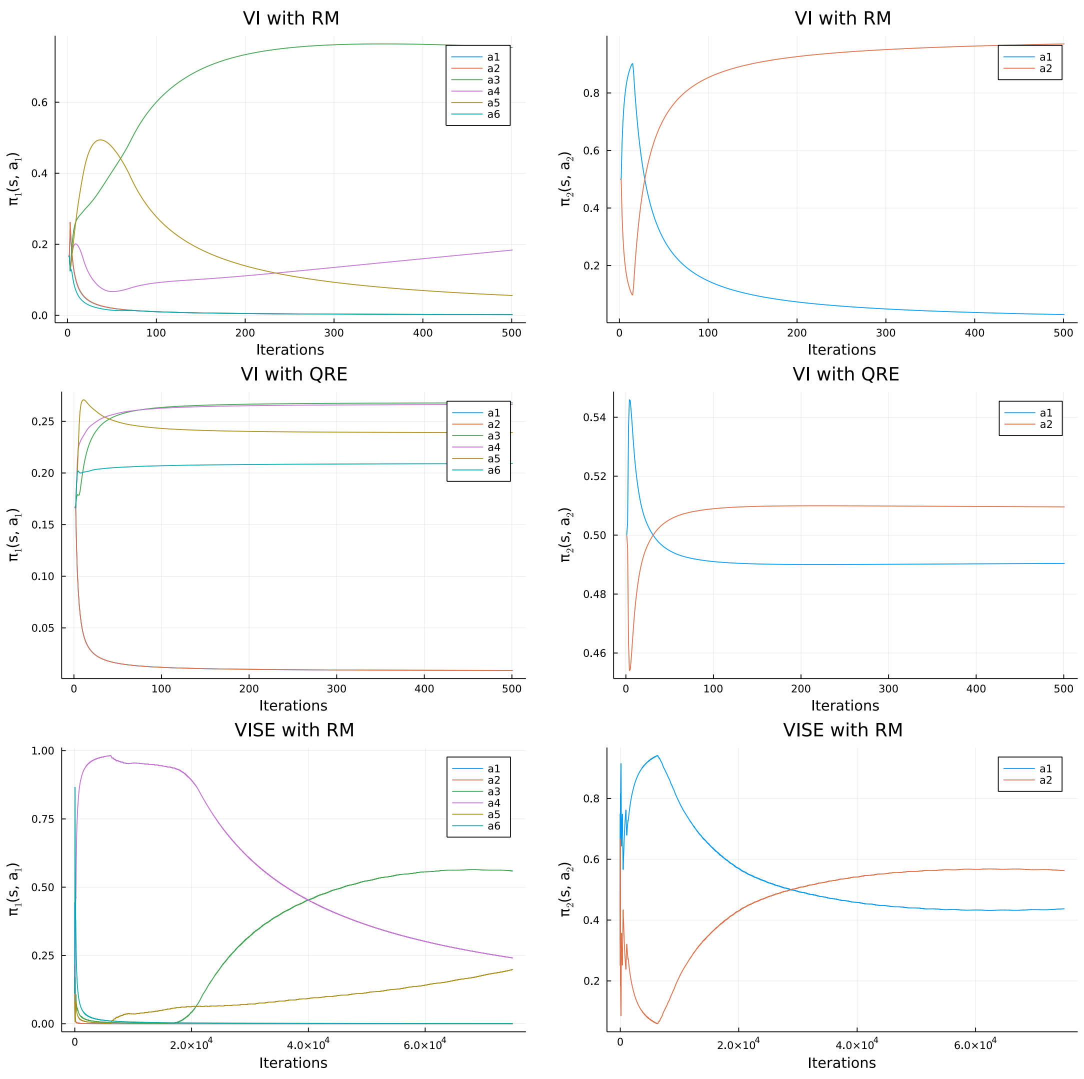

Figure A.9. Policies for each player in a 4×4 stochastic PEG in the initial state. Left is a policy of pursuer, and right is a policy of evader.

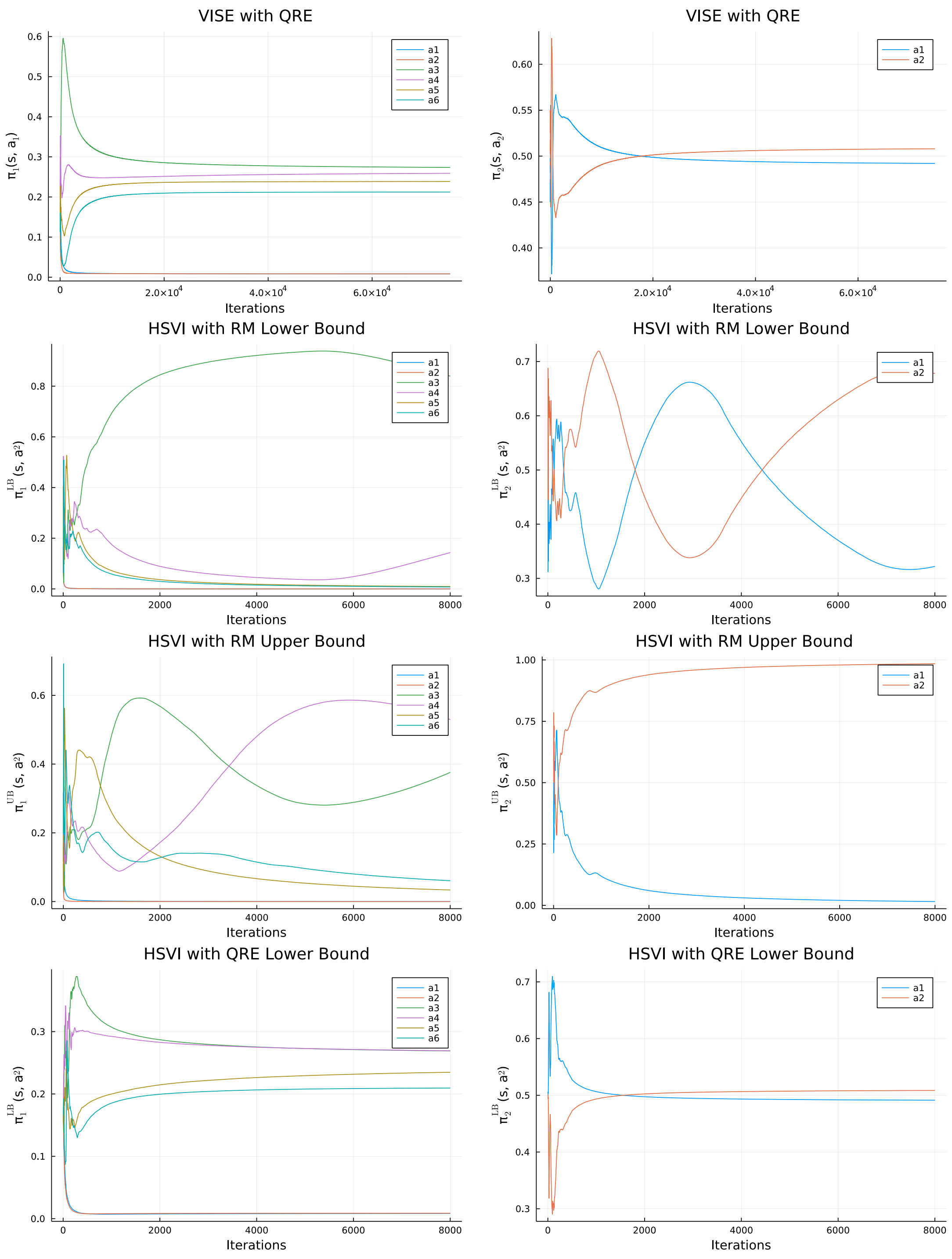

Figure A.10. Policies for each player in a 4×4 stochastic PEG in the initial state. Left is a policy of pursuer, and right is a policy of evader.

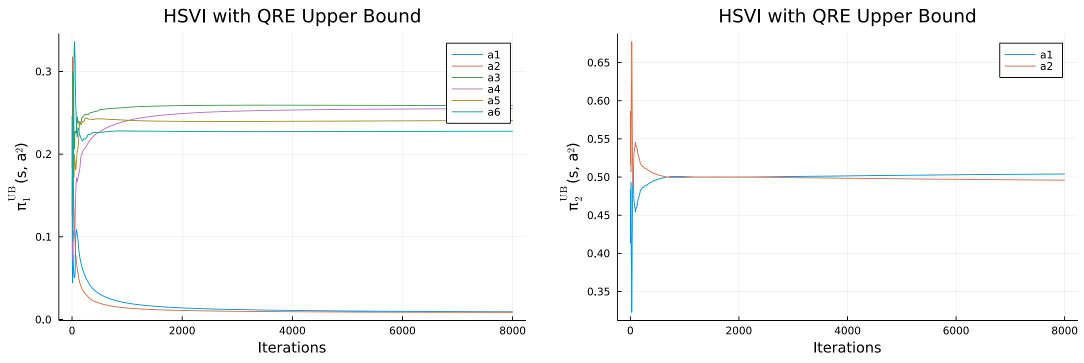

Figure A.11. Policies for each player in a 4×4 stochastic PEG in the initial state. Left is a policy of pursuer, and right is a policy of evader.

A.4 Policies in two non-initial states in stochastic PEG

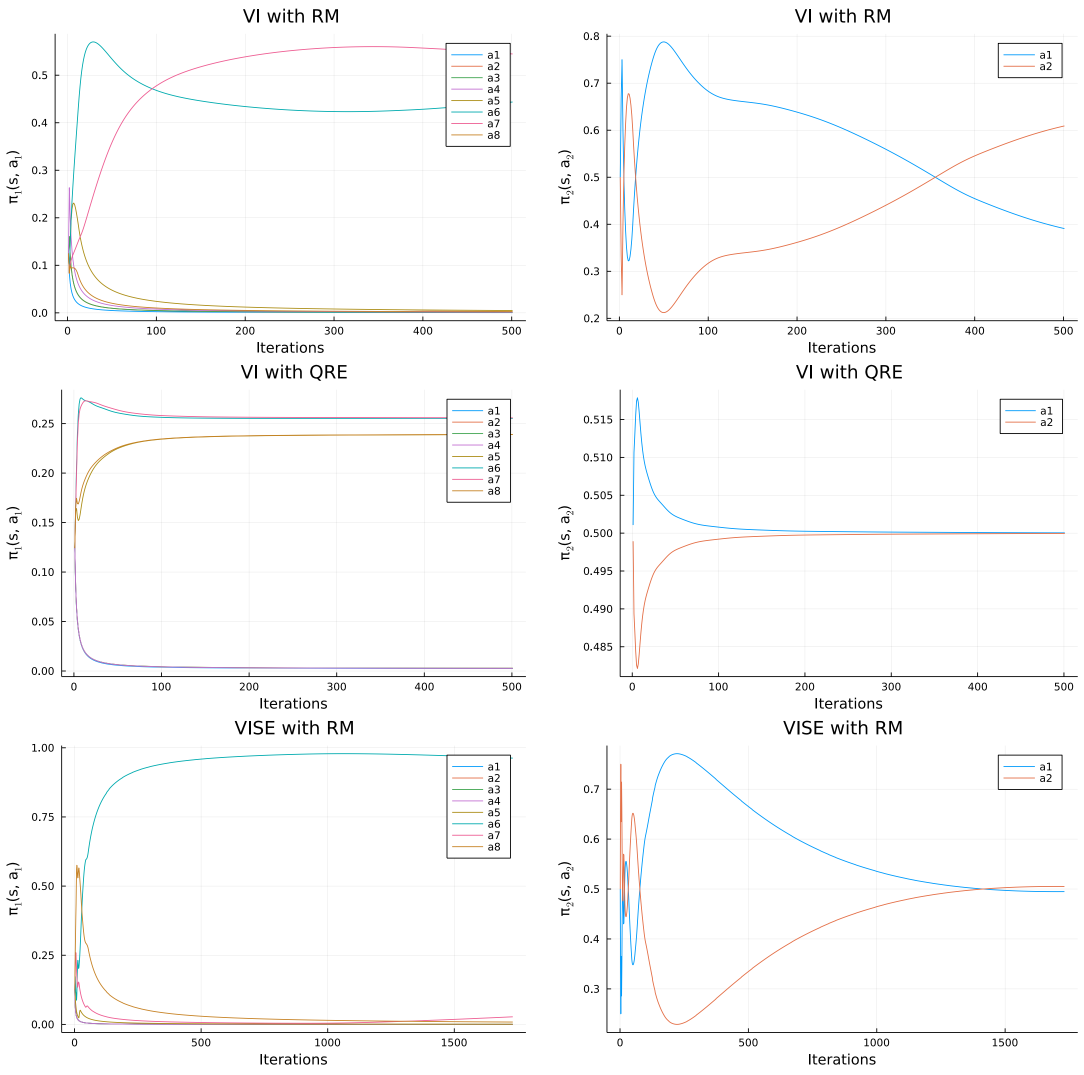

Figure A.12. Policies for each player in a 4×4 stochastic PEG in the non-initial state. Left is a policy of pursuer, and right is a policy of evader.

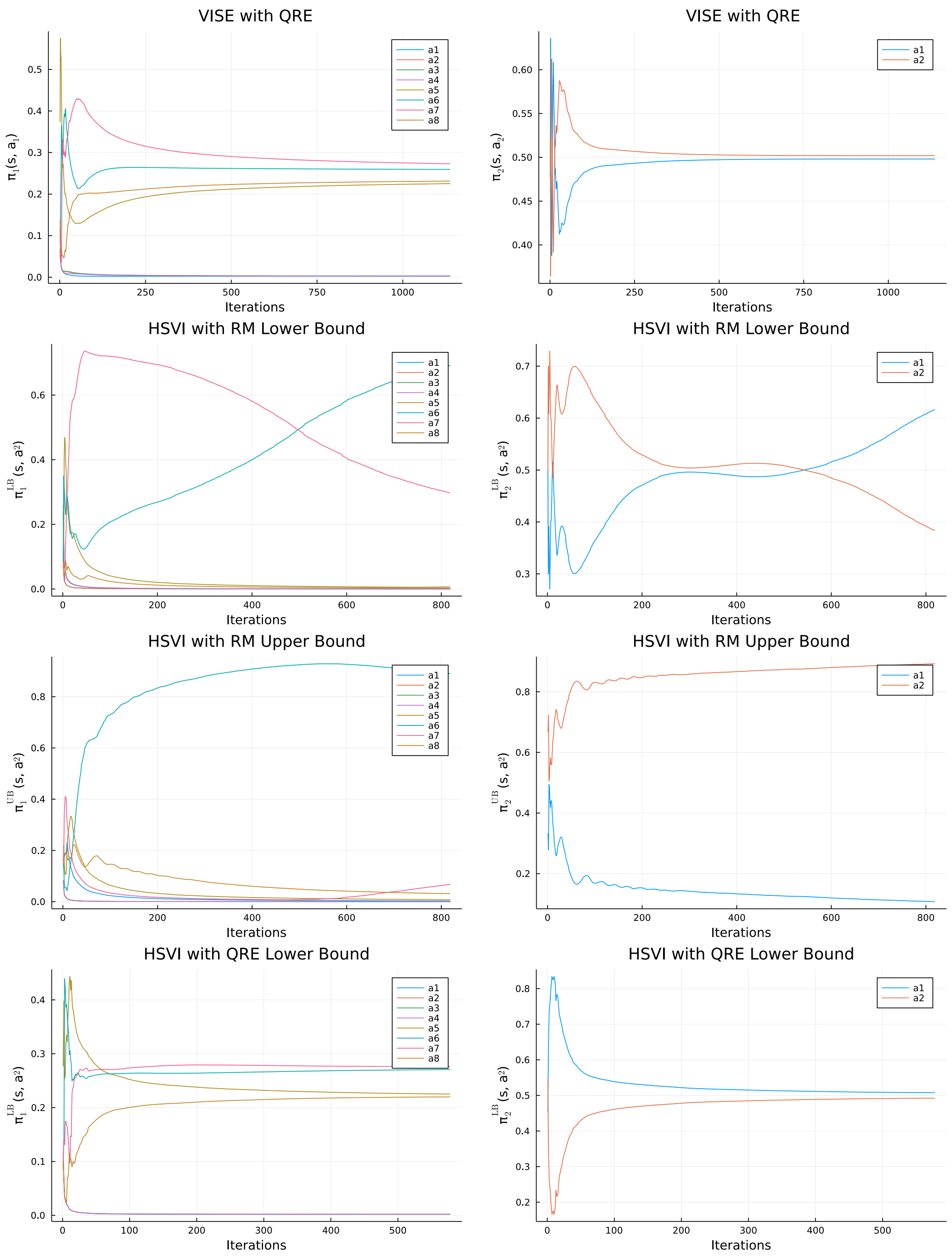

Figure A.13. Policies for each player in a 4×4 stochastic PEG in the non-initial state. Left is a policy of pursuer, and right is a policy of evader.

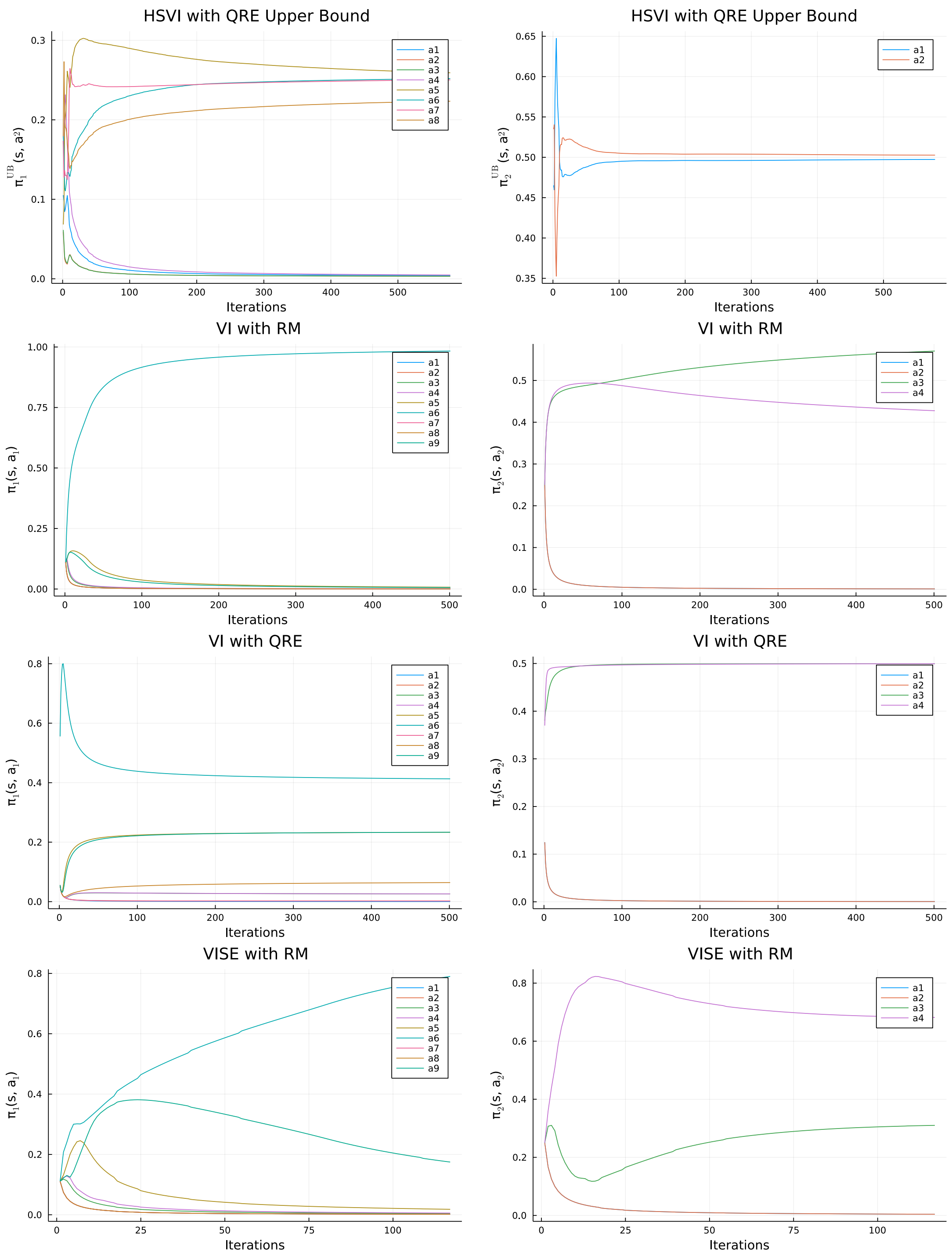

Figure A.14. Policies for each player in a 4×4 stochastic PEG in the non-initial state. Left is a policy of pursuer, and right is a policy of evader.

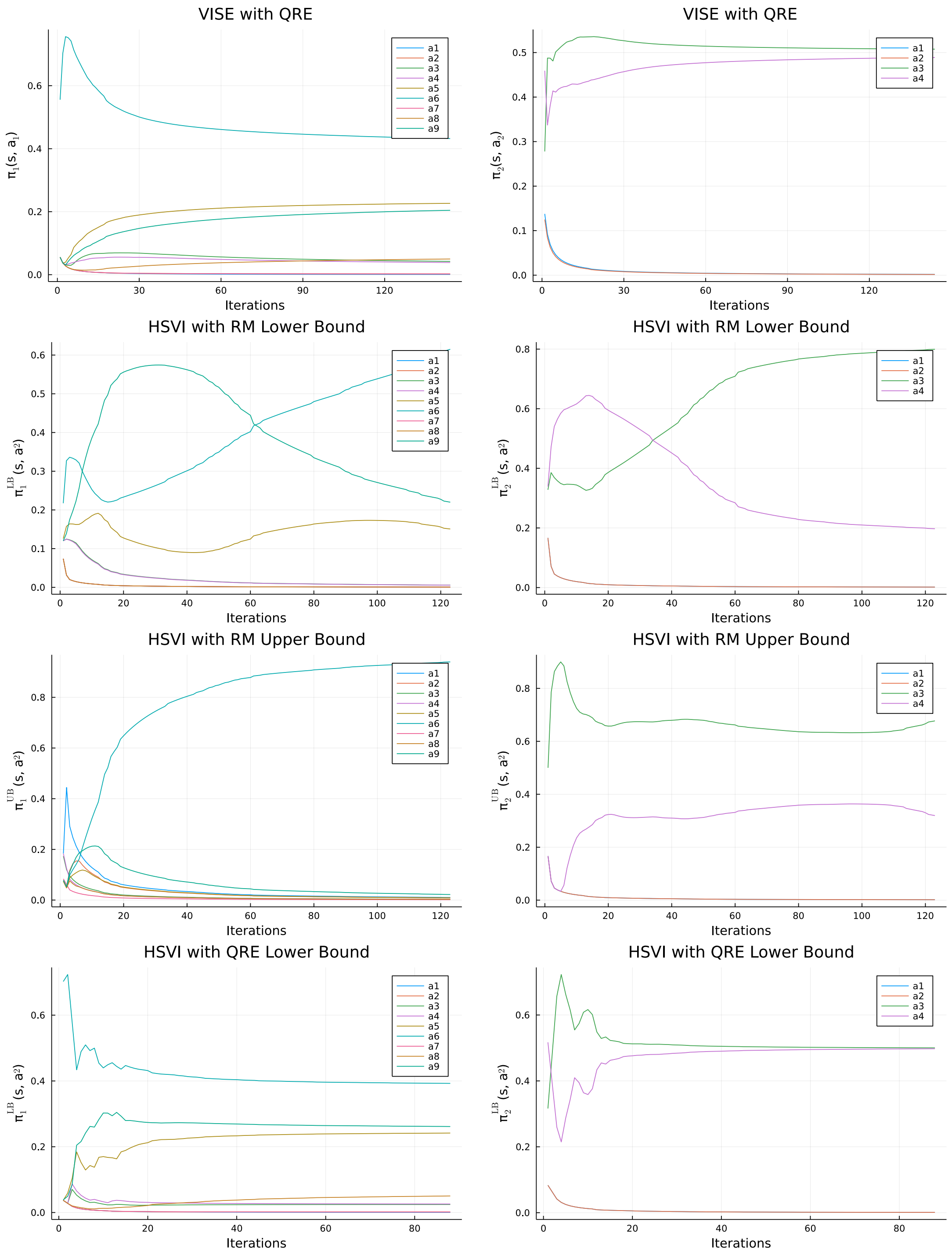

Figure A.15. Policies for each player in a 4×4 stochastic PEG in the non-initial state. Left is a policy of pursuer, and right is a policy of evader.

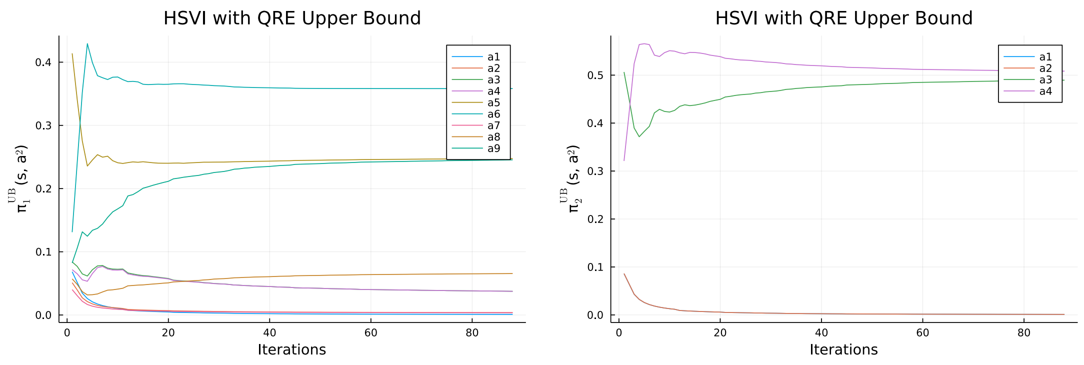

Figure A.16. Policies for each player in a 4×4 stochastic PEG in the non-initial state. Left is a policy of pursuer, and right is a policy of evader.

A.5 Expected values and policy variance in different PEGs

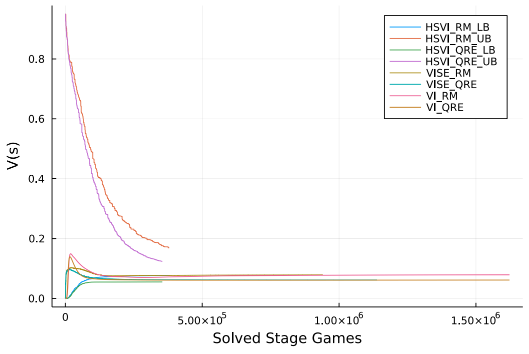

Figure A.17. Expected value in a 5×4 PEG in the initial state.

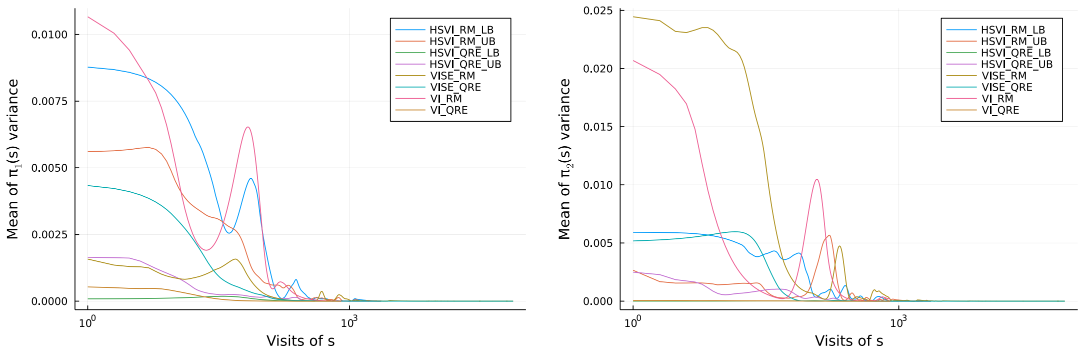

Figure A.18. Policy variance in a 5×4 PEG in the initial state. Left is a policy of pursuer, and right is a policy of evader.

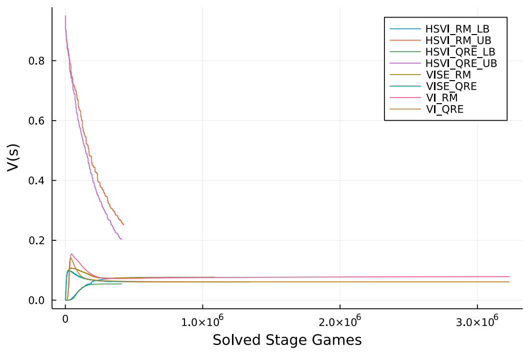

Figure A.19. Expected value in a 5×5 PEG in the initial state.

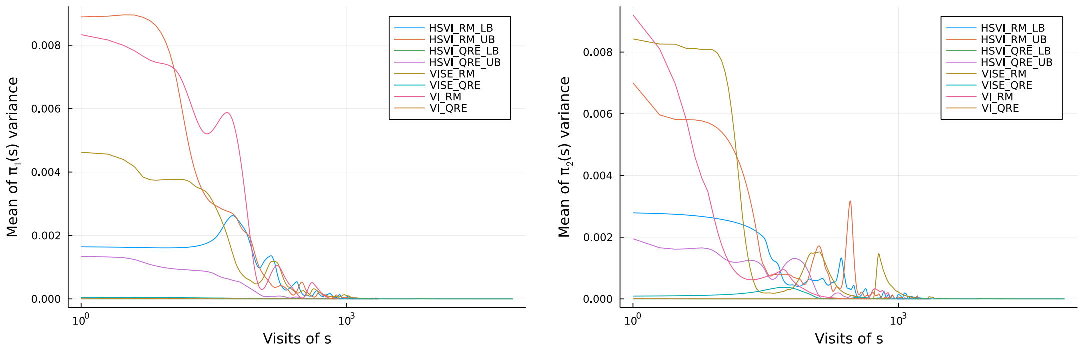

Figure A.20. Policy variance in a 5×5 PEG in the initial state. Left is a policy of pursuer, and right is a policy of evader.

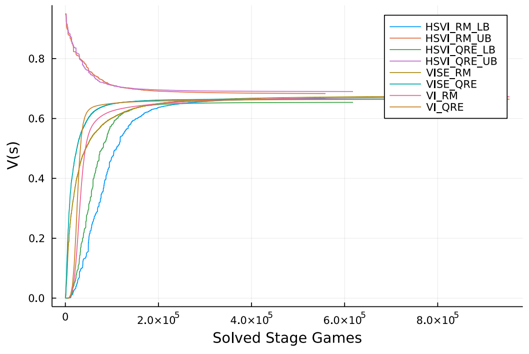

Figure A.21. Expected value in a 5×4 stochastic PEG in the initial state.

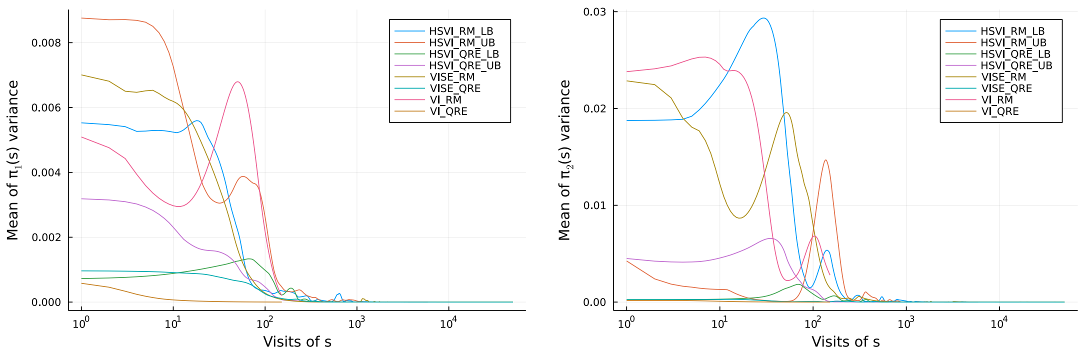

Figure A.22. Policy variance in a 5×4 stochastic PEG in the initial state. Left is a policy of pursuer, and right is a policy of evader.

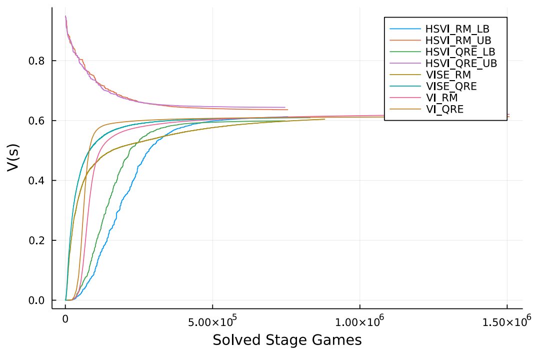

Figure A.23. Expected value in a 5×5 stochastic PEG in the initial state.

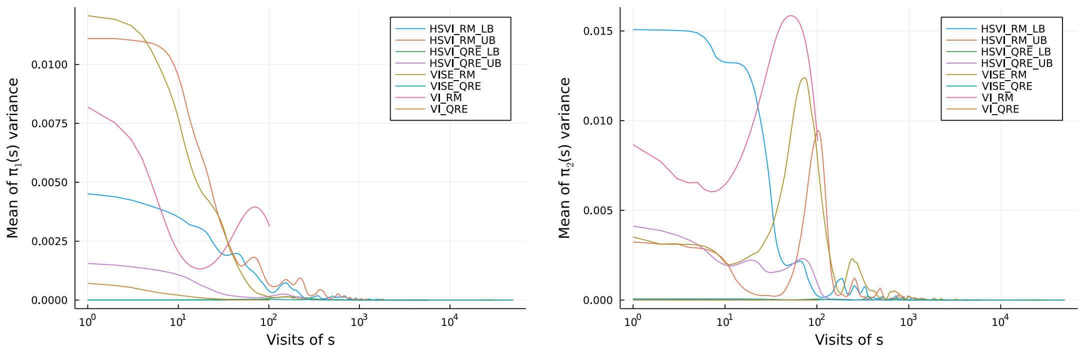

Figure A.24. Policy variance in a 5×5 stochastic PEG in the initial state. Left is a policy of pursuer, and right is a policy of evader.

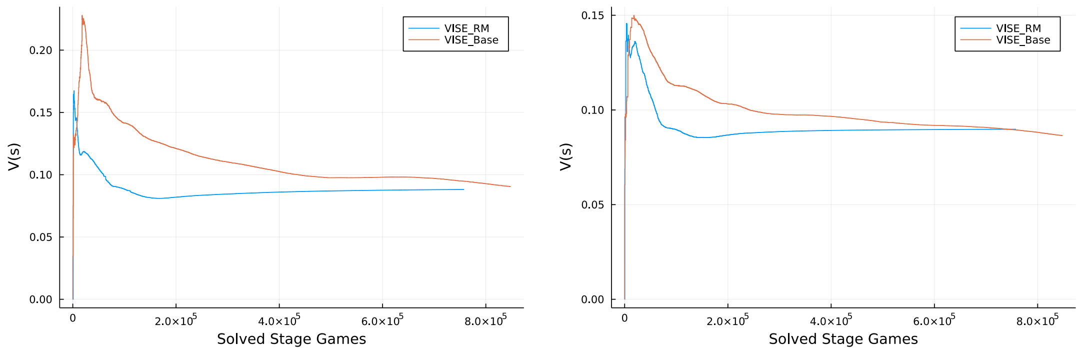

Figure A.25. Expected value in a 4×4 PEG in non-initial states.

A.6 Expected values and policy in non-initial states in PEGs with the usage of baselines

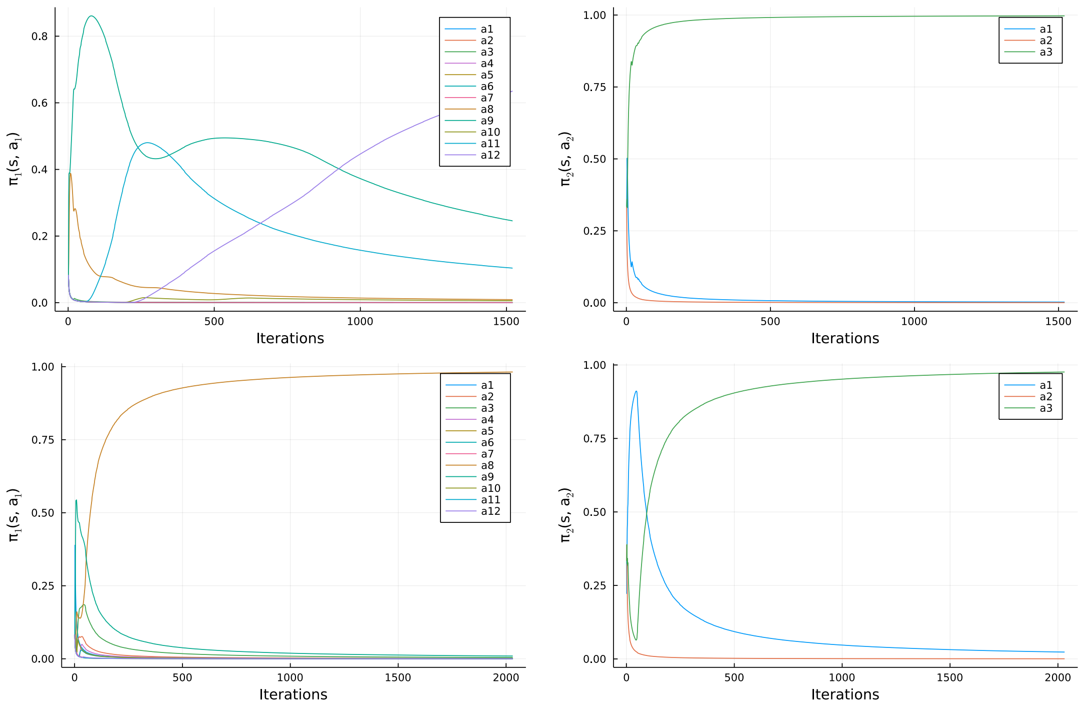

Figure A.26. Policy in a 4×4 PEG in a non-initial state. Left is a policy of pursuer, and right is a policy of evader. Upper plots are without baselines and the bottom ones are with baselines

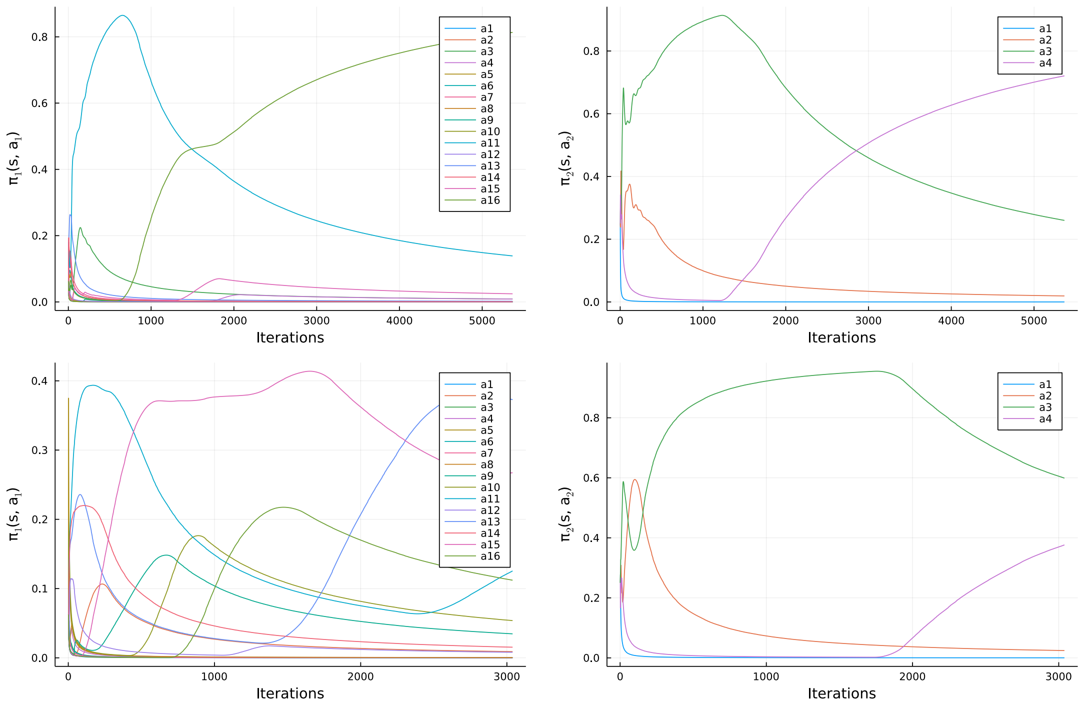

Figure A.27. Policy in a 4×4 PEG in a non-initial state. Left is a policy of pursuer, and right is a policy of evader. Upper plots are without baselines and the bottom ones are with baselines

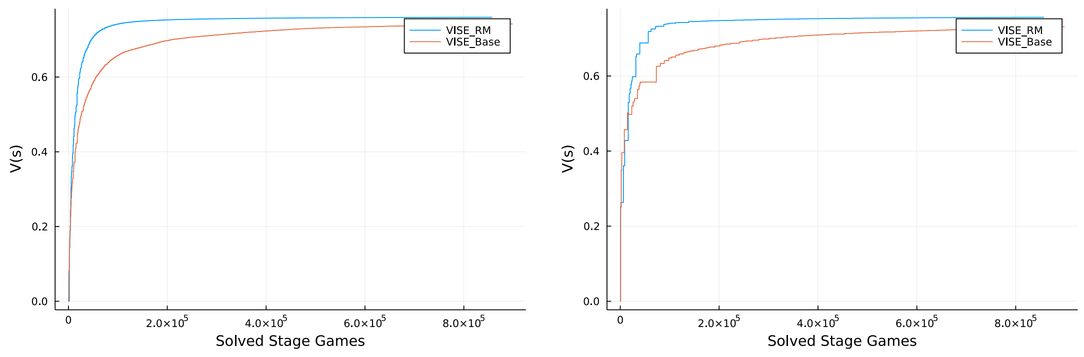

Figure A.28. Expected value in a 4 × 4 stochastic PEG in non-initial states.

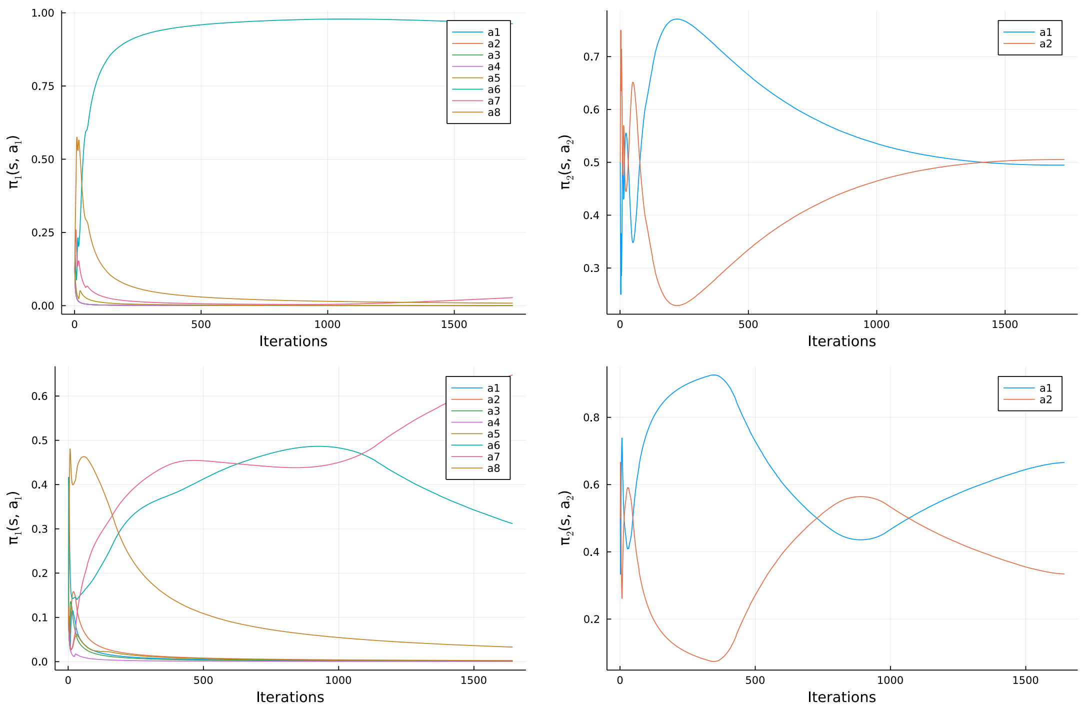

Figure A.29. Policy in a 4 × 4 stochastic PEG in a non-initial state. Left is a policy of pursuer, and right is a policy of evader. Upper plots are without baselines and the bottom ones are with baselines

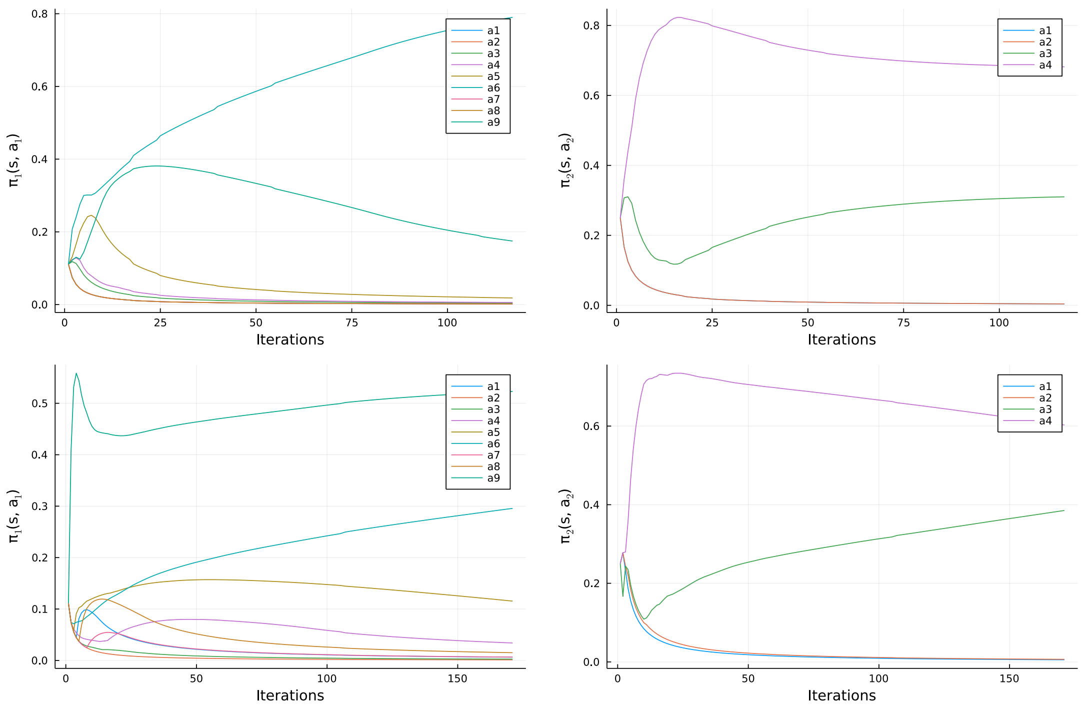

Figure A.30. Policy in a 4×4 stochastic PEG in a non-initial state. Left is a policy of pursuer, and right is a policy of evader. Upper plots are without baselines and the bottom ones are with baselines

A.7 Expected values and policy in different PEGs with the usage of baselines

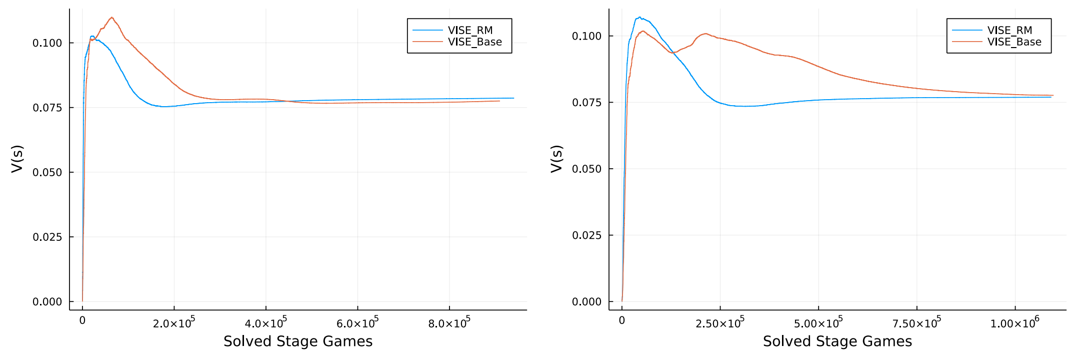

Figure A.31. Expected value in different PEGs in the initial state with and without baselines. Left figure is a 5×4 PEG and right figure is a 5×5 PEG

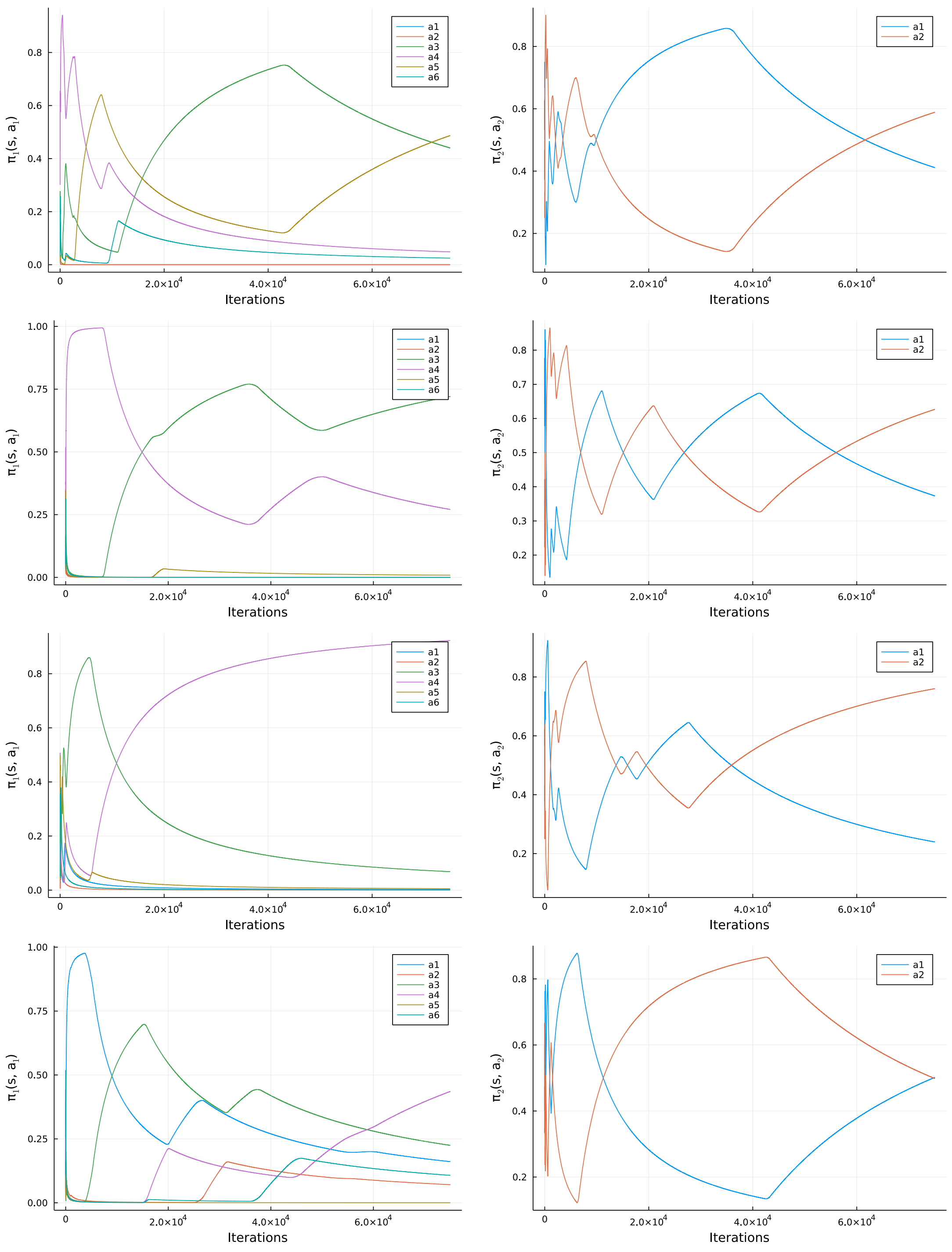

Figure A.32. Policy in a 5×4 and 5×5 PEGs in a initial state with and without baselines. Left is a policy of pursuer, and right is a policy of evader. First row of plots are for a 5×4 PEG without baselines, second row is for same game with baselines, third row is for a 5×5 PEG without baselines and last row is for the same game with baselines.

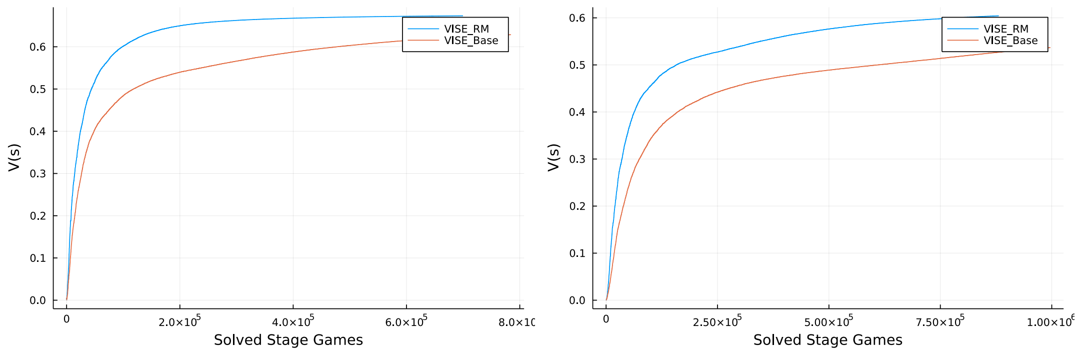

Figure A.33. Expected value in different stochastic PEGs in the initial state with and without baselines. Left figure is a 5×4 PEG and right figure is a 5×5 PEG

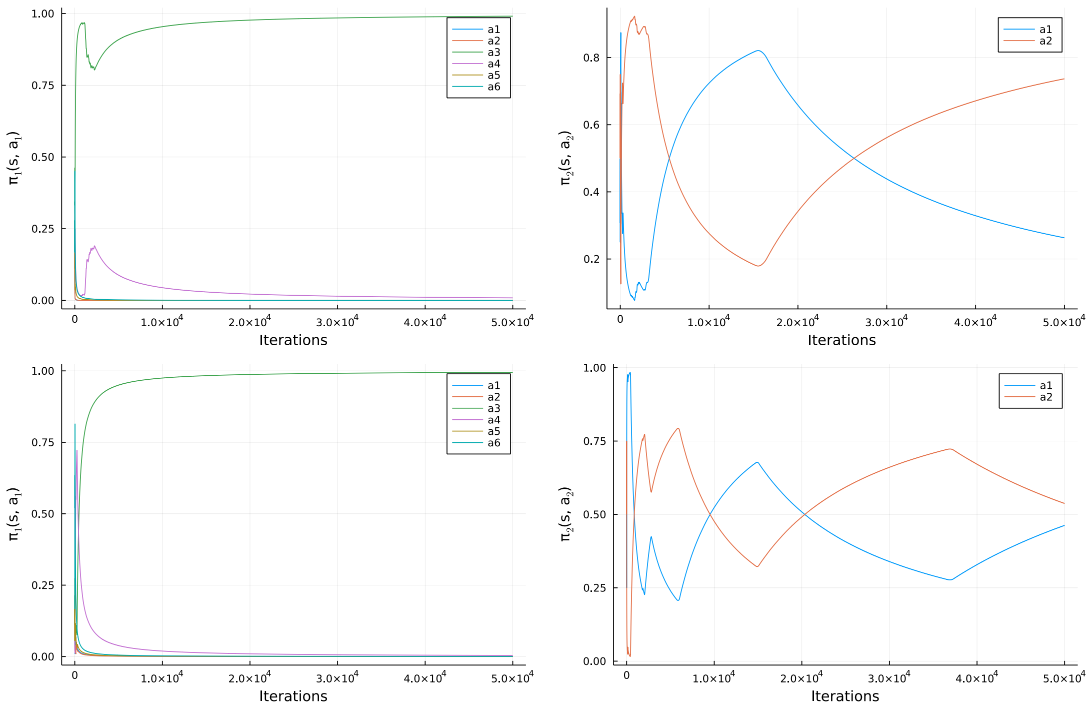

Figure A.34. Policy in a 5×4 stochastic PEG in a initial state with and without baselines. Left is a policy of pursuer, and right is a policy of evader. Upper plots are without baselines and the bottom ones are with baselines.

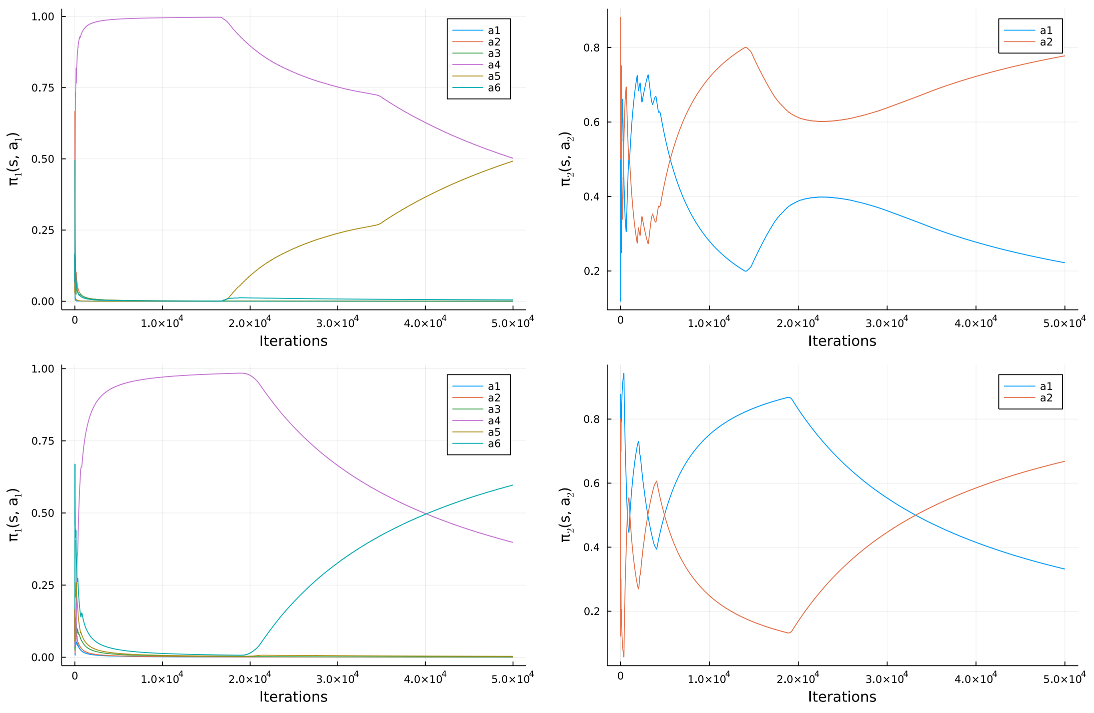

Figure A.35. Policy in a 5×5 stochastic PEG in a initial state with and without baselines. Left is a policy of pursuer, and right is a policy of evader. Upper plots are without baselines and the bottom ones are with baselines.

Appendix B
Implementation

The provided code is split into two Julia modules. First is called SGs, and it implements VI, VISE, VISE with baselines and HSVI for solving the SGs. All of these algorithms may then be run with different functions for an update in each state. First is Bellman update computed by linear programming as in (3.6a)-(3.6d), second uses QRE update, and third uses RM update. The solver for the LP used in this thesis was CPLEX from IBM [36]. However, Julia is a versatile programming language, and this solver could be easily swapped with a different solver without many changes in the code.

The second module is called OSPOSGs, and it implements the HSVI for OS-POSGs with the same methods for solving stage games as in SGs. There is one slight improvement in the OS-POSGs, and that is dividing states into disjunct sets called partitions. Partition is shown to the imperfect information player. Therefore its belief is only on a subset of states in the given partition.

We show the structure of the folders in two follwing directory trees.

```
OSPOSGs
├─games
│ ├─game_format.txt
│ ├─peg03.posg
│ ├─peg03_bf.posg
│ ├─peg04.posg
│ └─peg04_bf.posg
├─src
│ ├─types
│ │ ├─alpha_vector.jl
│ │ ├─args.jl
│ │ ├─context.jl
│ │ ├─game.jl
│ │ ├─game_params.jl
│ │ ├─parsed_game_definition.jl
│ │ ├─partition.jl
│ │ ├─reward.jl
│ │ ├─state.jl
│ │ └─transition.jl
│ ├─cfr.jl
│ ├─hsvi.jl
│ ├─check.jl
│ ├─load.jl
│ ├─logging.jl
│ ├─minmax.jl
│ ├─OSPOSGs.jl
│ ├─presolve.jl
│ ├─qre.jl
│ ├─utils.jl
│ └─value.jl
├─Manifest.toml
├─Project.toml
└─README.md
```

```
SGs
├─games
│ ├─game_format.txt
│ ├─Peg_4_4_0.1_Stoch.fosg
│ ├─Peg_4_4_0.8_Stoch.fosg
│ ├─Peg_4_4_0_1_1.fosg
│ ├─Peg_4_4_1x9.fosg
│ ├─Peg_5_4_0_1_1.fosg
│ ├─Peg_5_4_1x9.fosg
│ └─Peg_5_5_1x9.fosg
├─src
│ ├─FOSGs
│ │ ├─eval.jl
│ │ ├─generate.jl
│ │ ├─graph_utils.jl
│ │ ├─hsvi.jl
│ │ ├─load.jl
│ │ ├─lps.jl
│ │ ├─metagraphs.jl
│ │ ├─peg_generator.jl
│ │ ├─qre.jl
│ │ ├─qre2.jl
│ │ ├─rm.jl
│ │ ├─types.jl
│ │ ├─utils.jl
│ │ ├─values.jl
│ │ └─vi.jl
│ ├─types
│ │ ├─game.jl
│ │ ├─partition.jl
│ │ ├─state.jl
│ │ ├─transition.jl
│ │ └─tree.jl
│ └─SGs.jl
├─Manifest.toml
├─Project.toml
└─README.md
```

References

[1] Wilko Schwarting, Javier Alonso-Mora, and Daniela Rus. Planning and decision-making for autonomous vehicles. *Annual Review of Control, Robotics, and Autonomous Systems*. 2018, 1 187–210.

[2] Shinpei Kato, Eijiro Takeuchi, Yoshio Ishiguro, Yoshiki Ninomiya, Kazuya Takeda, and Tsuyoshi Hamada. An Open Approach to Autonomous Vehicles. *IEEE Micro*. 2015, 35 (6), 60-68. DOI 10.1109/MM.2015.133.

[3] Sven Koenig, Reid Simmons, and others. Xavier: A robot navigation architecture based on partially observable markov decision process models. *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*. 1998, (partially), 91–122.

[4] Ondrej Vanek, Zhengyu Yin, Manish Jain, Branislav Bosansky, Milind Tambe, and Michal Pechoucek. *Game-theoretic resource allocation for malicious packet detection in computer networks.*. In: *AAMAS*. 2012. 905–912.

[5] Karel Horák, Branislav Bošanský, and Michal Pěchouček. Heuristic Search Value Iteration for One-Sided Partially Observable Stochastic Games. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2017, 31 (1),

[6] Karel Horak, Branislav Bosansky, Vojtech Kovarik, and Christopher Kiekintveld. Solving Zero-Sum One-Sided Partially Observable Stochastic Games. *CoRR*. 2020, abs/2010.11243

[7] Jakub Brož. *Using Fast Upper-Bound Approximation in Heuristic Search Value Iteration*. 2021.

[8] Richard S Sutton, and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[9] Christopher JCH Watkins, and Peter Dayan. Q-learning. *Machine learning*. 1992, 8 (3), 279–292.

[10] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*. 1999, 12

[11] Richard Bellman. A Markovian decision process. *Journal of mathematics and mechanics*. 1957, 679–684.

[12] Reid Simmons, and Sven Koenig. Probabilistic Robot Navigation in Partially Observable Environments . 1995,

[13] K.J Åström. Optimal control of Markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*. 1965, 10 (1), 174-205. DOI https://doi.org/10.1016/0022-247X(65)90154-X.

[14] Paul A Gagniuc. *Markov chains: from theory to implementation and experimentation*. John Wiley & Sons, 2017.

[15] Edward J. Sondik. The Optimal Control of Partially Observable Markov Processes Over the Infinite Horizon: Discounted Costs. *Operations Research*. 1978, 26 (2), 282–304.

[16] Trey Smith, and Reid G. Simmons. Heuristic Search Value Iteration for POMDPs. *CoRR*. 2012, abs/1207.4166

[17] Yoav Shoham, and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.

[18] 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.

[19] R.D. Luce, and H. Raiffa. *Games and Decisions: Introduction and Critical Survey*. Dover Publications, 1989. ISBN 9780486659435.
`https://books.google.cz/books?id=msC_h 0wtC08C`.

[20] Constantinos Daskalakis, Paul W Goldberg, and Christos H Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*. 2009, 39 (1), 195–259.

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

[22] Lloyd S Shapley. Stochastic games. *Proceedings of the national academy of sciences*. 1953, 39 (10), 1095–1100.

[23] Kristoffer Arnsfelt Hansen, Rasmus Ibsen-Jensen, and Peter Bro Miltersen. *The complexity of solving reachability games using value and strategy iteration*. In: *International Computer Science Symposium in Russia*. 2011. 77–90.

[24] Luke Zettlemoyer, Brian Milch, and Leslie Kaelbling. *Multi-Agent Filtering with Infinitely Nested Beliefs*. In: D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, eds. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2008.
`https://proceedings.neurips.cc/paper/2008/file/6c3cf77d52820cd0fe64 6d38bc2145ca-Paper.pdf`.

[25] Richard D. McKelvey, and Thomas R. Palfrey. Quantal Response Equilibria for Normal Form Games. *Games and Economic Behavior*. 1995, 10 (1), 6-38. DOI https://doi.org/10.1006/game.1995.1023.

[26] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989,

[27] Gavin A Rummery, and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. Citeseer, 1994.

[28] Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. *Monte Carlo Sampling for Regret Minimization in Extensive Games*. In: Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, eds. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2009.
`https://proceedings.neurips.cc/paper/2009/file/00411460f7c92d2124a6 7ea0f4cb5f85-Paper.pdf`.

[29] Joshua Romoff, Alexandre Piche, Peter Henderson, Vincent François-Lavet, and Joelle Pineau. Reward Estimation for Variance Reduction in Deep Reinforcement Learning. *CoRR*. 2018, abs/1805.03359

[30] Martin Schmid, Neil Burch, Marc Lanctot, Matej Moravcik, Rudolf Kadlec, and Michael Bowling. Variance Reduction in Monte Carlo Counterfactual Regret Minimization (VR-MCCFR) for Extensive Form Games using Baselines. *CoRR*. 2018, abs/1809.03057

[31] 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/2007/file/08d98638c6fcd194a4b1e6992063e944-Paper.pdf`.

[32] Gordon Anderson, Ian Crawford, and Andrew Leicester. *Efficiency analysis and the lower convex hull approach.* 2008.

[33] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review.* 2017, 59 (1), 65–98.

[34] Rufus Isaacs. *Differential games: a mathematical theory with applications to warfare and pursuit, control and optimization.* Courier Corporation, 1999.

[35] Yoav Freund, and Robert E Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior.* 1999, 29 (1-2), 79–103.

[36] IBM ILOG Cplex. V12. 1: User's Manual for CPLEX. *International Business Machines Corporation.* 2009, 46 (53), 157.