Mathematical Formulation of the Energy Combat Game

Zeyu Liang
Northeastern University
Email: liang.zey@northeastern.edu

December 12, 2025

1 Game Formulation

We model the Energy Combat Game as a two-player zero-sum Markov game

$$\mathcal{G} = (\mathcal{S}, \mathcal{A}_1, \mathcal{A}_2, P, r, \gamma), \tag{1}$$

with a fixed discount factor $\gamma = 1$ since the episodic game always terminates.

1.1 State and Action Spaces

Each player $i \in \{1, 2\}$ has an energy level $e_i \in \{0, 1, 2\}$. The state space is

$$\mathcal{S} = \{0, 1, 2\} \times \{0, 1, 2\} \cup \{s_{\text{terminal}}\}. \tag{2}$$

Each player chooses one of five actions:

$$\mathcal{A}_i = \{\textsc{Store}, \text{AS}, \text{AB}, \text{DS}, \text{DB}\}. \tag{3}$$

Action feasibility follows energy and opponent constraints:

$$\text{AS} : e_i \geq 1, \quad \text{AB} : e_i \geq 2, \tag{4}$$

$$\text{DS} : e_{-i} \geq 1, \quad \text{DB} : e_{-i} \geq 2. \tag{5}$$

Attack and defense strengths are

$$\alpha(a) \in \{0, 1, 2\}, \tag{6}$$

$$\delta(a) \in \{0, 1, 2\}, \tag{7}$$

where $\alpha(\text{AS}) = 1$, $\alpha(\text{AB}) = 2$, $\delta(\text{DS}) = 1$, $\delta(\text{DB}) = 2$.

1.2 Hit Rule

Let (a_1, a_2) be the joint action and define attack strengths

$$A_i = \alpha(a_i), \quad D_i = \delta(a_i). \tag{8}$$

A player is hit if the incoming attack is not neutralized by cancellation or matching defense.
(1) Both attack:

- $A_1 = A_2$: attacks cancel, no hit.
- $A_1 > A_2$: player 2 is hit.
- $A_2 > A_1$: player 1 is hit.

(2) Only player 1 attacks ($A_1 > 0, A_2 = 0$):

$$\text{Player 2 hit} \iff D_2 \neq A_1. \tag{9}$$

(3) Only player 2 attacks ($A_1 = 0, A_2 > 0$):

$$\text{Player 1 hit} \iff D_1 \neq A_2. \tag{10}$$

Thus, a defense blocks an attack *only* when sizes match exactly.

1.3 Transitions

If a player is hit:

$$P(s_{\text{terminal}} \mid s, a_1, a_2) = 1. \tag{11}$$

If no hit occurs, energy updates deterministically:

$$e_i' = \begin{cases} \min(e_i + 1, 2), & a_i = \textsc{Store}, \\ e_i - A_i, & A_i > 0, \\ e_i, & \text{otherwise.} \end{cases} \tag{12}$$

The next state is $s' = (e_1', e_2')$.

1.4 Reward Structure

Player 1 receives

$$r(s, a_1, a_2) = \begin{cases} +1, & \text{player 2 hit}, \\ -1, & \text{player 1 hit}, \\ -c_{\text{step}}, & \text{nonterminal step}, \\ 0, & s = s_{\text{terminal}}. \end{cases} \tag{13}$$

Player 2 obtains the opposite reward.

2 Self-Play Learning and Q-Value Computation

2.1 Value Function

Given the Markov game defined in Section 1, a joint policy $\pi = (\pi_1, \pi_2)$ induces the discounted return

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t) \,\middle|\, s_0 = s \right], \tag{14}$$

where T is the termination time and we use a discount factor

$$\gamma = 0.9, \tag{15}$$

which slightly prefers faster wins and stabilizes the learning process.

For evaluating individual actions, we use the action-value function

$$Q^\pi(s, a_1, a_2) = \mathbb{E}_\pi \left[\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t) \,\middle|\, s_0 = s,\ a_0 = (a_1, a_2) \right]. \tag{16}$$

Algorithm 1 Independent Self-Play Q-Learning (matches `train_self_play`)

```
Initialize $Q_0(s,a)$ and $Q_1(s,a)$ to zero
for episode = 0 to $N-1$ do
  Reset environment; observe $s = (e_0, e_1)$
  $\epsilon \leftarrow \epsilon_{\text{start}} + \frac{\text{episode}}{N-1}(\epsilon_{\text{end}} - \epsilon_{\text{start}})$
  while $s$ is not terminal do
    for $i \in \{0,1\}$ do
      Compute valid mask $M_i = \text{ValidActions}(e_i, e_{1-i})$
      if random $< \epsilon$ then
        Sample $a_i$ uniformly from $\{a : M_i(a) = 1\}$
      else
        $a_i = \arg\max_{a:M_i(a)=1} Q_i(s,a)$
      end if
    end for
    Execute $(a_0, a_1)$; observe $r_0, r_1$, next state $s' = (e'_0, e'_1)$
    for $i \in \{0,1\}$ do
      Compute next valid mask $M'_i$
      if $s'$ is terminal then
        $\text{target}_i \leftarrow r_i$
      else
        $\text{target}_i \leftarrow r_i + \gamma \max_{a:M'_i(a)=1} Q_i(s', a)$
      end if
      $Q_i(s,a_i) \leftarrow Q_i(s,a_i) + \alpha(\text{target}_i - Q_i(s,a_i))$
    end for
    $s \leftarrow s'$
  end while
end for
return $Q_0, Q_1$
```

2.2 Q-Learning in Self-Play

We approximate equilibrium behavior through independent tabular Q-learning. Each player $i \in \{0,1\}$ maintains a Q-table $Q_i(s, a_i)$ and updates it according to

$$Q_i(s,a_i) \leftarrow Q_i(s,a_i) + \alpha\Big[r_i + \gamma \max_{a'_i \in \mathcal{A}_i(s')} Q_i(s', a'_i) - Q_i(s,a_i)\Big], \tag{17}$$

where the maximization is restricted to actions feasible in the next state s'. Both players select actions via an ε-greedy exploration scheme, and ε linearly decays between ϵ_{start} and ϵ_{end} across training.

The transition $(s, a_0, a_1) \to s'$ follows the deterministic hit and energy-update rules described earlier.

2.3 Self-Play Training Procedure

Algorithm 1 summarizes the learning process implemented in `energy_game_selfplay.py`.

2.4 Mixed Strategy Extraction via Softmax

After training, we compute a symmetric Q-table

$$Q_{\text{avg}} = \tfrac{1}{2}(Q_0 + Q_1), \tag{18}$$

and derive a stochastic policy via a softmax distribution:

$$\pi(a \mid s) \propto \exp\left(\frac{Q_{\text{avg}}(s,a)}{\tau}\right), \tag{19}$$

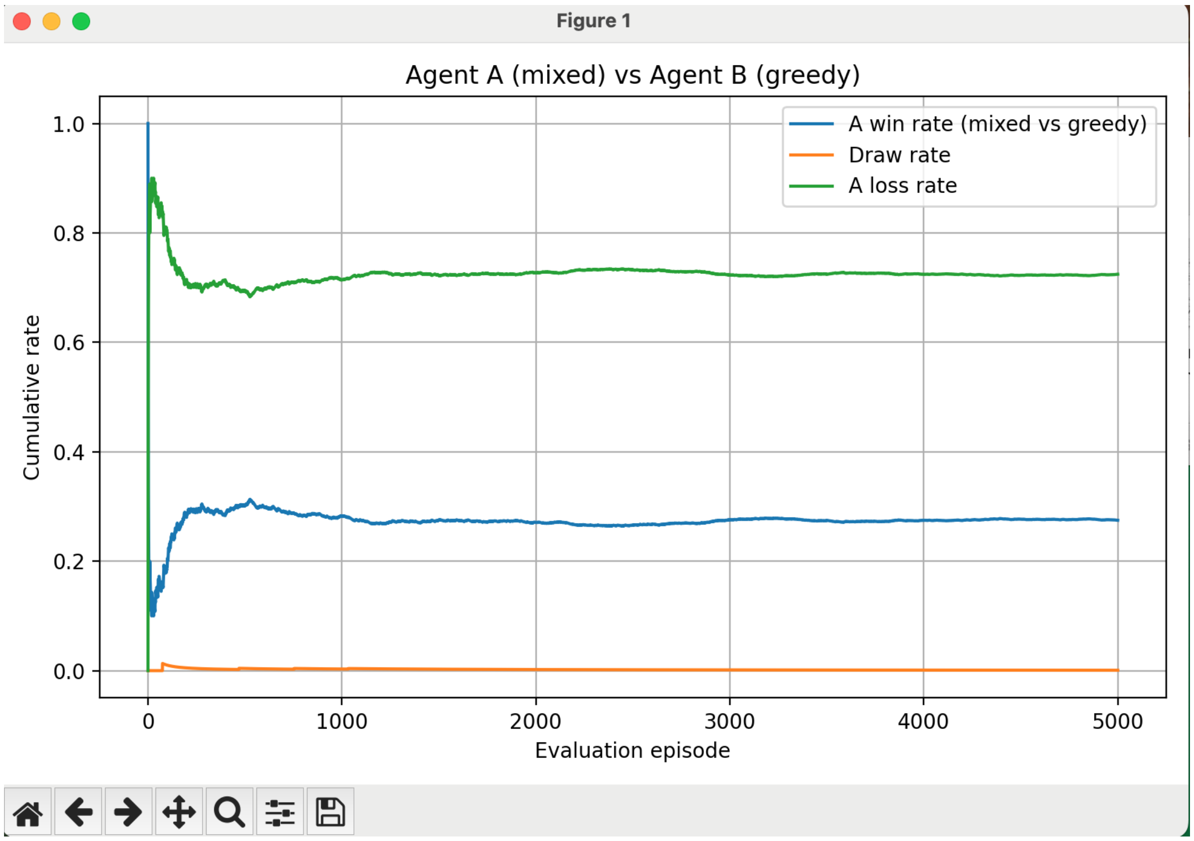


Figure 1: Mixed VS Greedy.

where τ is a temperature parameter. Invalid actions are assigned zero probability and the remaining probabilities are renormalized.

This yields a mixed strategy consistent with the empirical behavior of the trained agents, shown in Algorithm 2. Result of mixed policy using softmax against greedy policy is shown in Figure 1.

3 Opponent Modeling and Best-Response Learning

In addition to self-play, we study how well different opponent strategies (e.g., greedy, self-play mixed, optimal equilibrium) can be exploited by a learner that explicitly models the opponent's behavior. In these experiments, we fix player B's policy and train player A as a best response, while maintaining an empirical opponent model.

3.1 Empirical Opponent Model

For player A, the opponent (player B) is treated as part of the environment, but we keep an explicit estimate of B's conditional action distribution. For each state $s \in \mathcal{S}$ and opponent action $a_B \in \mathcal{A}_B$, we maintain a count

$$C(s, a_B) \in \mathbb{N}. \tag{20}$$

Whenever a transition (s, a_A, a_B, s') is observed, we update

$$C(s, a_B) \leftarrow C(s, a_B) + 1. \tag{21}$$

Algorithm 2 Softmax Policy Extraction (matches `stochastic_policy_from_Q`)

```
1: for each state $s$ do
2:     Compute valid mask $M$
3:     Let $z(a) = Q_{\text{avg}}(s, a)$ if $M(a) = 1$, else $z(a) = -\infty$
4:     Compute $p(a) = \exp(z(a)/\tau) / \sum_{a':M(a')=1} \exp(z(a')/\tau)$
5:     Store distribution $p(\cdot)$ as $\pi(\cdot \mid s)$
6: end for
7: return policy $\pi$
```

The empirical opponent policy is then given by

$$\hat{\pi}_B(a_B \mid s) = \frac{C(s, a_B)}{\sum_{a'_B} C(s, a'_B) + \varepsilon}, \tag{22}$$

where a small $\varepsilon > 0$ (e.g., implicit pseudocount of 1) is used in practice to avoid division by zero. This provides a state-conditional belief over the opponent's behavior.

3.2 Best-Response Q-Learning Against a Fixed Opponent

In our experiments, we consider a *fixed* opponent policy π_B (e.g., greedy, mixed from self-play, or optimal equilibrium), and train player A to learn a best-response Q-function $Q_A(s, a_A)$.

At each step, given the current state s:

- The opponent's action is sampled from the fixed policy $\pi_B(\cdot \mid s)$.
- Player A chooses an action a_A using ε-greedy exploration with respect to Q_A.

The environment then transitions to s' and returns reward r_A to player A.

The Q-update for player A is

$$Q_A(s, a_A) \leftarrow Q_A(s, a_A) + \alpha \Big[r_A + \gamma \max_{a'_A \in \mathcal{A}_A(s')} Q_A(s', a'_A) - Q_A(s, a_A) \Big], \tag{23}$$

with discount factor $\gamma = 0.9$ and learning rate α. As before, the maximization is restricted to valid actions in s'.

The opponent model $\hat{\pi}_B$ is updated in parallel, and can be used post-hoc to analyze the learned best-response or to compute expected exploitation against the empirical opponent behavior.

3.3 Best-Response with Opponent Modeling: Pseudocode

Algorithm 3 summarizes the training of player A against a fixed opponent policy π_B, together with an empirical opponent model. The result of opponent modeling vs mixed strategy trained in section 2 in shown in figure 2. The result shows that agent using opponent modeling wins around 74 percent of the time, which means that the mixed strategy trained in section 2 is not yet optimal.

4 Optimal Strategy Computation

In addition to self-play learning, we compute the game-theoretic optimal strategy using value iteration for zero-sum simultaneous-move Markov games. This yields both the state-value function and the Nash equilibrium policy, against which we evaluate the learned agents.

Algorithm 3 Best-Response Q-Learning with Opponent Modeling

```
Require: Fixed opponent policy $\pi_B(a_B \mid s)$
 1: Initialize $Q_A(s, a_A)$ to zero
 2: Initialize opponent counts $C(s, a_B) \leftarrow 1$ for all $(s, a_B)$
 3: for episode = 0 to $N - 1$ do
 4:   Reset environment; observe initial state $s$
 5:   Set $\epsilon$ (e.g., linearly decaying over episodes)
 6:   while $s$ is not terminal do
 7:     Sample opponent action $a_B \sim \pi_B(\cdot \mid s)$
 8:     Compute valid mask $M_A$ for player A at state $s$
 9:     if random $< \epsilon$ then
10:       Sample $a_A$ uniformly from $\{a : M_A(a) = 1\}$
11:     else
12:       $a_A = \arg\max_{a:M_A(a)=1} Q_A(s, a)$
13:     end if
14:     Execute joint action $(a_A, a_B)$
15:     Observe reward $r_A$, next state $s'$, and terminal flag
16:     Update opponent model: $C(s, a_B) \leftarrow C(s, a_B) + 1$
17:     Compute valid mask $M'_A$ at $s'$
18:     if $s'$ is terminal then
19:       target $\leftarrow r_A$
20:     else
21:       target $\leftarrow r_A + \gamma \max_{a:M'_A(a)=1} Q_A(s', a)$
22:     end if
23:     $Q_A(s, a_A) \leftarrow Q_A(s, a_A) + \alpha(\text{target} - Q_A(s, a_A))$
24:     $s \leftarrow s'$
25:   end while
26: end for
27: return $Q_A$ and empirical opponent policy $\hat{\pi}_B$
```

4.1 Bellman–Shapley Operator for Zero-Sum Markov Games

For each state s, let $\mathcal{A}_0(s)$ and $\mathcal{A}_1(s)$ denote the feasible actions for the two players, determined by the energy constraints. Given the deterministic transition $s' = f(s, a_0, a_1)$ and reward $r(s, a_0, a_1)$ (player 0's payoff), the value of a state under optimal play is defined by the Bellman–Shapley equation:

$$V(s) = \max_{\pi_0(\cdot|s)} \min_{\pi_1(\cdot|s)} \sum_{a_0,a_1} \pi_0(a_0 \mid s)\, \pi_1(a_1 \mid s)\, Q(s, a_0, a_1), \tag{24}$$

where the state-action value is

$$Q(s, a_0, a_1) = r(s, a_0, a_1) + \gamma V(f(s, a_0, a_1)), \tag{25}$$

with discount factor $\gamma = 0.9$.

Since the action space is small and mixed strategies may be required, we compute the minimax value

$$V(s) = \max_{\pi_0} \min_{\pi_1} \pi_0^\top Q_s \pi_1, \tag{26}$$

where Q_s is the payoff matrix for state s containing all $Q(s, a_0, a_1)$ entries over feasible actions. The maximization and minimization over distributions are solved using linear programming for each state.

4.2 Value Iteration for Optimal Equilibrium

We perform value iteration over the finite state space. Starting from an initial $V_0(s) = 0$, we iteratively apply the Bellman–Shapley operator

$$V_{k+1}(s) = \text{ShapleyBackup}(s, V_k), \tag{27}$$

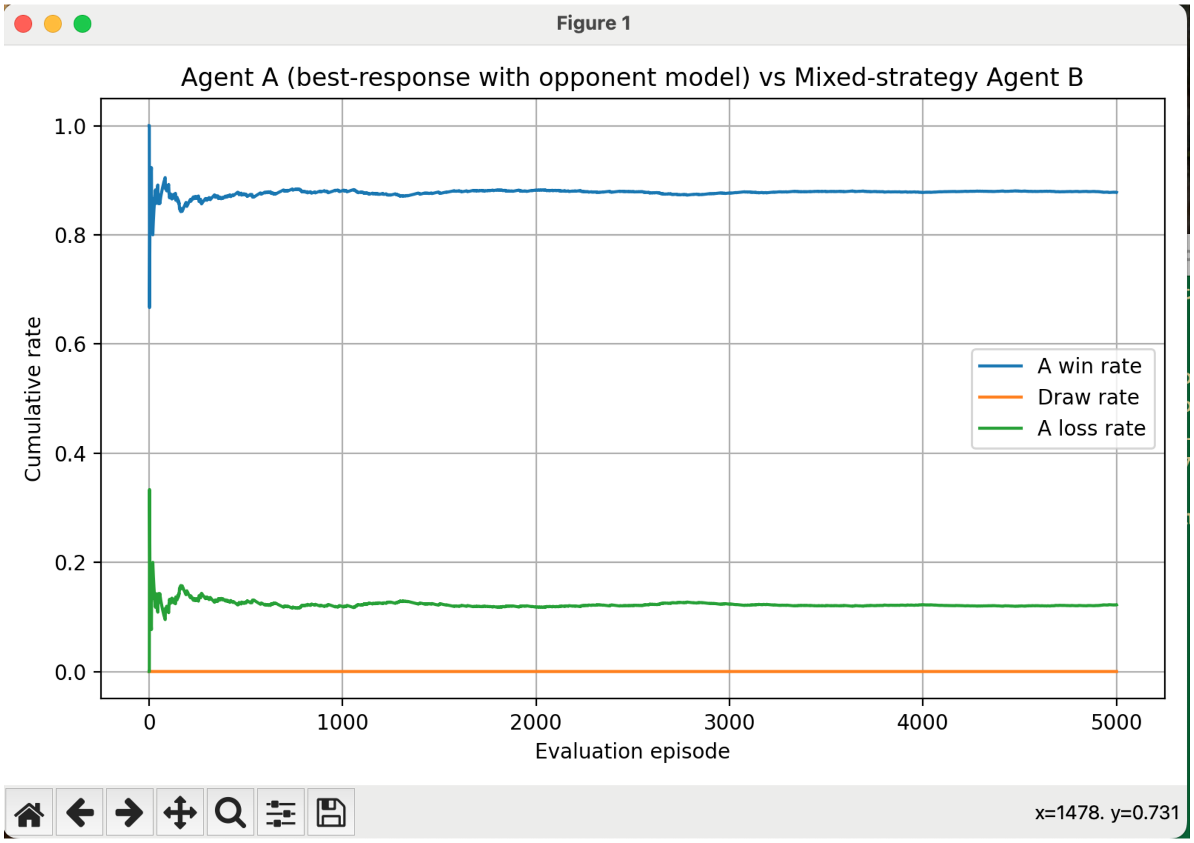


Figure 2: Opponent Modeling VS Mixed Strategy.

where the backup computes $Q(s, a_0, a_1)$ from V_k, forms the payoff matrix Q_s, and solves the associated minimax problem to obtain $V_{k+1}(s)$ and the corresponding optimal mixed policy.

Convergence is guaranteed since the game is finite and $\gamma < 1$. Upon termination, we extract the optimal equilibrium policy $\pi_0^\star(a_0 \mid s)$ for player 0 and $\pi_1^\star(a_1 \mid s)$ for player 1.

4.3 Pseudocode: Optimal Value Iteration

Algorithm 4 summarizes the procedure implemented in `solve_optimal_equilibrium.py`. The final optimal policy is shown in figure 3 and the result of opponent modeling VS optimal policy is shown in figure 4. As shown, agent using opponent modeling and agent using optimal policy both win around 50 percent of the time, which indicates this is indeed optimal.

5 Conclusion

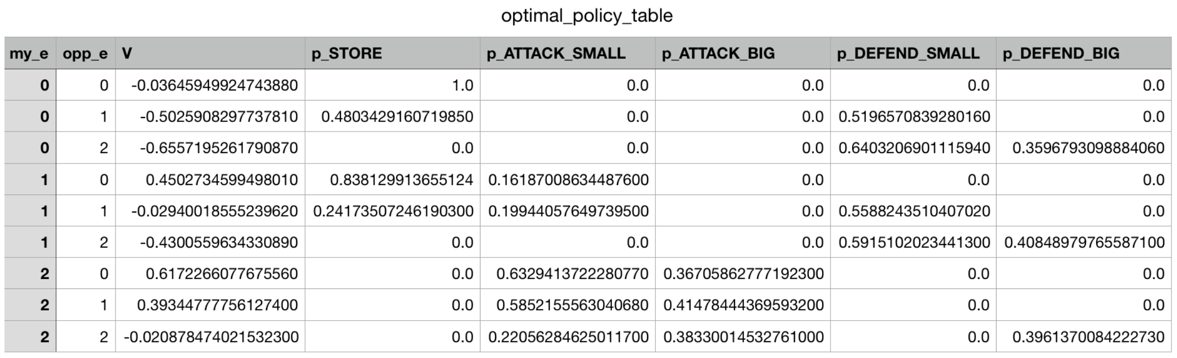
We studied a two-player zero-sum Energy Combat Game to evaluate how different reinforcement-learning approaches perform in a setting that requires mixed strategies. Our results show that independent Q-learning under self-play does not converge to the game-theoretic equilibrium: the resulting policy is exploitable, and a best-response agent with opponent modeling achieves a significantly higher win rate.

In contrast, the equilibrium computed through value iteration and the Bellman–Shapley operator provides a stable mixed strategy that cannot be exploited, as confirmed by best-response experiments. These findings highlight the limitations of naïve self-play in simultaneous-move games and the importance of equilibrium-aware algorithms.

Future work may explore extensions such as larger action spaces(energy level > 2, partial observability, or population-based training methods that better approximate Nash equilibria in more complex multi-agent

Algorithm 4 Optimal Equilibrium Computation via Value Iteration

```
1: Initialize $V(s) \leftarrow 0$ for all states $s$
2: repeat
3:     $V_{\text{old}} \leftarrow V$
4:     for each nonterminal state $s$ do
5:         Enumerate feasible actions $\mathcal{A}_0(s)$ and $\mathcal{A}_1(s)$
6:         For each pair $(a_0, a_1)$ compute
               $$Q(s, a_0, a_1) = r(s, a_0, a_1) + \gamma V(f(s, a_0, a_1))$$
7:         Form payoff matrix $Q_s$ over feasible actions
8:         Solve minimax problem
               $$V(s) = \max_{\pi_0} \min_{\pi_1} \pi_0^\top Q_s \pi_1$$
9:         Store corresponding optimal policies $\pi_0^\star(\cdot \mid s)$ and $\pi_1^\star(\cdot \mid s)$
10:    end for
11: until $\|V - V_{\text{old}}\|_\infty < \varepsilon$
12: return $V$, $\pi_0^\star$, $\pi_1^\star$
```


optimal_policy_table

my_e	opp_e	V	p_STORE	p_ATTACK_SMALL	p_ATTACK_BIG	p_DEFEND_SMALL	p_DEFEND_BIG
0	0	-0.03645949924743880	1.0	0.0	0.0	0.0	0.0
0	1	-0.5025908297737810	0.4803429160719850	0.0	0.0	0.5196570839280160	0.0
0	2	-0.6557195261790870	0.0	0.0	0.0	0.6403206901115940	0.3596793098884060
1	0	0.4502734599498010	0.838129913655124	0.16187008634487600	0.0	0.0	0.0
1	1	-0.02940018555239620	0.24173507246190300	0.19944057649739500	0.0	0.5588243510407020	0.0
1	2	-0.4300559634330890	0.0	0.0	0.0	0.5915102023441300	0.40848979765587100
2	0	0.6172266077675560	0.0	0.6329413722280770	0.36705862777192300	0.0	0.0
2	1	0.39344777756127400	0.0	0.5852155563040680	0.41478444369593200	0.0	0.0
2	2	-0.020878474021532300	0.0	0.22056284625011700	0.38330014532761000	0.0	0.3961370084222730

Figure 3: Optimal Policy Table.

environments.

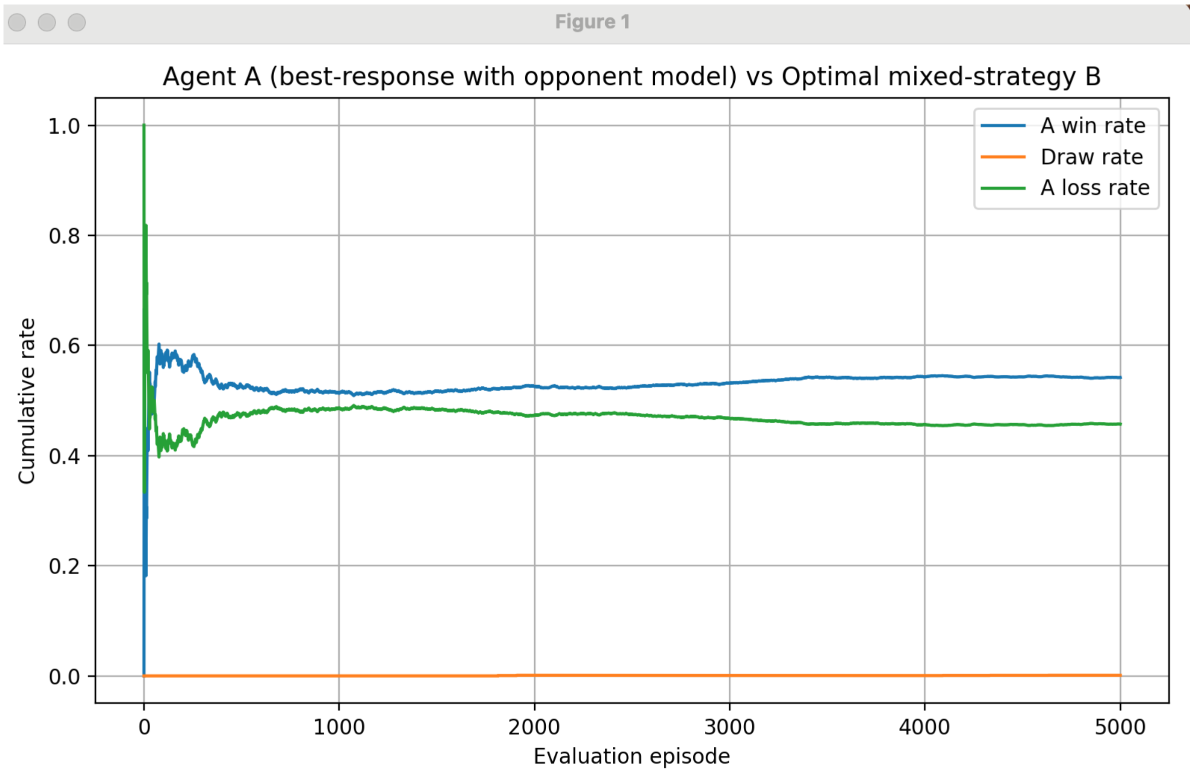


Figure 4: Opponent Modeling VS Optimal.