Mini-poker game

Abdellah BULAICH MEHAMDI, Mohamed mahmoud AHMED MALOUM, Zayed HERMA
Reinforcement Learning Project INF581

***Abstract*—This project focuses on developing reinforcement learning algorithms in a simplified version of a poker game. The approach begins with finding the best agent capable of defending, using a model where the agent always receives the opposing bet and decides whether to fold or call based on its hand and its estimation of the adversary's hand. Four different agents were developed using multi-arm bandit and Q-learning algorithms, and their approaches were compared to select the best defender agent. After identifying the best defender, the project moves on to developing an attack strategy that can be combined with the top-performing defender to create a complete agent capable of competing effectively against a human player.**

I. Introduction

Poker is a card game that involves betting and individual play, with the objective of having the best hand or forcing opponents to fold their hand. There are many variations of poker, but for this project, we will focus on a particular mini-poker version.

This is a 2-player game where each player is dealt a hand of 2 cards from a standard deck of 20 cards. The cards range from 7 to 10, with each card equal to its point value, and there is one Ace worth 11 points per suit. There are four suits in total. The game follows a standard betting structure where players take turns placing bets or raising the stakes. There is a maximum point value set at the beginning of the game, which is the same for both players.

The game continues until both players have reached the maximum point value of 10, which represents the maximum size of the pot. Each round begins with the dealer shuffling the cards and dealing two cards face-down to each player, with the first player to act changing in each subsequent round. In each round, players take turns betting and have three options:

- **Fold**: If a player feels that their hand is not strong enough to win the round or if they don't want to take risks, they can fold. Doing so will result in them losing points equal to the last amount of the bet they placed, or 1 point if they did not place a bet in the round.
- **Call**: If a player wants to stay in the game after several bets have been placed, they can call the current bet by matching the amount of the bet. At the end of the round, if a player has a lower hand sum, they lose points equal to the amount of the bet. If both players have the same hand sum, no points are lost.
- **Raise**: If a player believes they have a strong hand or wants to take risks, they can raise the bet by calling a larger amount of points. Doing so increases the size of the pot. The other player then has the opportunity to respond. The maximum size of the pot is 10 points.

Now that we have a good understanding of the principles of the mini-poker game, let's discuss some key aspects of our RL project, which is based on this game:

1) We aim to test reinforcement learning methods to develop an agent capable of playing mini-poker against a human or a created player. Due to the stochastic nature of the game, we will focus on learning a stochastic policy using methods covered during the course.
2) Learning a stochastic policy is a key aspect of our project. This involves understanding the behavior of the agent's adversary so that the agent can play according to the adversary's strategy. We have covered many reinforcement learning methods in class that seem promising for this type of game.
3) Our project presents several challenges. The first is adapting reinforcement learning methods to learn the adversary's strategy, the second is selecting the best method for the final version of the game, and the last is creating an agent that can compete against professional players.
4) Our mini-poker game is very different from a real poker game, so we have had difficulty finding similar examples. However, we have relied heavily on course materials and our understanding of these methods to adapt them to our game and achieve promising results.
5) In mini-poker, the agent must both attack and defend, so our approach is to first test the agent's defensive capabilities. We will design a good defender, similar to how one would approach playing blackjack, and then combine it with a strong attack strategy in a later stage. We will compare the performance of different agents in defense, select the best one, and combine it with a good attack strategy based on theoretical analysis and practical experiences.
6) The main results of the study indicate that the multi-arm bandit agents perform better and faster than the Q-learning algorithm when it comes to defense strategies. These findings suggest that the multi-arm bandit approach may be more effective than the Q-learning algorithm when designing agents for defense strategies. Additionally, the study found that the use of a beta distribution in the Thompson agent provides a more effective and robust approach for solving the multi-armed bandit problem in most scenarios, as compared to UCB1 and Exp3.

7) Due to time constraints and limitations, we were not able to try out all of our ideas or further manipulate our designed algorithms. If we had more time, we would have liked to:
   - Explore different algorithms: while the agents we developed and tested in this project performed reasonably well, there are many other reinforcement learning algorithms that could be explored to see if they perform better. For example, deep reinforcement learning algorithms such as Deep Q-Networks (DQN) and Actor-Critic methods could be implemented and tested.
   - Perform more extensive hyperparameter tuning: we performed some basic hyperparameter tuning in this project, but more extensive tuning could be done to find the best hyperparameters for each algorithm.
8) Github can be used to reproduce our results.

II. Background

Based on the approach explained in the introduction, we decided to begin by finding the best agent capable of defending. To do this, we used a model in which the agent always receives the opposing bet and then decides whether to **fold** or **call** based on its hand and its estimation of the adversary's hand. This model is illustrated in Figure 1 and corresponds to the following:

- The **agent**'s **state** at each round of the game consists of the sum of the agent's hand (which ranges from 14 to 22) and the opponent's bet (which ranges from 1 to 10). The agent has access to both pieces of information. We can denote the current state at round t of the game as $S_t = (\text{agent's hand sum}, \text{opponent's bet})$.
- The **environment** consists of a deck of cards and the adversary's strategy. The agent is trying to learn this strategy, which is the main part of the environment.
- Defining the **reward** independently of the method used to learn the opponent's strategy can be ambiguous. However, we can say that the agent's reward information is based on the number of points lost and the sum of the opponent's hand. This sum can be seen as an approximation of the adversary's hand.
- The agent's **actions** are dependent on the opponent's bet and the agent can either choose to fold or call. If the agent chooses to fold, he loses one point. However, if the agent chooses to call, the player with the lowest hand sum loses a number of points equal to the opponent's bet.

We began by developing this model, which enabled us to design four different agents using only the course material. Our goal was to compare these agents and select the best one capable of defending against the adversary's stochasticity and intelligence. The first three agents were based on multi-arm bandit algorithms, specifically, stochastic and adversarial bandits algorithms presented in Lecture 3 [1]: **UCB1**, **Thompson's sampling** and **Exp3**. Our task was to adapt these algorithms to our environment and game rules to create a strong defender agent.

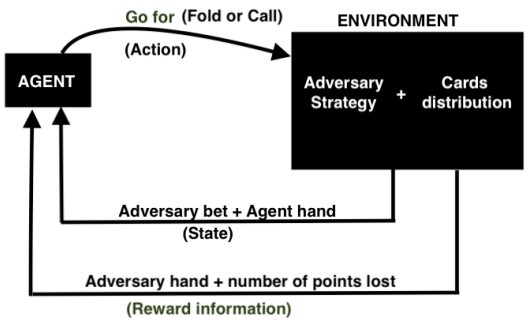

Fig. 1. Illustration of the interaction between the agent and its environment.

For these three agents, the objective was to estimate an upper-bound s of the adversary's hand sum from his bet b. To do this, the agent would sample an upper-bound from a distribution estimated based on the adversary's bets and the agent's own hand sum. Each agent used a different approach to estimate and update this distribution (probability, expected reward, etc.) and sample an instance from it. We will describe each agent's approach in detail in the next section.

Finally, the fourth defender agent was based on the Q-learning algorithm presented in Lecture 5 [2]. We used the following equation for the incremental update in the algorithm:

$$Q(S, A) \leftarrow Q(S, A) + lr * (R - Q(S, A)) \tag{1}$$

Where S is the state of the game which includes the sum of the agent's cards and the bet of the adversary, A denotes the chosen action, lr is a learning rate, and R represents the reward when the cards are revealed. Note that here, we do not have another state because in this particular case, there is no additional state to consider as the cards are returned to the deck and shuffled after each round.

Two methods for selecting actions were implemented in the environment. The first approach is **ϵ-greedy**, while the second approach is **softmax**, which involves a weighted probability distribution based on the learned rewards.

Once we had analyzed the defensive strategies using the four previously developed agents, our attention turned to creating effective attack strategies. However, due to time constraints, we decided to first explore a dominant attack strategy that does not necessarily require reinforcement learning. Our goal was to find an attack strategy that could be combined with a strong defensive agent to create a complete player. We plan to explore more complex attack strategies at a later stage.

One reason for prioritizing defense over attack in this type of game is that defensive strategies are typically more complex and challenging compared to attack strategies. This is because defense requires a thorough analysis of the

opponent's moves and hand, as well as the ability to make optimal decisions while managing risk. In contrast, attack strategies often involve making aggressive moves that may not carry as much risk as defensive ones, depending on the strength of the player's hand. If the player has a strong hand, an aggressive attack strategy can be less risky than playing defensively, as it allows the player to maximize his gains. However, if the player's hand is weak, an aggressive approach can be very risky and lead to significant losses. Therefore, the choice of an attack strategy largely depends on the player's assessment of their hand's strength relative to the opponent's hand.

Therefore, creating a strong defensive agent is crucial for achieving success in this game. Without a solid defense, the player's chances of winning may be significantly reduced, as even a small mistake in defense can lead to significant losses. Developing a strong attack strategy is also important, but it often requires less effort than developing a strong defense.

III. Methodology/Approach

This section presents in detail the four defense strategy agents, followed by a description of the experiments conducted to compare their performance. Additionally, a brief overview of the attack strategy exploration is provided.

A. Defense strategies

*1) **UCB1 agent**:* The UCB1 algorithm was used to design this agent, which balances exploring and exploiting the adversary's strategy [1]. When the adversary bets for $b \in 1, 2, ..., 10$, the agent notes $E^b = [E^b_{14}, E^b_{15}, ..., E^b_{22}]$, where E^b_s is the expected reward when the agent decides that the sum of the opponent's hands is less than or equal to **s**, and given that the adversary's bet is **b**. The agent's matrix of rewards is denoted as $(E^b_s)_{b,s}$ where:

$$E^b_s = \sum_{t^b_s=1}^{N^b_s} \frac{R^b_{s,t^b_s}}{N^b_s} \tag{2}$$

With N^b_s represents the number of times the agent has estimated that the sum of the opponent's hands is less than or equal to **s**, given that the bet is **b** and R^b_{s,t^b_s} represents the reward of the t^b_s th decision. The reward is defined in each round as:

$$R = \frac{\text{points lost by the adversary - points lost by the agent}}{10} \tag{3}$$

Thus, the reward **R** is a random variable in $\{-1, -0.9, -0.8, ..., 0.8, 0.9, 1\}$.

The UCB1 equation used to estimate an upper-bound of the adversary's sum hand, given the bet **b** is:

$$\underset{s}{\operatorname{argmax}}(E^b_s + \frac{\sqrt{\alpha \log(N_b)}}{N^b_s}) \tag{4}$$

With N_b represents the number of times the adversary has chosen to bet for **b**, and α is a parameter to be adjusted [1].

Refer to algorithm 1 in the appendix for more illustration.

*2) **Thompson agent**:* This agent utilizes the Thompson's sampling method principle [1] to sample and estimate an upper-bound of the opponent's hand sum. When the opponent bets for $b \in \{1, 2, ..., 10\}$, the goal is to estimate the distribution of $S_b \in \{14, 15, ..., 22\}$, which represents the sum of the opponent's hand. Based on this estimation and its own hand, the agent can decide to fold or call.

In order to adapt the method to our model, we first assume that the opponent bets b based on its hand sum S. We then suppose that there exists an upper-bound estimate S_b of the hand sum that satisfies:

$$S_b = 8Y_b + 14 \tag{5}$$

where $Y_b \sim \beta(a_b, b_b)$ is a random variable following the beta distribution with parameters (a_b, b_b). The coefficients 8 and 14 were chosen to map S_b in the range $[14, 22]$.

Using this assumption, the goal is to adjust the parameters (a_b, b_b) for each opponent's bet b in order to estimate the best upper-bound S_b of its hand sum.

Refer to algorithm 2 in the appendix for more details.

*3) **Exp3 agent**:* This agent is designed based on the Exp3 algorithm principle [1]. The aim is to estimate, for each opponent bet b, a histogram of the probability distribution P_b, giving for each $s \in \{14, 15, ..., 22\}$ the probability that the opponent's hand sum is less than or equal to s. If the opponent bets for $b \in 1, 2, ..., 10$, we denote $P_b = [P_{b,14}, P_{b,15}, ..., P_{b,22}]$. So the agent's matrix of probability distribution is $(P_{b,s})_{b,s}$.

We initialize the probability distribution P_b as uniform for each b. Then, for a given opponent's bet b, we sample an upper-bound s from the distribution P_b. Based on this upper-bound, we take an action and observe a reward $R_{b,s}$, which we use to update the cumulative reward $\hat{X}_{b,s} = \sum_{t=1}^{N_{b,s}} R_{b,s,t}$, where $N_{b,s}$ is the number of times we sample s knowing that the bet is b. The reward $R_{b,s}$ is defined exactly as for the UCB1 agent. Then, we update the distribution P_b using the exponential weighting formula:

$$P_{b,s} = \frac{\exp(\eta \hat{X}_{b,s})}{\sum s' = 14^{22} \exp(\eta \hat{X}_{b,s'})}, \tag{6}$$

where η is a learning rate that characterizes the degree to which the agent exploits aggressively or explores more. Note that $\hat{X}_{b,s} = 0$ for the initialization (uniform probability distribution).

*4) **Q-Learning agent**:* This agent is designed based on the Q-learning algorithm principle [2]. The Q-table is a dictionary where the keys represent states, and the values are dictionaries of actions and their corresponding accumulated rewards. The Q-table is initially empty, and it is updated during play and

exploration using equation (1). The agent selects an action from the Q-table using either the epsilon-greedy or softmax method:

- The ϵ-greedy approach involves randomly choosing to either fold or call with a probability of ϵ and selecting the action with the highest accumulated reward with probability $1 - \epsilon$.
- The softmax approach involves using the learned Q-table to create a probability distribution:

$$P_{S,A} = \frac{e^{Q[S,A]}}{\sum_{A'} e^{Q[S,A']}} \tag{7}$$

  Note that a temperature parameter can also be added, as in equation (5), to adjust the standard deviation of the distribution and make it more or less uniform.

The epsilon-greedy approach is more intuitive and natural since it balances exploration and exploitation by randomly selecting actions with a probability of epsilon. However, once exploration is turned off, the agent becomes fully deterministic and can be exploited by opponents who can recognize the pattern.

In contrast, the softmax approach is less predictable, as it uses probability distributions to choose actions. This can be an advantage because it can make the agent more difficult to predict, but it can also be a disadvantage if the agent makes unexpected or illogical moves that can deceive human players who are not paying close attention.

B. Comparison of agents :

In order to compare the previously designed agents, they were tested against two different attackers.

The first one is a **deterministic attacker** whose bet b is a deterministic function of its hand sum s: $b = s - 12$.

The second one is a **random mean attacker** whose bet b is defined as:

$$b \sim \begin{cases} U(\{1, ..., 5\}) & \text{if s} < 18 \\ U(\{6, ..., 10\}) & \text{else} \end{cases} \tag{8}$$

where U denotes the uniform distribution.

The performance of each agent is evaluated based on three criteria over several games ($N_G \simeq 5000$ in our experiments) against each attacker.

The first criterion is the ability of the agent to learn precisely the opponent's (attacker) strategy over a game.

The second criterion is the expected reward R_g for the agent at the end of a game, which is defined as:

$$R_g = \frac{\text{NptsAgent} - \text{NptsAdverser}}{\text{Worth}_0} \tag{9}$$

where Worth_0 is the initial equal worth (of points) for each player (constant at the beginning of each game).

The last criterion is the number of games won out of the total, and the mean number of rounds necessary to win a game (for more details see section IV).

C. Exploring attack strategy :

All of the previously developed agents have the ability to play as an attacker, but they may not perform as well as they do in defense. This is because in attack, they have access to less information (only their hand), which can limit their decision-making abilities. Therefore, for the attack strategy that will be presented in the next time (for the final report), we aim to develop a strong and effective approach that, when combined with a good defensive agent, will create a complete player capable of performing like a human. The goal is to find an attack strategy that can compensate for the limitations of the agent's information and still enable it to make optimal decisions while minimizing risks.

IV. Results and Discussion

The results of comparison explained in III.B are presented below.

A. First criterion: learn opponent strategy

*1) **Thompson agent**:* This section presents the performance of the Thompson agent against two adversaries: a deterministic attacker and a random mean adversary. Figure 2 displays the results against the deterministic attacker, while Figure 3 shows the results against the random mean adversary. Our analysis indicates that the agent accurately estimates a high upper bound for the sum of the adversary's hand, which is located near the top of the distribution in both cases. We represent the estimated distribution of the adversary's hand in a certain color corresponding to their action. For example, in Figure 3, we can observe the strategy developed by the agent, which involves calling only if the agent's hand sum is greater than 18 when the adversary bets between 6 and 10, and folding otherwise.

*2) **Exp3 agent**:* Here we present a probability matrix that illustrates the strategy learned by the Exp3 agent against two attackers, as shown in Figure 4 for the deterministic attacker and Figure 5 for the random mean adversary. For instance, in Figure 4, the strategy advises the agent to refrain from calling if the sum of its hand is less than 20 when the adversary bets 8...

*3) **UCB1 agent**:* Here, we present the expected reward matrix that displays the strategy learned by the UCB1 agent against two attackers, as shown in Figure 6 against the deterministic attacker and Figure 7 against the random mean adversary. For instance, if the random mean adversary bets 6 or more, the agent's most rewarding strategy is to call only if the sum of the agent's hand is 18 or more...

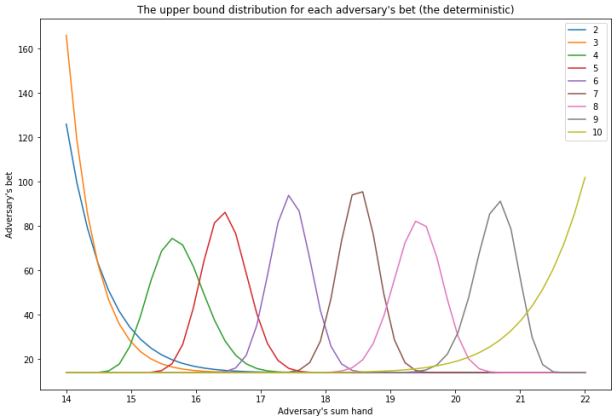

Fig. 2. Estimated distribution of determenstic adversary hand in function of his action

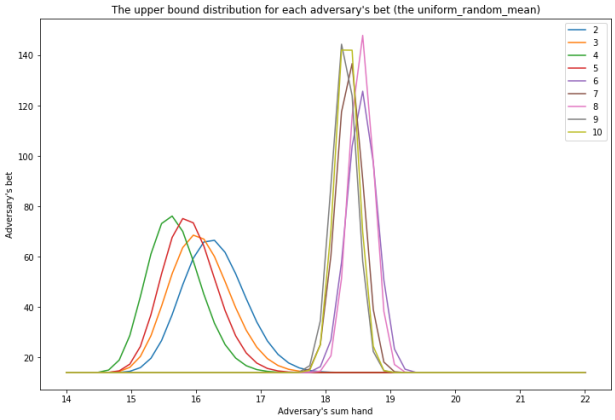

Fig. 3. Estimated distribution of random mean adversary hand in function of his action

B. Second and Third criterion

We present the performance results of our agents in the tables I, II, and III, based on various criteria over 5000 games.

Agent	Adverser	1	2	3	4
UCB	D	-0.14	1602	51	47
	RM	-0.16	1455	50	48
Thompson	D	0.20	4216	54	60
	RM	0.06	3106	56	60
Exp3	D	-0.12	1778	51	48
	RM	-0.14	1635	51	49
TD-softmax	D	-0.47	187	56	44
	RM	-0.49	138	55	44
TD-greedy	D	-0.46	202	55	45
	RM	-0.49	156	55	44

TABLE I
INITIAL NUMBER OF POINTS = 100, NUMBER OF GAMES = 5000

where D stands for deterministic adverser and RM stands for a randomized mean opponent. And:

- 1 represents the average of the expected reward over the games $R_g \in [-1, 1]$ defined by (9).
- 2 represents the number of games won out of the total.

Agent	Adverser	1	2	3	4
UCB	D	0.42	5000	560	nan
	RM	0.27	4999	644	786
Thompson	D	0.57	5000	518	nan
	RM	0.4	5000	569	nan
Exp3	D	0.28	4996	558	628
	RM	0.15	4702	626	667
TD-softmax	D	-0.41	6	631	514
	RM	-0.49	0	nan	497
TD-greedy	D	-0.18	750	663	594
	RM	-0.38	21	676	561

TABLE II
INITIAL NUMBER OF POINTS = 1000, NUMBER OF GAMES = 5000

Agent	Adverser	1	2	3	4
UCB	D	0.69	5000	5131	nan
	RM	0.5	5000	6018	nan
Thompson	D	0.71	5000	4977	nan
	RM	0.52	5000	5309	nan
Exp3	D	0.62	5000	5115	nan
	RM	0.49	5000	5860	nan
TD-softmax	D	0.15	5000	6470	nan
	RM	-0.06	280	7018	6896
TD-greedy	D	0.36	5000	6042	nan
	RM	0.14	5000	6968	nan

TABLE III
INITIAL NUMBER OF POINTS = 10000, NUMBER OF GAMES = 5000

- 3 represents the average number of necessary rounds to win a game.
- 4 represents the average number of necessary rounds to be defeated!

Our findings demonstrate that the multi-arm bandit agents outperform both versions of the Q-learning algorithm in terms of defense strategies, while also being faster. For instance, even with an initial score of 1000, the Q-learning algorithm fails to acquire an accurate understanding of the opponent's strategy and loses, although it may improve its speed with fine-tuning. Additionally, our analysis indicates that the use of the beta distribution in the Thompson agent provides a more effective and robust approach for solving the multi-armed bandit problem, surpassing UCB1 and Exp3 in most scenarios.

V. CONCLUSIONS

This project demonstrates the application of reinforcement learning algorithms in a particular version of the poker game, where the focus is on developing both defense and attack strategies. The project began by finding the best defender agent using a model where the agent always receives the opposing bet and decides whether to fold or call. Four different agents were developed and compared, with Thompson being the best approach for defense. The project's next stage will involve developing an attack strategy that can be combined with the selected defensive agent to create a complete player capable of calling, folding, and raising. Overall, the project provides valuable insights into the application of reinforcement learning algorithms in poker and highlights the importance of selecting the appropriate approach for the specific task at hand.

REFERENCES

[1] Read Lecture III - Bandits. In *INF581 Advanced Machine Learning and Autonomous Agents*, 2023.

[2] Read Lecture V - Reinforcement Learning II. In *INF581 Advanced Machine Learning and Autonomous Agents*, 2023.

APPENDIX

Algorithm 1 UCB1-poker Algorithm

$\{E_s^b\}_{b,s} \leftarrow \{0\}_{b,s}$ ▷ the matrix of the expected rewards
$\alpha \leftarrow \alpha_{optimal}$
while No one has lost **do**
  $ASH \leftarrow$ the sum of the agent's hand
  $b \leftarrow$ the adversary bet
  if $min(\{N_s^b\}_s)$ ==0 **then** ▷ We need to explore
    $s^* \leftarrow \text{argmax}_s(-E_s^b)$
    $N_{s^*}^b \leftarrow 1$
  else if $min(\{N_s^b\}_s)$!=0 **then**
    $s^* \leftarrow \text{argmax}_s(E_s^b + \frac{\sqrt{\alpha \log(N_b)}}{N_s^b})$
    $N_{s^*}^b \leftarrow 1 + N_{s^*}^b$
  end if
  if $ASH \geq s^*$ **then**
    $decision \leftarrow Call$
  else if $ASH < s^*$ **then**
    $decision \leftarrow Fold$
  end if
  Update E_s^b according to the reward information of the round and using the formula (2).
end while

Algorithm 2 Thompson-poker Algorithm

$S_b \leftarrow$ with $(a_b = 1, b_b = 1)$ for each b
$\alpha \leftarrow \alpha_{optimal}$ ▷ To regularize the way the distribution changes according to the adversary strategy
while No one has lost **do**
  $ASH \leftarrow$ the sum of the agent's hand
  $b \leftarrow$ the adversary bet
  $s^* \leftarrow$ sample from S_b
  if $ASH \geq s^*$ **then**
    $decision \leftarrow Call$
  else if $ASH < s^*$ **then**
    $decision \leftarrow Fold$
  end if

  if Agent call, but it was better to fold **then**
    $a_b \leftarrow a_b + \alpha \times b$ ▷ Shift the S_b to the right to get bigger upper-bound estimation
  else if Agent fold, but it was better to call **then**
    $b_b \leftarrow b_b + \alpha \times b$ ▷ Shift the S_b to the left to get smaller upper-bound estimation
  end if

end while

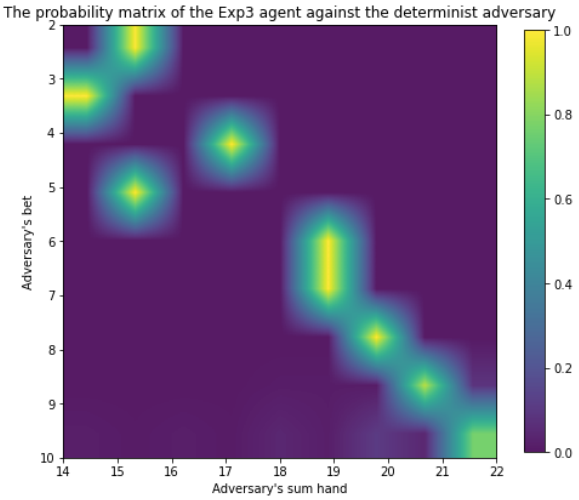

Fig. 4. The agent's probability matrix, learned when playing against the deterministic opponent

Fig. 5. The agent's probability matrix, learned when playing against the random mean opponent

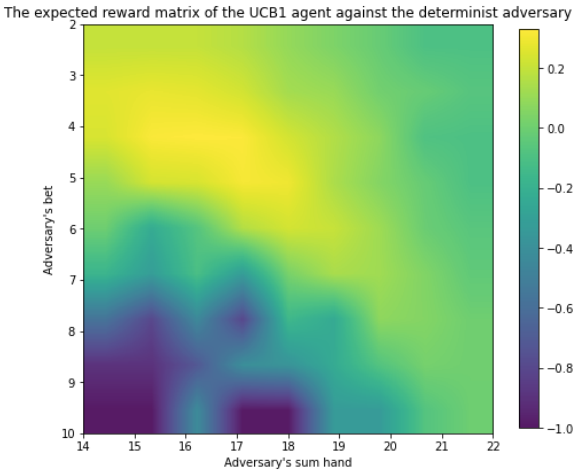

Fig. 6. The agent's expected reward matrix, learned when playing against the deterministic opponent

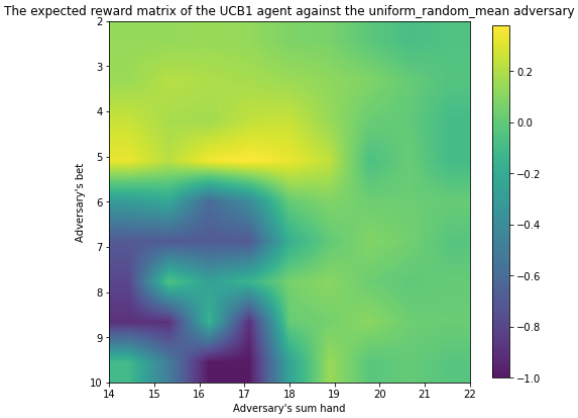

Fig. 7. The agent's expected reward matrix, learned when playing against the random mean opponent