The Bidding Elimination Game

Caleb Chiam
Computer Science Department
Stanford University
Stanford, California
calebc96@stanford.edu

Jonathan Li
Computer Science Department
Stanford University
Stanford, California
jlli@stanford.edu

Alan Lee
Computer Science Department
Stanford University
Stanford, California
yutselee@stanford.edu

***Abstract*—We examine the Bidding Elimination game, a multiplayer zero-sum imperfect information game involving placing bets with a limited budget in successive rounds. The game can be re-scaled to varying levels of difficulty by setting different numbers of player and different starting budgets. We perform analysis of pure and mixed Nash equilibrium strategies in small-scale versions of the game, implement three solution methods (parameter estimation & optimization, deep Q-learning, and value iteration) to tackle small and large versions of the game, and analyze our results. We find that all three approaches perform well in the small version of the game, and demonstrate promising results for deep Q-learning and value iteration for larger games.**

I. Introduction

This paper studies the Bidding Elimination game – a multiplayer game played by three or more players. The description of the game is as follows:

Suppose the game starts with n players. Each player is given m coins, where m is a positive integer. The exact amount is unimportant, as long as it is finite and not trivially small. The game proceeds by rounds of bidding, where one player is eliminated in each round. This means that that there is a maximum of $n-1$ rounds. The objective of the game is to be the last player standing.

A. How a round works

1) Every player places a bid from his/her coins. This permanently deducts the bid from the coins the player owns. The bids are placed secretly.
2) Once the bids are placed, a non-player host collects the bids and then reveals the lowest bid that was made.
3) The player who made that bid is eliminated, and this concludes the round. If multiple players made the lowest bid, one of them is randomly eliminated.
4) The lowest bid amount is revealed at the end of each round, but all other bid amounts are kept secret.

The rounds repeat until only one player is left. This player is the winner.

B. Key strategies in playing the game

We offer the following observations and intuitions about strategies in the game:

- A player should want to limit how many coins is spent on bids in each round, since he/she only needs to beat the lowest bidder in every round.
- At the same time, a player would not want to bid too little, because they risk elimination.
- Bidding 0 coins in some rounds is not necessarily a bad strategy. If other players bid 0 coins, there is a chance that the player can avoid the random elimination, which may allow the player to have the most number of coins in subsequent rounds.

For the purposes of this paper, we will analyze the 3-player game in depth before tackling the general n-player game, where $n \geq 3$.

II. Background

A. Game Theory

Traditionally, Nash equilibrium strategies have been an important solution concept in game theory. In two-player zero-sum games, a pair of strategies (S, T), where Player 1 chooses a strategy S and Player 2 chooses a strategy T, is a Nash equilibrium if S is a best response to T, and T is a best response to S. In this state of equilibrium, neither player has an incentive to deviate from their strategies [1].

That being said, in the three-player or non-zero-sum setting, it is an open problem as to whether Nash equilibrium strategies are successful. The success of playing a Nash equilibrium strategy depends on the strategy chosen by other players, and players following different Nash equilibrium strategies may consequently end up obtaining the worst possible payoffs. That being said, past work in large multiplayer games have demonstrated that the strongest existing agents are based on approaches that attempt to approximate Nash equilibrium strategies [2]. An analysis of the Nash equilibria of the game may thus be a fruitful line of inquiry.

B. Reinforcement Learning

Reinforcement Learning (RL) techniques are a family of learning algorithms for solving MDPs. They are typically defined in terms of an agent acting in an environment, where the transition and reward dynamics are defined by the associated MDP. While large-scale imperfect information games are typically solved using other methods [3], RL algorithms can be adapted to this setting as well.

III. POMDP Formulation

Recall that a POMDP is a 7-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, R, O, \gamma \rangle$. We define the POMDP formulation of a Bidding Elimination game with n players and m coins per player as follows:

1) $\mathcal{S} = \{(\mathcal{B}, \mathcal{H}) \,|\, \mathcal{B} \in \mathcal{P}([0, m]), |\mathcal{B}| \leq n\}$, where each $b_i \in \mathcal{B}$ represents the current budgets of each player, and $\mathcal{H}$ represents the history of minimum bids so far.
2) $\mathcal{A} = \{0, ..., m\}$ representing the possible bets each player can make. Note that some bets are invalid depending on the current state; in particular, you cannot place a bet that is more than your current budget
3) $\mathcal{O} = \{(b, \mathcal{H}) \,|\, b \in [0, m]\}$ where b represents the current budget of the player and $\mathcal{H}$ is the same bidding history as the current state $\mathcal{S}$.
4) $T(s'|s, a)$ is the probability of transitioning to a new state s' given the current budgets s and a player action a. These probabilities are determined by the strategies of the opposing players that are yet to be eliminated.
5) $R(s, a) = \begin{cases} -1 & a \text{ causes the player to be eliminated} \\ 0 & \text{the player survives the current round} \\ 1 & \text{all other players are eliminated} \end{cases}$
6) $O(o \,|\, a, s')$ is deterministic, returning the full history $\langle$ and the budget b corresponding to the player from the new state s'.
7) $\gamma \in [0, 1]$ is the discount factor.

We parameterize each instantiation of the Bidding Elimination game with a tuple (n, m, P), where n and m are the number of players and starting coins per player respectively, and $P = \{p_j\}_{j=1}^{n-1}$ represents the betting strategies of the $n - 1$ opponent players, where each $p_j(i \,|\, b, \mathcal{H})$ is a discrete conditional distribution over $i \in [0, m]$.

IV. Evaluation metrics

In the 3-player game, the game effectively has only one "real" round of bidding, namely the first round. After a player is eliminated in the first round, the remaining players will bid all their remaining coins in the second round. In effect, the bids in the second round are fully determined by the bids in the first round.

The strategy of a player can then be characterized by some discrete distribution $P(X = i)$, where X is the bid of the player in the first round and i is an integer that lies within the range $[0, m]$, and $\sum_{i=0}^{m} P(X = i) = 1$.

For the 3-player game, we evaluate the different agents by their performance in four evaluation scenarios. Let m be the number of coins allotted to each player.

1) $m = 2$ and opponents' strategies are static
2) $m = 2$ and opponents' strategies change roughly every 50 games
3) $m = 10$ and opponents' strategies are static
4) $m = 10$ and opponents' strategies change roughly every 50 games

For each agent we developed, we ran this evaluation suite of four scenarios (where each is run for 1000 games) 100 times to obtain a distribution of the agents' performance (% win rate out of 1000 games) in each scenario.

The high-level motivation for these scenarios is that we want to establish that our developed agents can perform well in a setting where opponents play a consistent strategy, as well as in a more realistic setting where opponents may change their strategies every once in a while.

V. Approaches and results

In this section, we discuss the various approaches we tried and their accompanying results, which, in some cases, further refined the approaches we were taking.

We began our analysis with simpler approaches that mainly tackled the 3-player game case with variable numbers of coins, before moving on to approaches that can handle an arbitrary n players.

A. Nash Equilibria

1) Expected payoffs: As discussed in Section V, we observe that the 3-player game is simple enough to be amenable to a game-theoretic analysis because there is only one 'real' round of bidding. With $m = 2$ coins, we can represent the expected payoffs for the players in Fig. 1. For convenience, we will refer to Players 1, 2, and 3 as P1, P2, and P3.

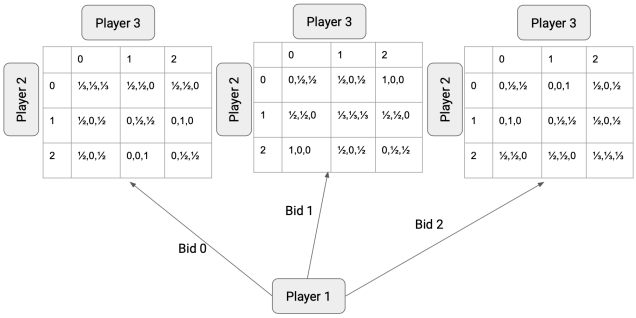


Fig. 1: Payoff matrices in a 3-player game, where each cell holds the expected payoffs for P1, P2, P3 for the given configuration of bids.

To compute the expected payoffs for any given configuration of bids, we treat winning the game as having a payoff as 1 and not winning as having a payoff of 0 (note that this is slightly different from the reward model in the POMDP formulation). Most of the expected payoffs for individual players are then fractional amounts because of the randomness in the elimination process when there are tied lowest bidders. For example, if P1 bids 0, and P2 and P3 both bid 2, the expected payoffs are (0, $\frac{1}{2}$, $\frac{1}{2}$). P1 is guaranteed to be eliminated in the first round and thus has an expected payoff of 0. P2 and P3 have the same number of remaining coins (i.e. 0) in the second round and have the same chance of winning, hence their expected payoff is $\frac{1}{2}$.

2) Pure strategy Nash equilibria: By inspection, we can identify three pure strategy Nash equilibria, namely when players all bid 0, 1, or 2 coins. Notice that if two players bid the same number of coins (be it 0, 1, or 2), then the third player only has a possibility of winning if he bids the same amount. This is because:

- If he bids a smaller amount, he is guaranteed elimination in the first round.

- If he bids a larger amount, he will survive elimination in the first round, but will have fewer coins than the other remaining player in the second round and is guaranteed to lose as well.

Bidding the same amount is the only way the third player has a chance of winning the game at all. Since the game is symmetric, the same analysis holds for the other players as well. If two other players are bidding some x coins in the first round, the remaining player has no incentive to deviate from the strategy of always bidding x coins in the first round. This holds for any m coins in the 3-player game, where $m \geq 2$.

3) Mixed strategy Nash equilibria: The next question we investigate is if any mixed strategy Nash equilibria exist. In such an equilibrium, the expected utility of any of the player's actions (i.e. bids) is the same. Given that the game is symmetric, we should expect that at equilibrium, the strategies employed by all 3 players are identical.

So, for the 3-player game, suppose the mixed strategy at Nash equilibrium is given by:

$$P(X = i) = \begin{cases} a, & \text{if } i = 0 \\ b, & \text{if } i = 1 \\ 1 - a - b, & \text{if } i = 2 \end{cases}$$

Using the standard method for computing mixed strategy Nash equilibria, we compute the expected utilities for each of the bidding actions. The player should be agnostic between these choices, i.e. the expected utilities should be the same. Thus, for a given bid amount by P1, we can simply calculate the expected payoff for P1 given that P2 and P3 bid according to the mixed strategy.

More formally, let $EU_P(x)$ represent the expected utility for a specified player P making a bid of x. Let the bid amounts of P2 and P3 as y and z respectively. Let $PO_P(x, y, z)$ be a function representing the expected payoff for player P given a configuration of bids (x, y, z) representing P1's bid, P2's bid, and P3's bid respectively. Finally, let P^x represent the event that Player P bids x.

$$EU_{P1}(\text{bid } x) = \sum_{y=0}^{m} \sum_{z=0}^{m} PO_{P1}(x, y, z) \cdot Pr(P2^y, P3^z)$$

This gives us the following expected utilities:

$$\begin{aligned} EU_{P1}(\text{bid } 0) &= -\frac{2}{3}a^2 + a \\ EU_{P1}(\text{bid } 1) &= -2a^2 - \frac{2}{3}b^2 - 2ab + 2a + b \\ EU_{P1}(\text{bid } 2) &= (a + b)(1 - a - b) + \frac{1}{3}(1 - a - b)^2 \end{aligned} \tag{1}$$

Solving the equality:

$$EU_{P1}(\text{bid } 0) = EU_{P1}(\text{bid } 1) = EU_{P1}(\text{bid } 2)$$

We find that there are no solutions where $0 \leq a, b \leq 1$ and $a + b \leq 1$. This implies that no mixed strategy Nash equilibria exist. We also wrote a mixed strategy Nash equilibria solver and found that this seems to hold true for any $m \geq 3$.

B. Parameter estimation and optimization

Instead of attempting to find an optimal strategy via Nash equilibria, we shifted our approach by formulating the game as an optimization problem. We observed that if P1 knows the strategies of P2 and P3 (and they are fixed), it can compute the total expected payoff as a function of its (P1's) strategy alone. P1 then simply needs to optimize its strategy to maximize this total expected payoff.

Identifying an optimal strategy then becomes a two-stage process:

1) *Parameter estimation*: Estimating the opponents' strategies based on their behavior in a series of games, aka learning the parameters of the discrete distributions that describe the opponents' strategies
2) *Optimization*: Calculating the optimal player strategy based on the estimated opponents' strategies

We create an *OptimAgent* that, when initialized for a new set of games, with beliefs that its opponents (P2 and P3) each follow a uniform strategy, where the probabilities of bidding any valid amount is equal.

At the end of each game, from the known state of the game (i.e. the lowest bids and eliminations in each round) and its own bids, it deduces the possible bids P2 and P3 made during the game (from enumeration of all possible game states), and uses this information to update its beliefs about P2's and P3's strategies.

The belief-update step is essentially done through maximum likelihood estimation (MLE). MLE is typically used when the counts in a distribution are exactly known, but since the exact actions taken by the opponents is typically unknown even at the end of the game, we give each possible action by the opponents a fractional count $\frac{1}{|\text{possible actions}|}$. For example, if P2 is deduced to have either bidded 1 or 2 in the first round, we represent the action they took as the vector $\boldsymbol{x} = [0, \frac{1}{2}, \frac{1}{2}]$. If P2 is deduced to have bidded 1, then the action is represented by the vector $\boldsymbol{x} = [0, 1, 0]$.

If we take $\hat{\boldsymbol{\theta}}_i = [a_i, b_i, c_i]$ to be *OptimAgent*'s belief about Player *i*'s strategy, then the belief-update step is simply an incremental estimation step:

$$\hat{\boldsymbol{\theta}}_i \leftarrow \hat{\boldsymbol{\theta}}_i + \alpha(\boldsymbol{x} - \hat{\boldsymbol{\theta}}_i) \tag{2}$$

where α is the learning / update rate. We set $\alpha = 0.01$ in our implementation. $\hat{\boldsymbol{\theta}}_i$ is also normalized after each update step so that the probabilities sum to 1. As shown in Fig 2, this works reasonably well, as the parameter estimates quickly approach the true parameter values.

Next, given its estimation of the opponents' strategies, *OptimAgent* computes an optimal strategy $s = [a_1, b_1, c_1]$ (representing the probabilities of bidding 0, 1, and 2 respectively) to maximize its payoff. For a given strategy for *OptimAgent*,

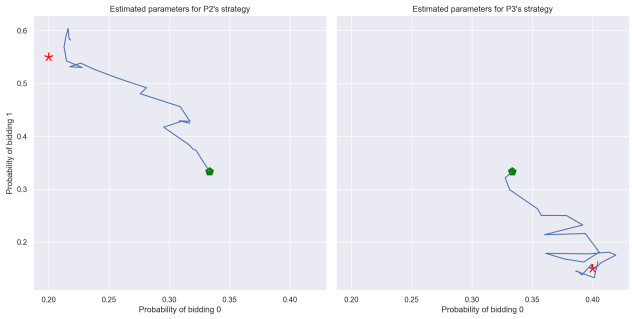


Fig. 2: Trajectory of parameter estimation when P2's strategy is $[0.20, 0.55, 0.15]$ and P3's strategy is $[0.40, 0.15, 0.45]$. These are the beliefs, aka parameter estimates, of *OptimAgent* from 500 games against P2 and P3 (sampled every 20 games), who both have static strategies.

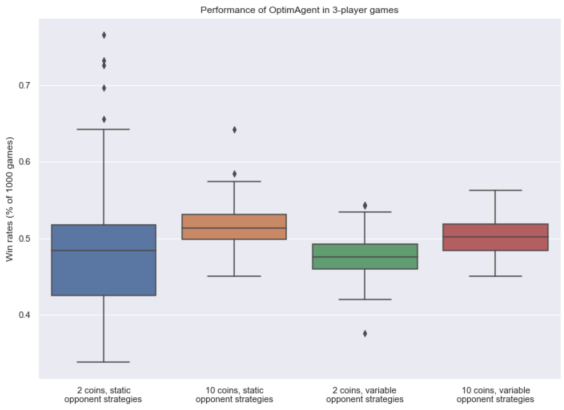


Fig. 3: Win rates of *OptimAgent* - a parameter learning + optimization agent

we can compute its total expected payoff W, where:

$$\begin{aligned} W &= \sum_{x,y,z \in [0,2]^3} PO_{P1}(x,y,z) Pr(P1^x, P2^y, \text{P3}^z) \\ &= \sum_{x,y,z \in [0,2]^3} PO_{P1}(x,y,z) Pr(P1^x) Pr(P2^y) Pr(\text{P3}^z) \end{aligned} \tag{3}$$

We optimized W with respect to s subject to the constraint $a_1 + b_1 + c_1 = 1$ using SciPy's $minimize$ function.

Thus, the typical game loop for *OptimAgent* proceeds as follows:

- Calculate an optimal strategy s given estimates of opponents' strategies, then sample from s (which defines a discrete distribution) to decide on the first bid amount
- At the end of the game, infer the possible bid amounts of P2 and P3, then update beliefs about their strategies

Because $OptimAgent$ updates its beliefs after every game, it effectively has an online learning capability that allows it to adapt to opponents changing their strategies at any point during a series of games (e.g. as in scenarios 2 and 4 in our evaluation suite).

As shown in Figure 3, *OptimAgent* performs well in all our evaluation scenarios, winning roughly 50% of the time in each scenario. It performs slightly worse when the opponents vary their strategies, which is understandable since *OptimAgent* will take a few games to learn the new strategies of its opponents.

We also observe that its performance in the *2 coins, static strategies* scenario has especially high variance. Because this scenario involves only a few opponents with static strategies, the performance of *OptimAgent* varies significantly depending on 'luck of the draw'. For example, in the best case, P2 and P3 could have strategies like 'bid 0 usually' and 'bid 2 usually', allowing P1 to win a majority of the time by always betting 1. In the worst case, P2 and P3 could both bid the same amount most of the time. This corresponds closely to the pure strategy Nash equilibria, where the best that *OptimAgent* can do is to bid that same amount and win roughly $\frac{1}{3}$ of the time – this is likely the situation that is responsible for the lowerbound of the boxplot for this scenario (which is roughly $\frac{1}{3}$). In comparison, the occurrence of these best case and worst case situations is much less likely in the other scenarios, either because the number of players involved is too high for it to be statistically likely or because the strategies of the opponents are variable.

Finally, we noticed from inspection, that *OptimAgent*'s strategy s was invariably a pure strategy, i.e. *OptimAgent* would definitely bid 0, 1, or 2 coins depending on the estimated opponent strategies. While this surprised us initially, we realized that the total expected payoff W can be rearranged to:

$$W = \sum_{x=0}^{2} Pr(P1^x) \sum_{y,z \in [0,2]^3} PO_{P1}(x,y,z) Pr(P2^y) Pr(\text{P3}^z) \tag{4}$$

Since $\sum_{y,z \in [0,2]^2} PO_{P1}(x,y,z) Pr(P2^y) Pr(\text{P3}^z)$ is a constant given fixed P2 and P3 strategies, this means that W is a linear function with respect to P1's strategy parameters, i.e. $Pr(P1^x)$. Maximizing W will then necessarily maximize one of a_i, b_i, c_i, resulting in a pure strategy. In other words, if P2's and P3's strategies are known, the optimal P1 strategy is necessarily some pure strategy. This generalizes for any m of coins since W will take the same form.

C. Deep Q-learning

Deep Q-Learning is an instantiation of the Q-Learning algorithm from class, where the action-value function is approximated by a deep neural network that takes in some state representation as input. We follow the original deep Q-learning algorithm from [4], which trains the network with experience replay and uses an auxiliary target network as a surrogate for the Q-value at the next iteration. The network is trained via backpropagating an L1 loss between the Q-values of the policy network (which is actually playing the game) and the target network.

We use a simple 3-layer fully-connected neural network for our Deep Q-Network. For an n-player, m-coin game, the output of the network is a $(m+1)$ vector denoting the Q-values

of bidding any amount from $0, ..., m$. The input is a one-hot representation of the observation $(b, \mathcal{H})$, where onehot$(b) \in [0,1]^{m+1}$ is concatenated with onehot$(\mathcal{H}) \in [0,1]^{(m+1)\times(n-2)}$ (there are $m+1$ possible minimum bids for each of the $n-2$ non-trivial rounds of the game). The network architecture is summarized below in Table I.

Layer	Number of Neurons
Fully Connected	128
Fully Connected	256
Fully Connected	$m+1$

TABLE I: Deep Q-Network Architecture

During training, we employ ϵ-greedy exploration to train the network, where ϵ is annealed linearly from 0.95 to 0.05 throughout training. During gameplay, if the DQN agent has a budget of $m' < m$ coins left, we take the argmax over the DQN's first $m'+1$ Q-values as the agent's action.

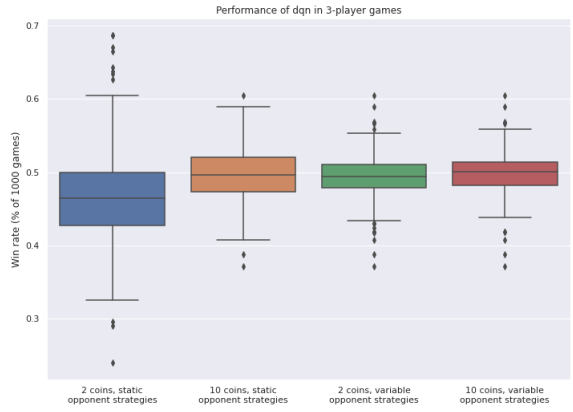


Fig. 4: Win rates of a Deep Q-Network trained w/ 10,000 rollouts

We present the evaluation results of our DQN method in Figure 4. In general, the DQN is able to find a deterministic strategy that is able to achieve a similar win rate as *OptimAgent*; we note that its performance tends to have higher variance (see the outliers in Fig. 4 compared to Fig. 3). As a model-free approach, Deep Q-Learning tends to be data-hungry and may take longer than the approach in Section V-B (the above performance took 10,000 game rollouts, around one minute of training). However, one advantage of DQN is that it is able to easily scale to larger versions of the game; in our preliminary experiments, we were able to reach $> 22\%$ win rate in a 10 player, 50 coin bidding game w/ variable policy opponents.

D. Value Iteration

Given the opponent strategies, we can convert the POMDP formulation of the game into an MDP by focusing on a single player. The information that a single player has at a given round of the game can be described by:

1) whether the player is actively in play,
2) whether the player has won or lost if the former is false,
3) the remaining coins the player has, and
4) the history of lowest bids during the game, and
5) the history of players eliminated from the game.

If we consider these pieces of information as the definition of single-player fully-observed states, then the transition probabilities between such states are fixed once the opponent policies are known. We now have an MDP for a single player, and can solve the MDP using a variety of techniques, including value iteration.

While value iteration is an exact solution method, we use it here to obtain approximate solutions for games of larger sizes. Games with more players and starting coins last more rounds, resulting in large bid histories and state spaces. Hence, we leave out these histories when describing single-player states to keep the problem tractable. In addition, while the transition probabilities are exactly computable in principle, they can get unwieldy for games with many players or starting coins, so we instead generate a set number of transitions to obtain a maximum likelihood estimate of the transition probabilities. Finally, for each simplified state, we compute its value function by repeatedly applying the Bellman equation:

$$U_{k+1}(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} T(s'|s,a) U_k(s') \right)$$

until convergence is approximately achieved, and extract the optimal policy from the converged U using:

$$\pi(s) = \arg\max_a \left(R(s,a) + \gamma \sum_{s'} T(s'|s,a) U(s') \right)$$

The evaluation results of the value iteration method on three-player games are shown in Figure 5. We observe a marginally poorer performance of the strategies obtained via value iteration compared to the previous two methods. This may perhaps be due to the omission of partial information to reduce the state space as explained above, resulting in that the player trained under value iteration has to make his bids based primarily on the number of coins he has remaining.

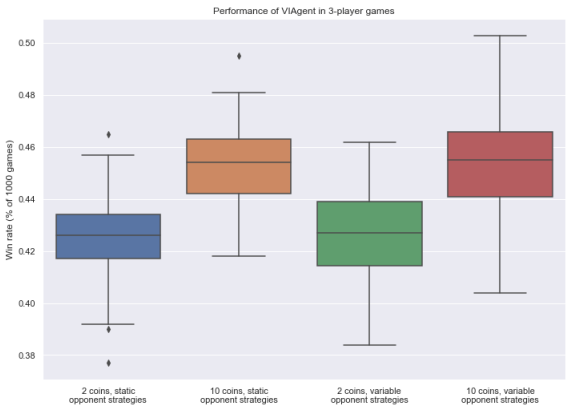
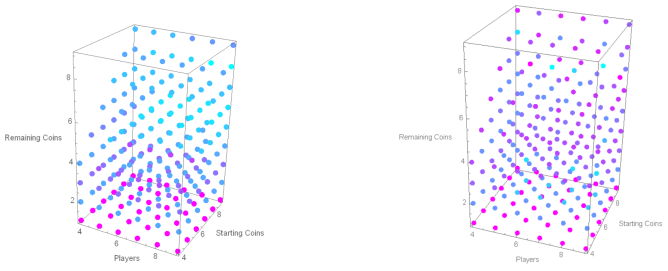


Fig. 5: Win rates of a Value Iteration agent

Nevertheless, this approach may be more useful in exploring the optimal policies for larger games. The reduced state space implies that value iteration can still be performed in reasonable time provided the number of transitions generated in order to obtained the MLE model and the tolerance for value function convergence are adjusted accordingly. For instance, compared to the DQN approach, the optimal strategy from value iteration is able to achieve $> 34\%$ win rate in a 10 player, 50 coin bidding game. The approximations that we have made under the value iteration approach have allowed it to scale reasonably and yet provide a good model-based solution.

We make further exploration on the optimal policy in larger games, first against random opponents for games with $4 \leq m, n \leq 9$. The policy is a function of three parameters: m, n and the number of remaining coins the player has. To visualize such policies, we plot a 3D array of points as in Figure 6a, where the axes represent the parameters and the darkness of each point represents the optimal fractional bid. For instance, the point at the top far corner of the figure corresponds to $m = 9$, $n = 4$ and nine coins remaining, and its slightly lighter color means that the optimal bid is a smaller fraction of the nine coins (four-ninths to be precise). We see that the figure shows a gradient of increasing lightness as the number of remaining coins increases. This can be compared with the optimal policy against opponents that anticipate that others behave randomly, shown in Figure 6b; in other words, Figure 6b is optimal against Figure 6a. Figure 6b instead has darker points near the top of the figure, showing how the player, anticipating that his opponents will make lower bids as in Figure 6a, should make a larger bid to ensure he moves on to the next round. This color inversion between the two figures does not appear in the lower half of the figures, likely because the possible bids there are much more constrained.



(a) Random (b) Optimal against random

Fig. 6: Optimal fractional bid against labelled opponent policies for games with $4 \leq m, n \leq 9$

Now, we consider games against random opponents with $n \leq 300$ total players, fixing the number of starting coins to be 50. The optimal fractional bid for each n and number of remaining coins is shown in Figure 7, again with lighter colors representing a smaller fractional bid. The colors do not vary completely smoothly due to the inherent stochasticity of our value iteration approach. There should be a natural gradient towards lighter colors on the right of the figure, because the denominator of the fractional bid is increasing. It is hence notable that the optimal fractional bids are actually higher when the player still has many remaining coins (roughly between 30 and 50). This is possibly because with a larger number of players, the minimum bid of the random opponents in each round tends to be lower, so the player can afford to make larger bids at the start, knowing that he has a smaller chance of being eliminated in later rounds.

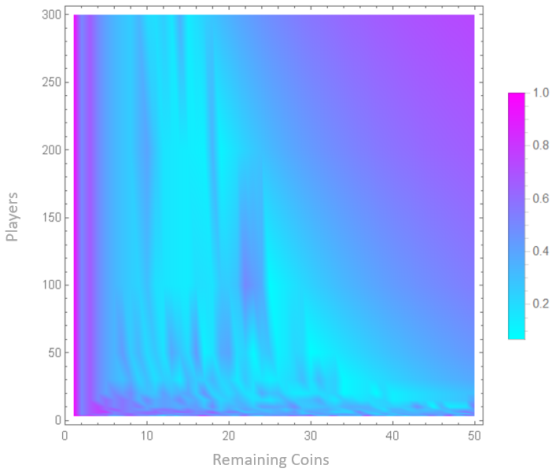


Fig. 7: Optimal fractional bid against random opponents in large games with $n \leq 300$ and $m = 50$

VI. Future work

For the value iteration approach, a more accurate MDP formulation of the game can be adopted by incorporating the bidding histories that have been omitted to reduce state space. More transition samples would then be required to produce an accurate MLE of the underlying MDP model, and the value iteration would incur greater computational costs. However, the policies are then tailored to the bidding histories, and will be no worse (likely better) in performance than the policies we have obtained using the current approach.

In exploring the optimal policy space of larger games, we can also consider games with a much larger number of starting coins m. In the limit of large m, the discretization of coins no longer matters, and we expect the optimal fractional bid to become independent of m.

Our DQN implementation makes a crude approximation of the POMDP as an MDP (treating the observations as states); future work can use deep RL methods that better accommodate the stochasticity of the formal POMDP specification [5].

VII. Division of work

Alan carried out the initial implementation of the game, and afterwards adapted the eventual implementation to support single-player states as required in the value iteration approach. He implemented and carried out value iteration for both three-player and larger games, and was responsible for policy visualization and analysis in larger games.

Caleb improved upon the initial game implementation done by Alan, and designed and implemented the architecture of the eventual game system, such as the game engine and scripts for simulation, evaluation, and visualization of results. He was in charge of the Nash equilibrium analysis of the 3-player game (implementing symbolic and numeric Nash equilibria solvers) and developed & implemented the parameter estimation + optimization approach in *OptimAgent*.

Jonathan implemented an OpenAI Gym interface to the game implementation for use with general RL algorithms, and implemented the DQN and its associated training and evaluation scripts. He carried out experiments and visualization for DQN results in small- and larger-scale versions of the game.

REFERENCES

[1] Easley, David and Kleinberg, Jone. "Networks, Crowds and Markets" ISBN 978-0=521-19533-1, Cambridge University Press (2010): 166-168.

[2] Ganzfried, Sam, Austin Nowak, and Joannier Pinales. "Successful Nash Equilibrium Agent for a Three-Player Imperfect-Information Game." Games 9.2 (2018): 1-2.

[3] Brown, Noam, and Tuomas Sandholm. "Solving imperfect-information games via discounted regret minimization." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 33. 2019.

[4] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

[5] Baxter, Jonathan & Bartlett, Peter. (2000). Reinforcement Learning in POMDP's via Direct Gradient Ascent. Proceedings of the Seventeenth International Conference on Machine Learning.