A considerable amount of work in artificial intelligence has focused on producing policies for playing formal games in which all players share access to the same information. However, many games feature asymmetries between players’ knowledge about the game-state, which are not amenable to classic tree-search algorithms like minimax and expectimax. CITE. Our project models the game of Liar’s Dice, in which states are partially unknown to players, who must reason strategically under uncertain. We describe an algorithm that can play Liar’s Dice by doing XYZ.

**Liar’s Dice**

Liar’s Dice can be played with any finite number of competing players, each beginning with a preset number of dice. At the start of each round, players roll their dice, but do not reveal the face-values of their dice to their opponents. The set of face values that come up for each player are form their “hand” for the rest of the round. Turns are taken cyclically starting with a random player, who can choose to “bid” that at least a certain number (ante) of a given face-value are on the table. Each turn, a player must place a bid whose ante is strictly greater than the previous ante, or choose to “challenge” the bid of the previous player. If a bid is challenged, all players reveal their hands and count how many dice on the table possess the face-value named in the bid. If this quantity is greater than or equal to the ante, the challenging player loses a die, otherwise, the player whose bid was challenged loses a die. After a challenge is resolved, all players reroll their concealed dice, and the round begins anew with the winner of the challenge placing the first bid. Players can also choose to “confirm” the previous bid, which will cause all opponents to forfeit a die if the ante is exactly equal to the number of dice with the face-value named in the bid. Otherwise, the bid-confirming player loses a die. Once a player has lost all their dice, they are out of the game. The game goes until all but one player, the winner is out, are out.

Probability plays a major role in Liar’s Dice, but players must also think strategically. The game’s tension derives from one’s need to place probable bids that opponents do not challenge, while simultaneously driving up the ante and forcing opponents to place improbable bids that can be safely challenged. Successful players monitor what their bids communicate about their hand and form hypotheses about opponent’s hands based on their actions, which can be fine-tuned with learning. As such, Liar’s Dice, although simple in outline, is an AI task that engages probabilistic reasoning and machine learning, as well as game theory.

**Relevant Past Work**

Laird, Derbinsky, and Tinkerhess (2011) employ the Soar cognitive architecture to create an agent that learns heuristics for playing Liar’s Dice, without using game-tree look-ahead. Their system uses a set of knowledge sources to compute action-preferences that are compiled (or “chunked”) into if-then rules for responding to opponent’s actions. For example, one chunk might state “*IF* I’m considering the action of denying a bid *AND* there are many hidden dice, *THEN* assign a low preference to this action”. Each chunk is assigned a weight that initially represents the probability that the action it embodies is consistent with the true state of the game. By simulating rounds, and using the resolution of challenges as a reward signal (i.e., whether the agent or its opponent lost a die when a bid was called into question), the chunk-weights are adjusted by Q-Learning until they reflect expected reward, rather than the raw probability that they describe “true” game-states. The agent then plays by choosing the action embodied in the chunk with the highest weight for each turn.

Chunking preferences into If-Then rules overcomes the need to search for optimal action-sequences, but the version of Liar’s Dice they describe includes numerous extra rules, that are readily exploitable by the type of symbolic reasoning Soar architectures are designed to perform. For example, Laird et. al allow their agents to strategically reveal their dice to other players, and give some face-values special “wildcard” properties. In other words, including such rules in the game description may complicate how Liar’s Dice is played, but it allows “nicer” features to be provided for a learning agent. As such, we are extending this past work by introducing an agent that learns to play Liar’s Dice given a sparser description of the game. Where the Soar agent created action-chunks by treating the game rules as a list of logical predicates to incorporate into learning, we intend to mine feature Q-Learning by developing an agent with a richer internal model of opponents’ hidden knowledge.

**Model**

As a first modeling step, we consider the case where opponents embody fixed policies. Due to the uncertainty inherent in Liar’s Dice, we treat each opponent as a noisy signaler, whose actions reflect some underlying strategy, viewed through the randomness of their dice-rolls. Therefore Liar’s Dice can be modeled as a MDP where opponents are chance nodes with some bias that the agent must infer from experience.

States:

Includes fields for current player, the number of dice possessed by each player, the face-values of the agent’s dice, and the history of bids from the beginning of the round.

Actions:

Players can choose from a finite number of *Bid*-actions, selecting a die-value and a count for that value ranging between the current ante and the total number of dice on the table. A player can also *Deny* a previous bid, or *Confirm* it.

Utility:

Our Liar’s Dice model deviates from the classic MDP in that it has an end-state utility determined by whether the agent won or lost, rather than a per-transition reward. However, as shown with the Soar agent, a reward signal can be induced from conflict resolutions during model learning.

**Algorithm**

Our first step in learning this model is to initialize an agent that treats all opponent actions as the output of a single “Environment” chance nodes. This simplifying assumption makes the learning step equivalent to the two-player training phase described in Laird et al. The major difference here is that instead of learning features over rule chunks, our algorithm XYZ.

**Going Forward**

The implementation of Q-Learning we describe for Liar’s Dice has two limitations we plan to address. First, by collapsing all opponents into a single chance node in the MDP, we fail to account for their individual policies. There have been other adaptations of Q-Learning to multi-agent problems CITE, which treat a state’s reward and transition probabilities as functions of opponent’s subsequent actions. But because our game includes a component of imperfect information, it is not possible to minimize over opponent’s actions because, not knowing the actual state of the hands, we cannot determine if their choice to “deny” or “confirm” would produce positive or negative reward for the agent in advance.

Instead, we propose XYZ.

Second, although the players’ bid histories are used as features, our agent does not develop an explicit model of other player’s hands based on these bids. Given that our description of Liar’s Dice admits fewer features than Laird et al., we believe giving our agent a mechanism for constructing beliefs about other player’s hands (and beliefs about other players’ beliefs about other players’ hands) can provide more features for Q-Learning. We plan to accomplish this by creating a Bayesian Network at each game-state for which it is the player’s turn, as pictured in **figure 1**.

Here, each node represents a vector-valued random variable, and each column corresponds to a different player who went previously to the agent in a round. The nodes labeled H represent an n-dimensional vector across the n dice in a player’s hand. The nodes labeled B represent an n\*m-dimensional vector, reflecting what the given player thinks are the values of each of the m player’s hands. And the nodes labeled A represent the bids placed by each player (these are never “Deny” or “Confirm” actions, as the graph is constructed on a per-round basis).

The network reflects the intuition that one’s beliefs can depend only on the actions of other players, and one’s knowledge about one’s own hand. The agent’s job is then to assign a value to it’s *own* Belief node (pictured in yellow), by inferring the joint probability of the unknowns Hands variables (in blue), conditioned on available information (red nodes). We anticipate using Expectation Maximization to fit these unknown quantities. Once the belief vector has been assigned, these values can be used as features to improve Q-Learning.

**Preliminary Results**

We have established a baseline, oracle, and dummy agent to create appropriate metrics of comparison for our algorithm.

Baseline Agent:

On its turn, the baseline agent iterates over all actions available to it in a given game-state, and calculates the probability that each is consistent with the state’s unknown information. For example, if a baseline agent must respond to a bid of “four sixes”, and it already has two sixes in it’s hand, it subtracts this known quantity from the opponent’s bid, and computes the chance of there being “two sixes” among the unknown dice using the binomial distribution. It then selects whichever action was assigned the highest probability.

Oracle Agent:

The oracle represents an agent that can infer perfectly the unknown values in all players’ hands. It receives a complete description of the game state and assigns the previous player’s bid a truth-value by using everyone’s hand as a look-up table. If the previous bid was false, the Oracle denies it. If exactly true, the Oracle confirms it. Otherwise, the oracle places some bid chosen at random from a list of true bids.

Dummy Agent:

Of all the legal moves at a given game-state, the dummy agent simply picks one at random. It is included so both the Baseline and Oracle can be tested against a common inferior opponent.

The agents were tested by simulating 5,000, three-player games, where each player was initialized with 3 dice. Each agent was matched against two opponents of the same type. The win ratio of each can be summarized as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Agent\Opponent** | vs. Baseline-Baseline | vs. Oracle-Oracle | vs. Random-Random |
| Baseline | **33.66%** | 3.08% | 99.3% |
| Oracle | **83.94%** | 33.02% | 99.88% |
| Random | 0.24% | 0.0% | 32.7% |

The main diagonal can be viewed as a sanity check, which tells us that a player is not favored by their agent index. One metric to evaluate our algorithm would be to assess its performance against random agents in comparison to the Baseline and Oracle. However, given that their win ratios are almost identical in this match-up, this measure may be uninteresting.

Alternatively, we were surprised by how well the Baseline performed against the Oracle itself. We currently plan to assess our algorithm by simulating games against the baseline, and seeing where its performance falls in the delta between the bolded values (33.66%-83.94%).

References

Blair, J. R. S., Mutchler, D., Liu, D. (1993). Games with imperfect information. In *AAAI Technical Report* *FS-93-02*. Retrieved from <http://aaaipress.org/Papers/Symposia/Fall/1993/FS-93-02/FS93-02-009.pdf>

Laird, J. E., Derbinsky, N., & Tinkerhess, M. (2011). A case study in probabilistic decision making and learning in a symbolic cognitive architecture: Soar plays dice. In *AAAI Fall Symposium* – *Technical Report*. Retrieved from <http://alpha.tmit.bme.hu/speech/docs/education/SoarPlaysDice.pdf>.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th International conference on Machine Learning*. Retrieved from <https://www.cs.duke.edu/courses/spring07/cps296.3/littman94markov.pdf>