Intoduction to AI – Final Project

Ron Itzhak 311604938, Amit Segal 311340350, Amit Baskin 312259013, Avichai Haimi 305761637

Abstract

In this project we attempted to develop an intelligent agent to play and beat human players and simple agents in the famous, Russian card game “Durak”, where we chose the two-players version for complexity reduction considerations.

We used several AI algorithms to attempt to solve the problem: Reflex Heuristics, the Minimax Algorithm, and Approximate Q-learning:

The Reflex Herisrtics approach is an elementary go-to approach, and can be used throughout the game.

We chose to implement the Minimax algorithm as it can be useful in two-agent games, such as ours.

Since for the most part, the game is only partially-observable, and undeterminstic, we opted for a model-free approach, which is why we also chose to implement an approximate Q-learner. We chose approximate q-learning over q-learning since the game’s state space is very large, which means such a learner would have a very large amount of weights, which do not repeat often, making the Q-matrix sparse, and limited in its usefulness.

Finally, we measured the results by conducting matches between the different agents. Note that we also implemented a human agent, which can be used to play against any other agent we implemented.

Introduction

Rules

Durak is a traditional card game in Russia, where the goal is to get rid of the cards in your hand, and the **last** to do so is called the “Durak”, i.e. the “Fool”.

The game consists of 36 cards out of a standard deck of 54 cards, excluding the jokers and the 2-5 cards in every suit.

At the beginning you draw a card from the deck, and put it exposed under the rest of the cards in the deck, as the last card to be drawed. The suit of this card represents the “trump suit”: the strongest suit which beats every other suit.

Each player starts with 6 cards in his hand, and the game itself is carried out by rounds, such that when a round begins each player has to have at least 6 cards in his hand so long as there are cards to be drawn from the deck.

In each round there is an attacker and a defender. The attacker begins an attack by putting a card from his hand on the table, and the defender can defend with a card of a higher rank in the same suit or a trump card of any rank, if the attacking card isn’t a trump.

At the beginning of the game, the first one to attack is the player who has the lowest trump card in his hand, and during the game itself the players switch their roles, unless a player fails to defend and so remains the defender.

When a sequence of attack and defence occurs, there are at first two cards on the table, one on top of another, representing an attack card and a defence card to block it, and every further attack has to be of the same rank of the ranks of the cards which are already on the table.

If an attack has been blocked and the attacker doesn’t have any more cards to add (or simply doesn’t want to), then the cards on the table are discarded. Otherwise, if the defender fails to defend (or just chooses not to), then all the cards on the table go to the defender’s hand.

Furthermore, there can be a total of maximum 6 attacks every round and in the case of drawing cards, for reaching 6, the attacker draws first.

And so the players keep playing until they get to the end-game where the deck is completed, and the first to get rid of his cards is the winner, or as the Russians say: **not** the “Durak”.

Durak as a Problem

Predicting if an attack can be answered in Durak is NP-Hard, After getting to know the rules of the game, we can make several observations regarding the properties of Durak as an AI problem:

- Partial observability - for most of the game we can only observe the cards in our hand, on the table, and outside of play. **Important note**: when the deck is emptied, the game is fully observable.
- Undeterministic - as this game is played with a shuffled deck, it is undeterministic.
- Multi-agent - in our case, two agents.

Methods

A priori, the reflex agent approach for this problem would be to choose the smallest card available in each stage, so that you don’t waste your high cards in order to be able to defend when needed.

Our SimplePlayer uses this approach and with this simple base line, we test our more sophisticated methods: the Reinforcement Learning (namely, approximate Q-learning) and Minimax with (using alpha-beta pruning for efficiency).

In order to try and learn an optimal state-action system, we used both the Q-learning and the approximate Q-learning algorithms, and at the end-game stage we also tried the Minimax algorithm where there is full knowledge at each state and it is possible to search through the states-space (note that search algorithms such as A-star search cannot be applied to this problem, as it is not composed of a single agent). Indeed, since we have complete knowledge in the end-game, one could not help but thinking of the Minmax approach which is highly intuitive to apply in such a case.

For the Minimax agent (the alpha-beta pruning algorithm) and for the Q-learning agents we used the code from the corresponding exercises in the course, where we implementd them, and furthermore, we implemented problem-specific feature extractors, and evaluation functions for developing the q-table.

The evaluation function used in the Minimax algorithm used the difference between the number of cards in the players’ hands. We also later used this function for Q-learning rewards.

This problem presented a challenge in the form of the size of the large states-space, and it required us to adjust the algorithms we used:

- In the Minimax algorithm we used depth of 5 which in many scenarios is not enough for reaching the end of the game. And yet, even this depth led to slow running times.
- We abstained from saving zero-valued weights, thus reducing the sparsity of the q-values and their memory consumption considerably.
- We used approximate Q-learning with feature extraction for dimension reduction. This way we managed to represent the different states abbreviatedly, thus saving a lot of memory in the q-table and also being able to find different states which are similler in their main properties and so mapping them to similler actions.

The features we used include several properties of the cards a player holds, e.g. the total number of cards in his hand multiplied by the inverse of the number of cards remained in the deck (plus one, to avoid dividing by zero), which increases the significance of the number of cards in the hand as the deck gets smaller, for indeed as we get closer to an empty deck, the more urgent it is to get rid of your cards; the number of trump cards, the most poweful cards in the game which are very important for both attack and defence, especially for the stage of the end-game where they play a crucial role; the variance of the ranks of the cards, in which a low one is a good property for attack, since as the cards ranks are close to each other, the higher the probability to attack with more cards; the variance of the suits, which is important for defence from different suits; the number of “strong cards”, those which are over 10, an important property for both attack and defence, i.e. for strong attacks and for high probability to be able to defend.

Indeed, these features project the main relevant data of a state of the game, and surely we observed this in the process of adding these features to the learning of our q-agent.

Now, the reason we thought of the Q-learning method in the first place was that most of the game there is only partial knowledge about the game states. During the early stages of the game (i.e. so long as there are still cards in the deck), the problem is only partially observable since the location of the cards which have yet to be seen is not known, i.e. they could be either in the deck or in the opponent's hand, and furthermore, if they are in the deck, it could be in any order. In other words, we just don't know with probability 1 which cards the opponent holds and we don't know which cards each of the players is going to draw and when. These facts, due to the random nature of cards games, implied to a "model-free" approach as we don't have a direct access to all successor states, until new cards are placed on the table by the opponent or until we draw new cards. Therefore the q-agent learns by evaluating states and deducing actions from them, by playing a large number of games and developing a matrix to decide which action to take in each state.

Running Instructions (aquarium computers)

In order to run our code, install ImageTk and then run: python3 GUI.py <p1> <p2> <num_games_in_a_row> where p1 can be 'q', 'reflex', 'minimax' or 'random', and p2 can also be 'human'.

Evaluation

We created some graphs that represent the learning process while running approximate Q-learning: we trained the agent against the random agent and reflex agent in succession and observed their progress over the episodes.

Our agent's success rate is measured by the percentage of games won within a single training episode (can be seen in the graph's y axis). Our learning methodology against the random agent was as follows:

1) 100 training episodes using learning rate (denoted by α) of 0.008
2) 100 additional training episodes using learning 0.08. This is evident in the graph by the improvement shown after 100 epochs.

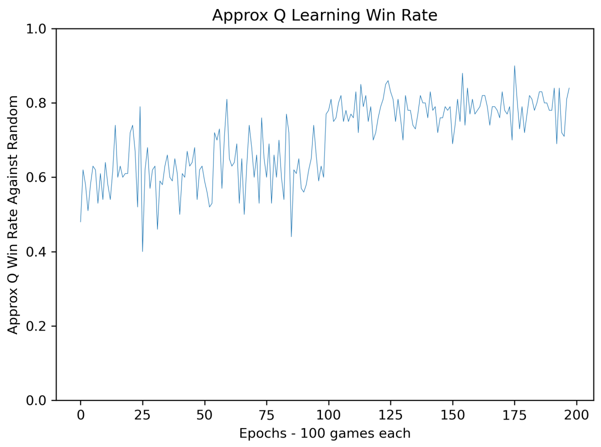

Figure 1. Q-learning agent win rate against random agent over 200 training episodes

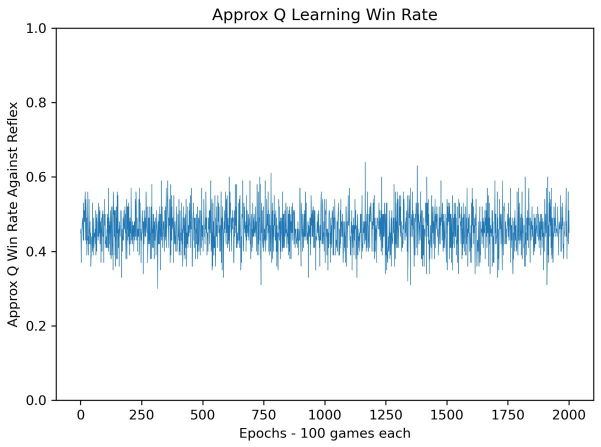

Figure 2. Q-learning agent win rate against reflex agent over 2000 additional training episodes

We then continued to train that same agent against our reflex agent for 2000 additional training episodes:

As evident, our approximate q-learning agent was unable to improve its win rate against our reflex agent - we suspect this might be because of insufficient features for modeling the game states.

Note: after observing the live win rate during the first training episode, we saw that the majority of the agent's training was during that episode i.e. after ~20 games, we saw a win rate of ~0.4, by ~50 games, win rate increased to ~0.55, etc.

During development of the approx. q-learning agent, we saw that more features do not necessarily mean better results. However, our experiments have led us to discover the importance of end-game features (i.e. features instansiated only when the deck was empty). This had significant impact on our approx. Q-learning agent results. Perhaps further experiments with the end-game features could lead

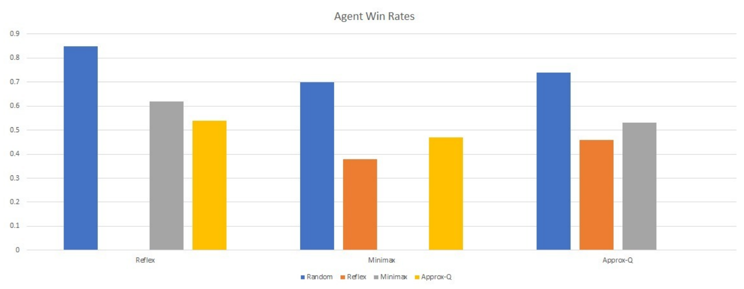

Figure 3. Final win rates between all agents (percentages are written in regard to column agents)

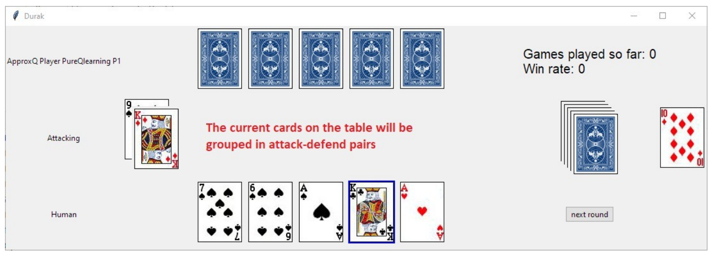
Durak
ApproxQ Player PureQlearning P1
Games played so far: 0
Win rate: 0
Attacking
The current cards on the table will be grouped in attack-defend pairs
Human
next round

Figure 4. GUI demonstration

us to better results.

The GUI

When opening the GUI with a human player, the game starts immediatly and the opponent agent will automatically start playing. The GUI will then wait for selection from the player. You will see your cards at the bottom of the screen. The cards surrounded by a blue line are the valid cards to play. In any state of the game except for a first attack, a button will appear on the right, allowing you to skip to the next round. In case non of the cards in your hand are valid, the only option is to skip to the next round by pressing the button. Note that your cards will be replenished automatically from the deck.

Conclusion

While approximate Q-learning can be used to reduce the states-space dimensionality, it is quite difficult to do so efficiently in a way that preserves relevant information for the agent's decision making. Conversely, our Minimax and Reflex agents which do not rely on modeling the rewards between all transitions, yield better results. However, their effectiveness is limited: for example, Minimax requires full observability and optimality on the opponent's side, and the Reflex agent can be outsmarted quite easily by a human player.

We would also like to note that we became aware of this paper from Stanford University tackling the same problem. As you can see, they too had problems getting impressive results with their efforts. However, empirically our results are markedly better than theirs (around 20% better against the same reflex agent).

References

- Durak's Wikipedia page - Link
- The Complexity of Playing Durak - Edouard Bonnet - Institute for Computer Science and Control, Hungarian Academy of Sciences - Link
- Learning Game Playing Strategy for Durak, written by Sammy Nguyen and Narek Tovmasyan from Stanford University. - Link