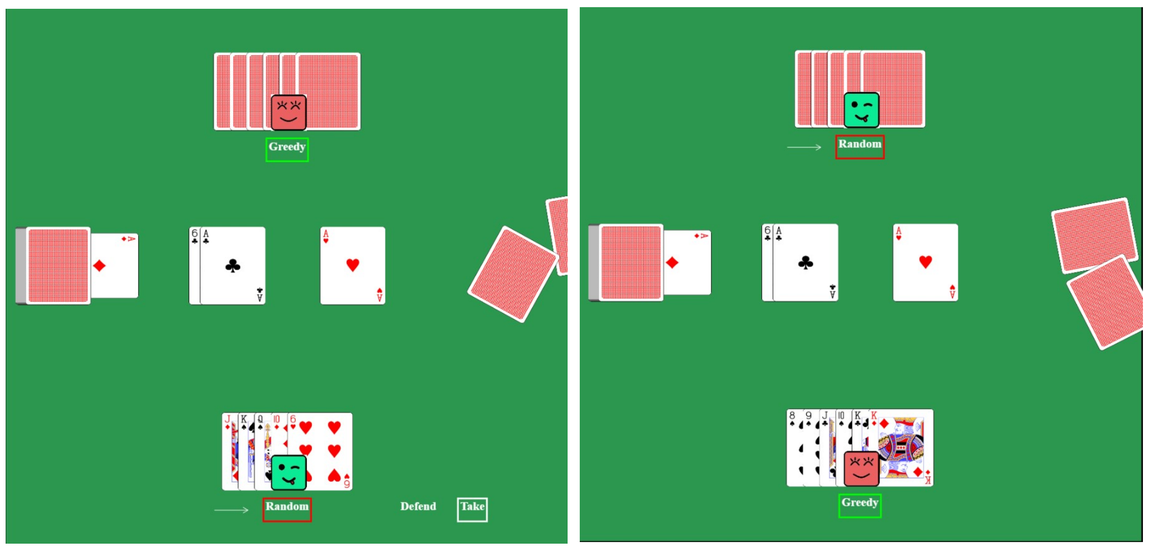
Artificial Intelligence for the Card Game Durak

Author: Azamat Zarlykov | Supervisor: Adam Dingle, M.Sc. | 2023

Introduction



Durak is a two-player card game that starts by dealing cards to each player until they have a hand of six cards. The final card of the deck is then placed face up to determine the trump suit for the game. The gameplay consists of a series of bouts where the attacker begins by placing a card, and the defender may attempt to beat it by playing a card of their own. After that, the attacker can continue the attack or end it, and the defender can continue to defend or pick up the cards. The winner of each bout becomes the attacker for the next bout. The aim of the game is to get rid of all one's cards. Our experiments have showed that the game's average branching factor is about 2.2, and typical games are 30-60 moves in length. The goal of this work was to develop various AI players for the game, with a focus on achieving an average movement time of around 100 ms, and to compare their performance.

Artificial Intelligence Agents

The project features five distinct agents:

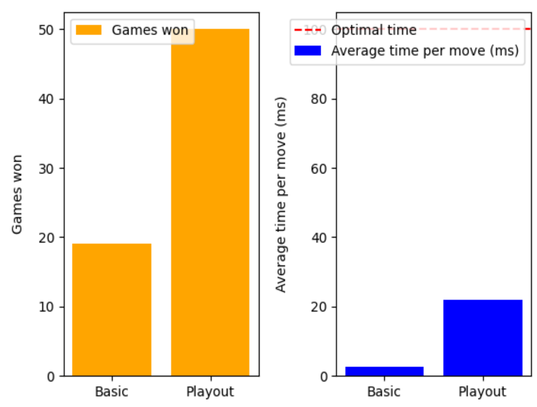
- **Random**: makes random moves without considering the game state.
- Rule-based agents:
  - **Greedy**: makes moves by selecting the lowest value cards.
  - **Smart**: uses strategies that take into consideration the opponent's hand.
- **Minimax**: evaluates the best move based on the minimax principle. The user can choose from two heuristics to evaluate game states:
  - Basic, a heuristic function that does not look ahead.
  - Playout, which plays out the remainder of the game using a fixed strategy to see which player will win.
- **MCTS (Monte Carlo Tree Search)**: builds a game tree in memory dynamically, using playouts to estimate move values.

Minimax Heuristic Comparison

The basic heuristic function considers the following aspects when evaluating the state:

- The player's hand
- The player's hand size
- Existence of weaknesses, which are ranks that are dominated by a card of the opponent

The following graph shows that the playout heuristic outperforms the basic heuristic with a search depth of 3 in a series of games against the greedy agent.



Games won and time per move in a 100-game simulation against the greedy agent

Monte Carlo Tree Search

Monte Carlo tree search iteratively builds a tree of nodes representing game states. On each iteration, it performs these steps:

- **Selection**: Traverse the tree to select the most promising node to expand.
- **Expansion**: Add a new child node to the selected node, representing an unexplored move.
- **Simulation**: Run a random simulation from the expanded node to estimate the potential outcome.
- **Backpropagation**: Update win/loss statistics to improve future selections.

After a certain number of iterations, the algorithm selects the best move from the generated game tree. The selection phase uses the UCT (Upper Confidence Bounds applied to Trees) formula. It balances exploration and exploitation by considering the average reward and the number of visits of each child node.

Framework and Selected Technology

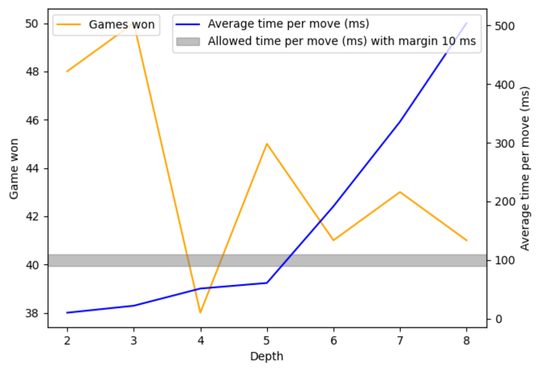
The game is implemented in C# for .NET 6. It includes a command-line interface with options for running experiments between AI players.

Hidden Information

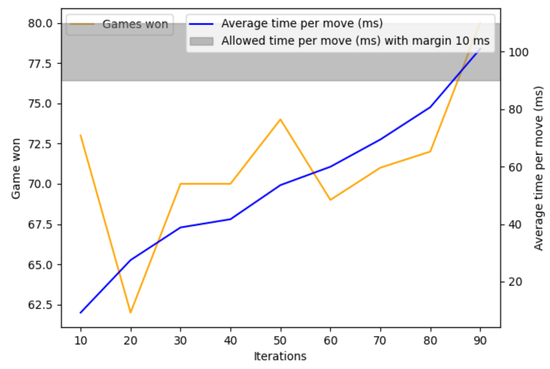
Durak is a game with hidden information, which makes it challenging to develop AI players. The random and greedy agents are not affected by this constraint, but the other agents are. The smart agent overcomes this restriction by maintaining a record of observed cards and deducing the opponent's cards in the endgame. The MCTS and minimax agents work by exploring a tree of game states, but in the presence of hidden information the current game state is not fully known. To overcome this limitation, they use a **sampling** method that randomly generates game states based on currently observable information. These agents explore the game tree starting with each of these random states, and choose the move that performs best in the largest number of random states.

Minimax and MCTS Parameter Selection

I ran experiments, comprising 100 games against a greedy agent, to determine parameters (depth and iterations) for the minimax and MCTS agents that maximize performance while still allowing the agents to make moves in around 100 ms on average. The number of samples was set to 10 in these experiments. The results of the experiments led to the selection of a depth parameter of 3 for the minimax algorithm and a parameter of 90 iterations per sample for the MCTS algorithm.



Minimax agent: win rate and move time by search depth



MCTS agent: win rate and move time by number of iterations per sample

Results

I evaluated the relative performance of the following agents, listed with their parameters and average move time:

- Random
- Greedy
- Smart

} (<0.1ms)

- Minimax (30 samples, search depth = 3, playout heuristic) (~169ms)
- MCTS (30 samples, 30 iterations/sample, exploration constant = 1.4, greedy playouts) (~45ms)

The MCTS agent was the strongest, followed by the smart and minimax agents.

	random	greedy	smart	minimax	MCTS
random		8.4% ± 2.8%	6.2% ± 2.4%	6.9% ± 4.3%	1.5% ± 1.1%
greedy	91.5% ± 2.8%		50.6% ± 1.2%	47.4% ± 1.7%	19% ± 4.1%
smart	93.7% ± 2.4%	49.3% ± 1.2%		53.8% ± 3.7%	26.9% ± 4.7%
minimax	93.1% ± 2.6%	52.5% ± 1.7%	46.2% ± 3.7%		16.7% ± 3.8%
MCTS	98.4% ± 1.1%	80.9% ± 4.1%	73.1% ± 4.7%	82.8% ± 4.4%	

Win rates between agents (98% confidence intervals)

Conclusion

This study developed an AI framework for the card game Durak and tested various agents. In the tournament, the MCTS agent consistently outperformed the others. The smart agent was the second most successful, though the minimax and greedy agents performed nearly as well as it. The playout heuristic in the minimax agent greatly outperformed the basic heuristic. Also, the sampling technique worked well for dealing with hidden information.

Acknowledgments

I am grateful to my supervisor Adam Dingle for his patience, expertise, and willingness to go above and beyond to provide assistance.