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 attempts to beat it by playing a card of their own. The attacker can continue the attack or end it, and the defender can continue to defend or pick up the cards. This ensures the variability of strategies that can be used during the game. The aim of the game is to get rid of all one's cards and the player left holding cards is declared the fool. The aim of the work is to develop the range of AI agents that use various techniques, including rules-based heuristics, minimax search, Monte Carlo tree search and, by comparing their performances in a mutual play, identify the most effective and efficient agent.

Framework and Selected Technology:

Durak AI framework includes the following components:

- Game model: this is the core component that implements the rules and logic of the game Durak.

- AI agents: these are the different agents developed to play the game of Durak.

- Command-line interface (CLI): this allows for interaction between the game model and the AI agents.

All these components are implemented using the C# programming language and targeted for the .NET 6 platform.

Artificial Intelligence Agents

The project features five distinct agents: Random, Greedy, Smart, Minimax, and Monte Carlo Tree Search:

- 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: a game-theoretic algorithm that evaluates the best move based on the minimax principle. It uses two heuristics to evaluate the state:
  -Basic, which is a traditional approach using the game state.
  -Playout, which generates a copy of the game state and plays out the whole game using greedy agents.

- MCTS (Monte Carlo Tree Search): an algorithm that uses simulated playouts of a game to estimate the value of each possible move.

Each agent was tested and compared in perfect and imperfect information scenarios, and the results were analyzed to identify the most effective technique.

Acknowledgments:

I am grateful to my supervisor Adam Dingle for his support and guidance throughout the process and for his constant patience, expertise, and willingness to go above and beyond to provide assistance.

Github:

Results:

The experiment was conducted both in open and closed world environments to gain a better understanding of the adaptability of the agents and to determine whether the agents that performed best in perfect information scenarios are able to achieve similar success in an imperfect one. The results are provided in tables where the numbers within the brackets represent the amount of games played to determine the winner with sufficient level of confidence.

Open-world:

(players have complete knowledge of the game state and can see all the cards)

The MCTS agent is the clear winner in terms of win rate in the open environment, followed by the Minimax agent. The smart agent had a higher win rate than the greedy agent, but it took more simulations to find a significant winner. The random agent had the lowest win rate among all agents. Therefore, the MCTS agent with its respective parameter configurations is the most effective at winning games.

	random	greedy	smart	minimax	MCTS
random		5.6%-11.3% (500)	3.3%-8.0% (500)	1.2%-4.6% (500)	0.0%-1.1% (500)
greedy	88.7%-94.4% (500)		45.7%-49.9% (3500)	33.5%-43.6% (500)	1.1%-4.4% (500)
smart	92.0%-96.7% (500)	50.1%-54.3% (3500)		1.2%-4.6% (500)	2.8%-7.2% (500)
minimax	95.4%-98.8% (500)	56.4%-66.5% (500)	55.0%-65.3% (500)		16.4%-24.8% (500)
MCTS	98.9%-100.0% (500)	95.6%-98.9% (500)	92.8%-97.2% (500)	75.2%-83.6% (500)	

Table 1: Tournament between the agents in the open world.

Closed-world:

(players do not have complete information about the game state and can only see certain elements such as their own cards and the trump card)

The closed-world tournament results indicate that the MCTS agent with its specified parameter configuration had the highest win rate among all agents, with strong performance against all competitors. The Smart agent was the second most successful, performing well against the Random and Minimax agents. The Minimax agent and the Greedy agent followed, while the Random agent had the lowest win rate. These results suggest that the MCTS agent is the most effective at winning closed-world games.

	random	greedy	smart	minimax	MCTS
random		5.6%-11.3% (500)	3.8%-8.7% (500)	4.3%-9.5% (500)	0.4%-2.7% (500)
greedy	88.7%-94.4% (500)		49.4%-51.9% (10000)	45.7%-49.2% (4500)	14.9%-23.2% (500)
smart	91.3%-96.2% (500)	48.1%-50.6% (10000)		50.1%-57.5% (1000)	22.2%-31.6% (500)
minimax	90.5%-95.7% (500)	50.8%-54.3% (4500)	42.5%-49.9% (1000)		12.9%-20.6% (500)
MCTS	97.3%-99.6% (500)	76.8%-85.1% (500)	68.4%-77.8% (500)	78.4%-87.3% (500)	

Table 2: Tournament between the agents in the closed world.

Conclusion:

This study developed an AI framework for the card game Durak and tested various agents such as Minimax, Rule-Based, and MCTS. The MCTS agent consistently outperformed the others in both perfect and imperfect information versions of the game. The results show the effectiveness of MCTS for developing AI agents for Durak. The parameters used in the MCTS agent, such as iterations, exploration constant and simulation strategy, greatly impacted its performance. By adjusting these parameters, strong results were achieved in both perfect and imperfect information scenarios.