Universiteit Leiden
The Netherlands

# Opleiding Informatica

An Analysis of CuBirds using Different Decision-Making Agents

Noëlle Boer

Supervisors:
Jeannette de Graaf & Mark van den Bergh

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

01/07/2025

**Abstract**

In this thesis two main topics are researched. First, we examine whether particular game strategies are especially effective in a two-player game of CuBirds. Second, we examine whether certain starting positions offer a measurable advantage to a player. This analysis is performed through a simulation using three decision-making methods: a random method, a Flat Monte Carlo method, and a heuristic method. The heuristic decision-making method is tuned using a genetic algorithm. All three methods contribute to the analysis of the game, but the heuristic method best balances computational cost and efficient gameplay. The results from different methods show that the starting player does not have a significant advantage and that the heuristic method prefers seven unique bird cards over two sets of three bird cards. Furthermore it seems that starting with a common bird in the collection gives a player a significant advantage.

# Contents

# 1 Introduction

CuBirds is a modern card game released in 2018 [Ale18]. The game CuBirds is designed by Stefan Alexander and illustrated by Kristiaan der Nederlanden. The objective of the game is to be the first player to complete a bird collection by strategically placing cards in one of the four rows on the table and collecting bird cards at the right time. CuBirds provides an interesting environment for computational analysis, due to a combination of incomplete information about the opponent's hand and the randomness of the draw pile, which makes for a relatively large action-state space.
This thesis investigates two main research questions. First, it explores whether there are particular strategies that are most effective in a two-player game of CuBirds. For example, is it more profitable to focus on collecting rare bird cards or collecting common bird cards?
Second, it examines whether certain starting positions offer a measurable advantage. For example, does a player who starts with a rare bird in its collection have a significant advantage? Or does the starting player have an advantage in the game?
To address these questions, a simulation-based approach is used. Three different decision-making methods are implemented and evaluated by simulating thousands of two-player games.
The first method is a random decision-making approach that serves as a baseline and measures the performance of an agent that decides which move to make in a turn without any strategic intent.
The second method performs a Flat Monte Carlo Search for each legal move on a turn, the method simulates numerous random play-outs and selects the move with the highest win rate. This method can be used to analyze the performance and win rates of an agent that can plan ahead.
The third method is a heuristic decision-making approach, which selects a move based on a weighted evaluation of various game state parameters. The weights of these factors are decided using a genetic algorithm to balance the effect of the parameters and improve performance.

## 1.1 Contributions

This research will deliver the following items:

- A complete simulation environment for CuBirds, implemented in `C++`. The environment includes support for various decision-making methods: random, Flat Monte Carlo, and heuristic. The weight factors of the heuristic method can be optimized using a genetic algorithm. The framework is designed to be extensible with new decision-making methods.
- A set of experiments and their results, recorded in JSON format that allows the analysis of the performance of the game and the three decision-making methods.

## 1.2 Thesis Overview

Section 2 describes the rules and mechanics of CuBirds; Section 3 reviews related work and background literature; Section 4 presents the basic architecture of the simulation environment; Section 5 describes the three decision-making methods in more detail; Section 6 describes how the weight factors of the heuristic method are tuned using a genetic algorithm, followed by experimental results and analysis; Section 7 gives a detailed overview of the different experiments that were conducted and their results; Finally, Section 8 summarizes the findings and discusses suggestions for future work.

# 2 The Rules and Mechanics of CuBirds

CuBirds is a turn-based card game that can be played with one to five players. The players need to gather a collection of bird cards to win the game. The rules of the game can be found in the rulebook and are also explained in the remainder of this section [Cat25].

## 2.1 Game Components

CuBirds consists of a set of 110 bird cards, distributed in eight species.
Each bird card contains an illustration of the bird, the number of bird cards needed for a small family and a large family, and the total number of bird cards of that species present in the game. In Figure 1, the eight different cards from the game are displayed. In Figure 2, one of the cards is enlarged to show where to find the information mentioned on a bird card.

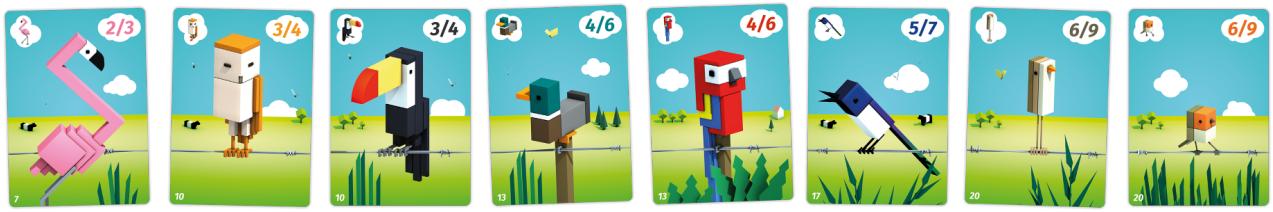



Figure 1: The eight different species used in the game CuBirds. From left to right the species are: flamingo, owl, toucan, duck, parrot, magpie, reed warbler and robin. The cards in the figure are sorted according to rarity, with the leftmost card (flamingo) being the most rare [Spe24].

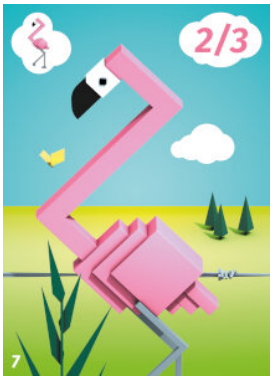



Figure 2: The flamingo bird card. In the upper right corner is the number of bird cards needed for respectively a small family (two cards) and a large family (three cards). In the lower left corner is the total number of flamingo cards in the game (seven cards) [Yuc25].

## 2.2 Game Setup and Table Configuration

Before starting the game, all 110 bird cards are shuffled. Then four rows of three bird cards are placed face-up on the table. Each row must consist of three different types of bird cards. The remaining cards are placed face down as a draw pile on the table. Each player draws eight cards for their starting hand and one card for their starting collection, which is displayed face-up on the table. The table also has a discard pile that is empty at the start of a game. Figure 3 shows a possible configuration of a two-player setup.


Figure 3: A possible setup of a two-player game of CuBirds. The figure shows four rows, each containing three different bird cards, a face-up starting card for each player's collection and a drawpile left of the first row [Spe24].

## 2.3 End of the Game and Winning Criteria

The game ends when a player completes their collection or when it is impossible to deal eight new cards to one of the players for a new hand, even after reshuffling the discard pile into the draw pile. This can happen at phase 1g or 2c of a turn, which can be found in Section 2.4.
A complete collection consists of either at least one card from seven different species (out of eight), or two sets of at least three bird cards from the same species.
When the game ends due to an insufficient draw pile, the player with the most cards in their collection wins. When two players have the same number of cards in their collection, the game ends in a tie.

## 2.4 Turn Structure

A turn in CuBirds consists of two phases. In the first phase of a turn, a player must place one or more cards from their hand on the table. After this, the table is resolved in reaction to that player's move. This means that the player collects the enclosed bird cards and more bird cards are added from the draw pile to the row if needed.
In the second phase of a turn, a player may play cards from their hand to add cards to their collection. For clarity a step-by-step overview of a turn is given:

1. Place Birds on the Table (Mandatory)
    (a) The current player chooses a species in their hand to play this turn.
    (b) The current player chooses a row and a side of this row to place the bird cards of the chosen species.
    (c) The current player places all bird cards of this species from their hand on the decided row and side.
    (d) If the cards are added to a row that already contains the played species, the current player takes the bird cards that are enclosed by the played species and adds them to their hand. An example of this phase can be found in Figure 4.
    (e) If after collecting the bird cards there is only one species of bird cards left in a row, cards are added from the draw pile to the row until the row contains at least two different species.
    (f) If no bird cards were enclosed when the cards were added from the player's hand to the table, the player may choose to draw two cards from the drawpile.
    (g) If the hand of the current player is now empty, the turn ends immediately. Each player discards their hand and draws eight new cards from the drawpile. The current player gets another turn.
2. Complete a Flock (Optional)
    (a) The current player may choose a bird species in their hand. The number of bird cards from the chosen species has to be at least the amount needed for a small family.
    (b) If the number of cards was greater than or equal to the number of cards needed for a large family, the current player adds two cards of that species to their collection. Otherwise, the current player adds one card from that species to their collection. The rest of the cards from that species are discarded from the hand to the discard pile.
    (c) If the current player has no cards left in their hand after completing a flock, each player discards their hand and draws eight new cards from the drawpile. The current player gets another turn.

The players alternate turns until the game ends.

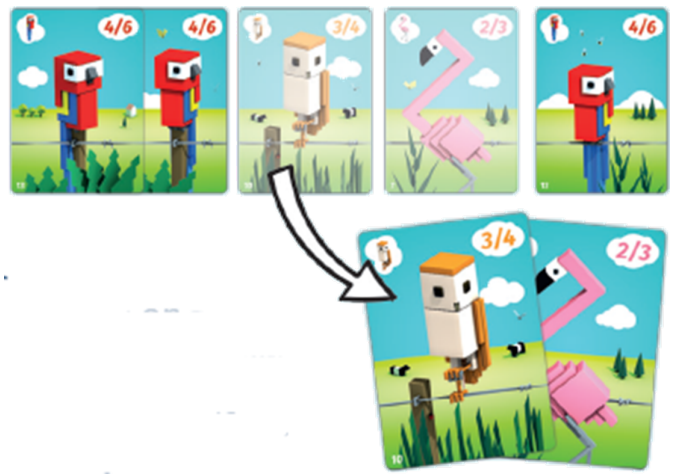



Figure 4: An example of a row from phase 1d. The player played the parrot species this turn on one of the sides of the row and collects the two birds that are enclosed by the parrot cards in the row: an owl and a flamingo [Yuc25].

# 3 Related Work

Although the CuBirds card game has not yet been the subject of academic research, several techniques employed in this thesis have previously been applied to the strategic analysis of similar games. These methods can be adapted to CuBirds to support a systematic exploration of its strategic properties. This section describes a number of papers in which these techniques were previously used to place this thesis in the context of previous work.

## 3.1 Random Method as a Baseline in Game Analysis

Agents that select moves uniformly at random are often used as baselines in game analysis. Kelly (2016) implemented a random baseline alongside a Monte Carlo Tree Search (MCTS) to quantify improvements compared to the random agent [Kel16].
Godlewski (2022) used a random agent both to analyze the complexity of the researched game and as the default policy in MCTS play-outs [God22].
Bakker (2024) also used a purely random agent as a baseline to compare with the results of the other agents [Bak24].

## 3.2 Flat Monte Carlo Simulation

The Flat Monte Carlo simulation is used in environments with stochastic elements or with incomplete information about the state of the game. The core idea is to evaluate each possible action from the current state by simulating a large number of random play-outs, then selecting the move with the highest empirical win rate. Unlike Monte Carlo Tree Search (MCTS), Flat Monte Carlo does not build or expand a search tree.

Shapiro (2003) provided a theoretical foundation for Monte Carlo sampling methods, including variance reduction techniques and convergence guarantees, which are applicable to game simulations [Sha03]. Browne et al. (2012) position Flat Monte Carlo as a precursor to MCTS, and emphasize its role in early game AI systems before tree-based exploration became a dominant technique [BPW+12].

In more applied contexts, Zook et al. (2015) demonstrate the effectiveness of Flat Monte Carlo simulation in predicting human strategic behavior in games [ZR15]. Their results show that even without full tree expansion, simulated game play can give information on the success rate of different move options. Additionally, recent work such as that by Blaauw (2020) explores Flat Monte Carlo simulation in the context of card games, reinforcing its accessibility and adaptability for academic experimentation in smaller or less formally studied games like CuBirds [Bla20].

### 3.3 Heuristic Agent Tuned with a Genetic Algorithm

Genetic and evolutionary algorithms have proven to be effective in optimizing heuristic weight factors in various strategic games. Montoliu et al. (2020) apply the N-Tuple Bandit Evolutionary algorithm to adapt heuristic feature weights, improving performance in multi-action, adversarial gameplay [MGPL+20]. Similarly, García-Sánchez et al. (2024) demonstrate the efficiency of coevolutionary training, where agents tuned with a genetic algorithm play against opponents based on MCTS [GSTFLC24].

In the collectible card game domain, Kowalski & Miernik (2022) investigate genome representations and fitness definitions for evolving evaluation functions through genetic programming and linear weight vectors [MK22]. Kowalski & Miernik (2020) introduce an evolutionary approach to deck building strategies, showcasing targeted genetic algorithm optimization for card selection policies [KM20].

Beyond card games, similar themes appear in other domains: Pratola & Wolf (2003) fine-tune search-heuristic weights in the game Go via genetic algorithms [PW03], and Canaan et al. (2018) evolve rule sequences for agents using genetic algorithms [CST+18]. This shows the versatility of genetic algorithms in AI-based games based on heuristics. These studies collectively use genetic algorithms to systematically adjust heuristic weights in complex and strategic settings.

# 4 Computational Implementation

To analyze strategies in CuBirds, we developed a custom implementation of the card game in C++ [Boe25]. This section describes the key components of the implementation and the design decisions made during development.

To efficiently store information about an assembly of bird cards (e.g., the hand of a player or the draw pile), a fixed indexation of the bird cards was determined. This indexation can be found in Table 1. Bird cards are indexed by decreasing rarity, with index 0 assigned to the rarest species.

Table 1: Overview of the bird cards in CuBirds. Bird cards are indexed by decreasing rarity, with index 0 assigned to the rarest species. The small and large family sizes indicate how many cards are needed to collect that type of family. The final column shows the total number of cards for each species present in the game.

| Index | Bird Name | Small Family | Large Family | Total Cards |
|---|---|---|---|---|
| 0 | Flamingo | 2 | 3 | 7 |
| 1 | Owl | 3 | 4 | 10 |
| 2 | Toucan | 3 | 4 | 10 |
| 3 | Duck | 4 | 6 | 13 |
| 4 | Parrot | 4 | 6 | 13 |
| 5 | Magpie | 5 | 7 | 17 |
| 6 | Reed Warbler | 6 | 9 | 20 |
| 7 | Robin | 6 | 9 | 20 |

The implementation consists of three classes to carry out the game logic: the `Table` class, the `Player` class, and the `Game` class. These classes are used in the `main` function to be able to run the different methods and experiments used in this thesis. All three classes contain print, get, and set functions to facilitate debugging and easy use.
The `Table` class is responsible for managing the game table logic. It handles the draw pile, the discard pile, and the four rows of bird cards present on the table. The draw and discard pile are implemented as arrays of size eight, corresponding to the bird index shown in Table 1. The table layout itself is represented as an array of four vectors, one for each row, to reflect and support the fixed number of rows and the variable number and order of cards within each row. The `Table` class also contains the necessary constants, such as the number of rows on the table or the indexation used.
The `Player` class keeps track of the collection and hand of each of the players. Both the hand and the collection are an array of size eight using the indexation shown in Table 1. Each element in the array represents the number of birds belonging to the species indicated by its index. A player from the `Player` class can also be assigned different values for the reward function, which is discussed in Section 6.
The `Game` class manages the general flow of the game and the end-game logic. The `Game` class initializes a game, which includes initializing the rows on the table and dealing eight random cards to the hand and one card to each of the collections of the players. The `Game` class can run a game using one of the three decision-making methods described in Section 5.

# 5 Overview of the Three Decision-Making Methods

In this section, the three decision-making methods used to play CuBirds are described: the random method; the Flat Monte Carlo method; and the heuristic method. These three different methods are used to make all decisions during the turn of a player. The three decision-making methods are used and compared in the experiments described in Section 7.

## 5.1 Overview of the Random Method

The random decision-making method serves as a baseline for interpreting the results of the other two methods and enabling basic experimental analysis.
In each turn, the agent that uses the random decision-making method first selects a species from its hand at random. The chance of selecting a certain species from the hand is weighted by the number of cards of that species that are in the hand of the player. This ensures that every bird card in the hand of a player has an equal chance of being selected.
The agent then randomly picks one of the four rows on the table and a side of that row (left or right) on which to play the cards of the selected species of the hand.
After that the agent randomly picks between possible families that can be played and the option to play no family.

## 5.2 Overview of the Flat Monte Carlo Method

The Flat Monte Carlo decision-making method is implemented to create an agent that selects moves based on empirical outcomes, without relying on heuristic strategies. When deciding how to play a turn, the Flat Monte Carlo method first determines every possible legal move. The method then plays out a number of random games for each of these possible moves and counts how many games are won for each move. After this, the move with the highest win count is selected. The number of possible moves can be calculated using the following formula.

$$\text{possibleMoves} = S_{max_1} \cdot R \cdot D \cdot F = S_{max_1} \cdot R \cdot D \cdot (S_{max_2} + 1) = 8S_{max_1} \cdot (S_{max_2} + 1)$$

With:

- $S_{max_1}$: the number of unique bird species in the hand ($S_{max_1} \leq 8$),
- $R = 4$: the number of rows on the table,
- $D = 2$: the number of places bird cards can be played on a row (left or right side),
- $S_{max_2}$ : the number of unique bird species in the hand after the first phase of a turn ($S_{max_2} \leq 7$),
- $F = S_{max_2} + 1$: the number of possible families that can be played in the second phase of the turn plus the option of doing nothing.

This equation results in a maximum of 512 possible moves per turn. For each of these moves, *nRepeats* random games are played out. A win is counted only when a player completes their collection. This encourages more efficient gameplay by discouraging moves that only result in winning by depleting the draw pile. The first move that is found with the highest win count is carried out. The pseudocode of the algorithm for this method is described in Algorithm 1.

**Algorithm 1** Flat Monte Carlo Algorithm

```
1:  while game is not over do
2:      possibleMoves or ← get all legal moves
3:      for all move ∈ possibleMoves do
4:          wins ← 0
5:          for i = 1 to nRepeats do
6:              result ← play random game starting from move
7:              if result is a win then
8:                  wins ← wins + 1
9:              end if
10:         end for
11:         store win count for move
12:     end for
13:     execute first found move with highest win count
14: end while
```

## 5.3 Overview of the Heuristic Method

A custom heuristic method was developed to support strategic decision-making in CuBirds. For each turn, the method evaluates all possible moves using six weight factors in combination with the current game state parameters. Most weight factors adjust the contribution of specific components of the game state within the score function described in Section 5.3.7. This score function assigns a value to each possible set of cards that could be collected in phase 1d of Section 2.4. For every possible move, the resulting set of collected cards is scored, and the move which results in the highest-scoring set is selected as the optimal move. This section describes each weight factor, its role in the decision process, and how the parameters interact to determine the optimal move for a given game state.

### 5.3.1 Weight Factor $k$: Usefulness of the Card Set for Completing the Player's Collection

The weight factor $k$ adjusts the value of a given set of cards based on its direct usefulness to complete a player's collection. First, the relevance ($\alpha$) of each card $i$ in the set is determined: a card is assigned a value of 1 if it is needed to complete the player's collection, and 0 otherwise. Whether a card is needed depends on the player's current collection and the decided configuration of the winning collection: either seven unique bird cards or two sets of three identical bird cards. This goal is decided every turn by comparing the number of cards needed to complete a collection of two sets of three cards multiplied by $o$ to the number of cards needed to complete a collection of seven unique bird cards. The configuration that results in the lowest result is chosen as a goal. Factor $o$ is further explained in Section 5.3.5.
The total relevance score of the set is calculated by summing the values of all the collected cards and dividing by the total number of cards in the collected set of cards $c$, resulting in a value between 0 and 1 that represents the average usefulness of the set. Finally, the weight factor $k$, which can range from zero to one, scales this usefulness score to reflect its relative importance compared to other components of the final score function:

$$\text{Collection usefulness} = k \cdot \frac{1}{c} \sum_{i=1}^{c} \alpha_{b_i}$$

**Where:**

- $k$: weight factor for the importance of the cards for completing the player's own collection.
- $c$: number of cards collected.
- $\alpha_{b_i}$: direct usefulness of card $i \in \mathit{scoredCards}$, equal to 1 if the card is needed to complete the collection, and 0 otherwise.

### 5.3.2 Weight Factor $l$: Potential Usefulness of the Card Set for the Next Turn

The weight factor $l$ adjusts the value of a given set of cards based on its potential usefulness in the following turn. In CuBirds, it could be useful to plan one move ahead by collecting cards that might later be used to enclose desired bird cards. To evaluate this, the next turn is simulated with the current set of cards added to the player's hand. The maximum score achievable from the cards that could be collected in that simulated turn is then calculated. This score is based on three factors: the usefulness of the cards collected in the simulated turn, the rarity of the cards, and the number of cards collected. The calculation excludes the score for the player's subsequent turn and any potential gain for the opponent in its next turn to avoid infinite recursion. Finally, the weight factor $l$, which can range from zero to one, scales this usefulness score to reflect its relative importance compared to other components of the overall score function:

$$\text{Usefulness for player's next turn} = l \cdot T_{\text{future}}$$

**Where:**

- $l$: weight factor for the potential importance of the cards in the player's next turn.
- $T_{\text{future}}$: score obtained by simulating the player's optimal next turn.

### 5.3.3 Weight Factor $m$: Degree of Preference for Common versus Rare Birds

When evaluating a set of collected cards, the score function also considers the rarity of the cards. If the method strongly favors rare bird cards, it will attempt to collect two triplets of rare species (e.g., flamingos and owls). When aiming for a collection of seven unique birds, it will prioritize collecting rare birds first and focus on common birds later in the game. This strategic preference was incorporated into the score function through the rarity score described in the following.
First, a frequency vector was constructed that contains the number of cards available in the game for each species of birds (see Table 1):

$$A = \{7, 10, 10, 13, 13, 17, 20, 20\}.$$

To assign higher scores to rarer species, the vector $A$ was inverted by computing $\frac{1}{v}$ for each value $v \in A$. This resulted in values ranging from a maximum of $\frac{1}{7} \approx 0.1429$ for the rarest species (index 0) to a minimum of $\frac{1}{20} = 0.05$ for the most common species.
Next, the inverted values were normalized to the interval $[0, 1]$ using:

$$r_i = \frac{v_i - \min(V)}{\max(V) - \min(V)}$$

where $V$ is the vector of inverted values. This produced the normalized rarity score vector:

$$R = \{1, 0.538, 0.538, 0.290, 0.290, 0.095, 0.000, 0.000\}.$$

In the final scoring step, each card in the collected set is assigned its rarity score from $R$ and the total is averaged over the set of cards. The weight factor $m$ then scales this average to reflect the relative importance of rarity in the overall score function:

$$\text{Rarity score} = m \cdot \frac{1}{c} \sum_{i=1}^{c} r_{b_i}.$$

**Where:**

- $m$: weight factors for the preference for rare cards over common cards (or vice versa).
- $c$: number of cards collected.
- $r_{b_i} \in R$: normalized rarity score from vector $R$ for card $i \in \textit{collectedCards}$.

Like weight factors $k$ and $l$, the value of $m$ can range from zero to one to represent the strength of the preference for rare cards. However, to allow the method to also favor common birds, $m$ can take negative values. In this case, the score decreases for the rarer birds, rewarding the common species. Consequently, the range of $m$ is extended to $[-1, 1]$: If $m \geq 0$, the method prefers rare bird cards, while if $m < 0$, it prefers common bird cards. As with the other weight factors in the score function, the further $m$ deviates from zero, the stronger its influence on the total score.

#### 5.3.4 Weight Factor $n$: Potential Usefulness of the Card Set for the Opponent

The weight factor $n$ adjusts the value of the set of collected cards based on its potential usefulness against the opponent. In CuBirds, it could be beneficial to block the opponent from collecting cards that would complete their collection. To evaluate this, the usefulness of the set of cards collected against the opponent is scored. This score takes three factors into account: the usefulness of the cards collected for the collection of the opponent, the rarity of the cards, and the number of cards collected. The weight factor $n$, which can range from zero to one, then scales this score to reflect its relative importance compared to other components of the overall score function:

$$\text{Opponent's gain} = n \cdot T_{\text{opponent}}$$

**Where:**

- $n$: weight factors for the possible importance of the cards to the opponent.
- $T_{\text{opponent}}$: score the opponent would achieve if they received the enclosed cards.

#### 5.3.5 Weight Factor $o$: Degree of Preference for Seven Unique Birds versus Three Triplets

The weight factor $o$ does not directly contribute to the total score. Instead, it influences the choice of which winning collection configuration to pursue: either seven unique bird cards or two triplets of bird cards. As a standalone factor, $o$ describes the bias towards one collection configuration over the other. If $o < 1$, the method favors the collection of two triplets of birds. If $o = 1$, it has no preference. If $o > 1$, the method favors the collection of seven unique birds over two triplets. When the value of $o$ is close to 0 or 2, the preference is very strong; even if the alternative goal is nearly achieved, it will still try to pursue the preferred goal.

### 5.3.6 Weight Factor $p$: Usefulness of Collecting a Large Quantity of Cards

The weight factor $p$ adjusts the value of a given set of cards based on the number of cards collected. To assign a score between zero and one for the quantity of cards, the number of cards collected $c$ is divided by the total number of cards in the game, using a logarithmic scaling. To avoid calculating the logarithm of zero, one was added to both the numerator and the denominator. Finally, the weight factor $p$, which can range from zero to one, scales this usefulness score to reflect its relative importance compared to other components of the overall score function:

$$\text{Size bonus} = p \cdot \frac{\log(1+c)}{\log(111)}$$

**Where:**

- $p$: weight factors for the importance of collecting a large number of cards.
- $c$: number of cards collected.

### 5.3.7 Construction of the Final Score Function

To score a collected set of cards, all the previously mentioned components are combined into one scoring function. The final score of a collected set of cards is calculated as follows:

$$\text{Score} = \underbrace{k \cdot \frac{1}{c}\sum_{i=1}^{c} \alpha_{b_i}}_{\text{(1) Collection usefulness}} + \underbrace{m \cdot \frac{1}{c}\sum_{i=1}^{c} r_{b_i}}_{\text{(2) Rarity of cards}} + \underbrace{p \cdot \frac{\log(1+c)}{\log(111)}}_{\text{(3) Size bonus}} + \underbrace{l \cdot T_{\text{future}}}_{\text{(4) Player's next turn}} + \underbrace{n \cdot T_{\text{opponent}}}_{\text{(5) Opponent's gain}}$$

**Where:**

- $k, l, m, n, p$: weight factors from Table 2.
- $c$: number of cards collected.
- $\alpha_{b_i}$: direct usefulness of card $i \in \textit{scoredCards}$, equal to 1 if the card is needed to complete the collection, and 0 otherwise.
- $r_{b_i} \in R$: normalized rarity score from vector $R$ for card $i \in \textit{collectedCards}$.
- $T_{\text{future}}$: score obtained by simulating the player's optimal next turn.
- $T_{\text{opponent}}$: score the opponent would achieve if they received the enclosed cards.

Table 2: Overview of the six weight factors used in the scoring function of the heuristic method

| Weight Factor | Bounds | Description |
|---|---|---|
| $k$ | $[0,\ 1]$ | Importance of the cards for completing the player's own collection |
| $l$ | $[0,\ 1]$ | Potential importance of the cards in the player's next turn |
| $m$ | $[-1,\ 1]$ | Preference for rare cards over common cards (or vice versa) |
| $n$ | $[0,\ 1]$ | Potential importance of the cards to the opponent |
| $o$ | $[0,\ 2]$ | Bias towards seven unique species over two triplets |
| $p$ | $[0,\ 1]$ | Importance of collecting a large number of cards |

The weight factors $k, l, m, n, p$ are optimized using a genetic algorithm, ensuring that each component contributes to the total score in proportion to its strategic importance, as described in Section 6.

#### 5.3.8 Deciding the Second Phase of a Turn

Next, we move to the second phase of the turn, as described in Section 2.4. If possible, a bird card is collected that further completes the desired configuration of the collection. The method used to decide the move in the second phase of the turn is described in Algorithm 2. If the agent is aiming for a winning configuration consisting of seven unique birds, it will only try to collect bird cards that are not yet in the collection.
If the agent is aiming for a winning configuration consisting of two triplets, the agent first tries to complete a single bird in the collection by playing a big family of this kind. This completes a triplet. If that is not possible, it may play a small family if the opponent has a high risk of clearing their hand or attempt to start a new triplet.

**Algorithm 2** Family Collection Logic

```
Evaluate current collection: count species with 1, 2, and 3 bird cards
Compute distance to both win conditions (7 species vs. 2 triplets)
Use factor o to weigh preference for one of the goals
if weighted goal prefers 7 different species then
    for each bird species not yet in collection do
        Collect large/small family of this species if able
        if family is played then
            return
        end if
    end for
end if
if weighted goal prefers 2 triplets or no preference then
    // First try to complete a triplet by playing large family
    for each species with 1 in collection do
        Play large family if able
        if family is played then
            return
        end if
    end for
    // Check for risk of opponent clearing hand
    if opponent has few cards and game is progressing then
        Consider using small families early
    end if
    // Try to complete small family if at 2 cards or high risk of discarding
    for each species with 2 in collection or high risk do
        Play small family if able
        if family is played then
            return
        end if
    end for
    // Add missing species to start new family or if others are hard to complete
    if only one species in collection or most cannot be completed then
        for species not in collection do
            Play large/small family if able
            if family is played then
                return
            end if
        end for
    end if
end if
```

# 6 Optimizing the Heuristic Score Function using a Genetic Algorithm

In Section 5.3, a heuristic method is described as a decision-making method in the game CuBirds. This heuristic method uses a scoring function to assign a score to a set of cards that can be collected. The score of the set of cards represents the desirability of this set of cards for a specific state of the game. The function considers several aspects, such as the number of cards, the rarity of the species, and the current collection of a player. Each of these factors contributes differently to the final score. For example, card rarity may have a greater influence on the desirability of a move than the quantity of cards gained. In this way we can try to find strategies that gives an advantage when playing CuBirds. To determine the optimal weighting of these aspects, a genetic algorithm is employed. This approach allows the weight factors of the score function to adapt based on the relative importance of each factor, resulting in an improved decision-making method. When the weights of the factors are calculated using the genetic algorithm, they are assigned to a player and remain static throughout the game.
A genetic algorithm is an algorithm inspired by the process of natural selection. It evolves a population of candidate solutions over multiple generations, using operations such as selection, mutation and cross-over. Over the span of multiple generations, the solutions converge towards a set of solutions that perform well according to a predefined fitness function [Gol89].

## 6.1 Overview of the Genetic Algorithm

The genetic algorithm maintains a population of $N$ individuals, each defined by weight vector $w = (k, l, m, n, o, p)$. In each generation, every individual plays against $M$ randomly selected opponents from the current generation population. The individual's win count in these matches determines its fitness.
To produce the next generation, two parents are selected at random from the top 50% of the population based on their fitness. These parents are combined to create a new individual that then undergoes a mutation. A decaying mutation rate is applied to balance exploration in early generations with convergence in later ones, reducing the risk of becoming trapped in local optima [Bä96]. This evolutionary process continues for a predefined number of generations. The complete procedure is summarized in Algorithm 3. In the experiments $N = 200$, $M = 100$ and $G = 100$.

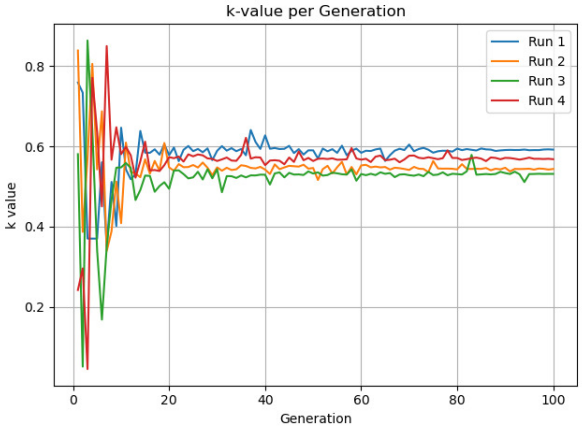
## 6.2 Results of Tuning the Heuristic Weight Factors

To test the optimal balance of the different weight factors, the genetic algorithm is used to create the optimal set of weight factors for the heuristic method. If one of the factors of $k$, $l$, $m$, $n$ or $p$ is relatively higher than the other weight factors, it is relatively more important to take this factor into account when playing CuBirds. For example, if $p$ is relatively higher, it is more important to take into account the number of cards collected with a certain move than it is to take into account the other components of the scoring function.

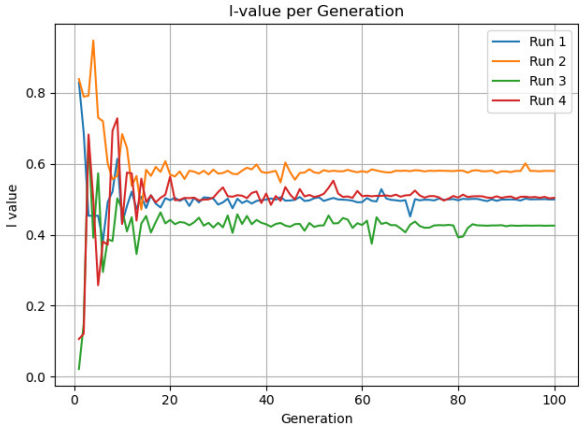
Figure 5 shows the convergence of the six weight factors. The graphs show a convergence to a specific value. From these values, several conclusions can be drawn. The factors $l$ and $p$ both converge to approximately 0.5. For convenience, this value is used as a reference point to interpret the relative importance of the other weight factors.
The weight factor $k$ converges to a slightly higher value than $l$ and $p$, suggesting that it is somewhat more important to prioritize the usefulness of cards for a player's own collection.
The weight factor $n$ converges to a slightly lower value, indicating that taking into account the opponent's possible score when evaluating the collected cards is slightly less important.
The weight factor $o$ converges to around 1.8, showing a strong preference for collecting seven unique bird cards over completing two sets of three bird cards.
The weight factor $m$ converges to a small positive value. This indicates a preference for rare cards over common ones. The small value indicates that this component is less important for the final score.

**Algorithm 3** Genetic Algorithm for Optimizing Weight Factors

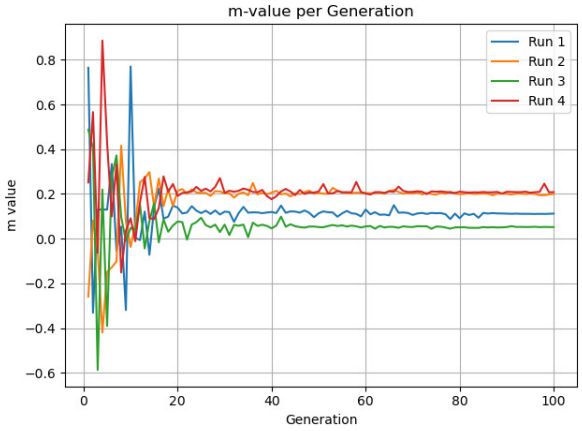
Initialize population of $N$ individuals with random weight vectors $(k, l, m, n, o, p)$
G = total number of generations
**for** each generation $g$ **do**
    Set mutation rate $\mu = \mu_0 \cdot \left(1 - \frac{g}{G}\right)$
    **for all** individuals $p$ in population **do**
        $wins \leftarrow 0$
        **for** each of $M$ random opponents from population **do**
            **for** $i = 1$ to $nRepeats$ **do**
                Play a game with $p$ as Player 1 and opponent as Player 2
                **if** $p$ wins **then**
                    increment $wins$
                **end if**
            **end for**
        **end for**
        Compute fitness of $p$ as: $\frac{wins}{M \cdot nRepeats}$
    **end for**
    Sort population by fitness (descending)
    Retain top 50% as elites
    **while** nextGeneration not full **do**
        Select two parents from elite pool
        Generate child via gene-wise average crossover and mutation:
        $x_i^{\text{child}} = \text{clamp}\left(\frac{x_i^{\text{parent1}} + x_i^{\text{parent2}}}{2} + \delta, \text{bounds}_i\right)$
        where $\delta \sim \mathcal{U}(-0.05, 0.05)$ with probability $\mu$
        Add child to next generation
    **end while**
    Replace population with new generation
**end for**
Return best individual found



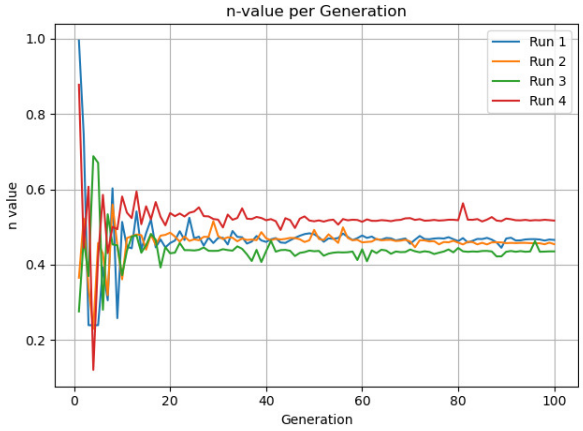

(a) Weight factor $k$: weight factor for prioritizing own collection. $k$ converges to $\approx 0.55$.



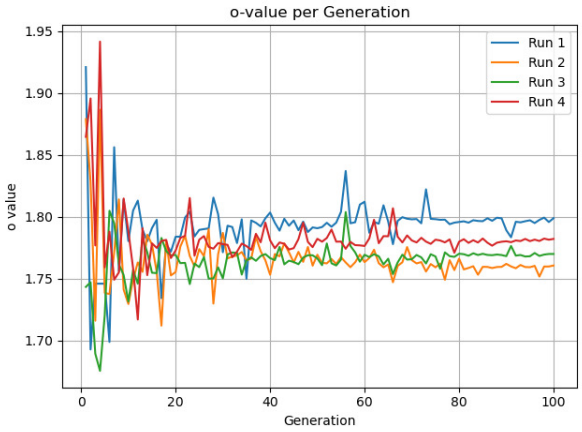

(b) Weight factor $l$: weight factor for looking one turn ahead. $l$ converges to $\approx 0.50$.



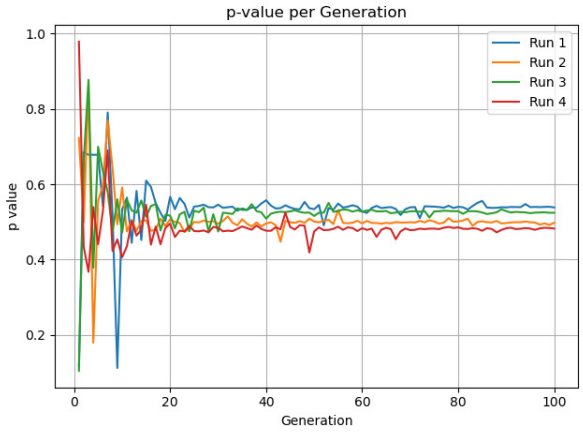

(c) Weight factor $m$: weight factor for favoring rare cards. $m$ converges to $\approx 0.10$.




(d) Weight factor $n$: weight factor for considering possible opponent score. $n$ converges to $\approx 0.50$.




(e) Weight factor $o$: bias toward 7 unique species over 2 triplets. $o$ converges to $\approx 1.80$.




(f) Weight factor $p$: weight factor for collecting larger card quantities. $p$ converges to $\approx 0.50$.

Figure 5: Evolution of weight factor values over generations in the genetic Algorithm. Each subplot corresponds to one of the six weight factors.

# 7 Experiments and Results

## 7.1 Basic Gameplay Analysis

To analyze different aspects of the game, three different decision-making methods are tested: random, Flat Monte Carlo and a heuristic method. The weight factors for the heuristic method are set to the optimal values found in Section 6.2 in Figure 5 ( $w = (k; l; m; n; o; p) = (0.55; 0.50; 0.10; 0.50; 1.80; 0.50)$. Every method is tested by running 1,000 games, in which both agents use the same method. Table 3 shows the results for the random method. The average number of turns per game is 77.36 and the average time per turn is 0.00013 seconds. Table 4 shows the results for the Flat Monte Carlo method. The average number of turns is 53.19 and the average time per turn is 338.40 seconds. Table 5 shows the results of the heuristic method. The average number of turns per game is 17.04 and the average time per turn is 0.14 seconds.
The following sections describe the conclusions that can be drawn based on these results.

Table 3: Results of 1,000 random games of CuBirds. The table shows the number of wins per win condition for each player, as well as ties and totals per row and column.

| Win Condition | Player 1 | Player 2 | Tie | Total |
|---|---|---|---|---|
| 2x3 | 145 | 140 | 0 | 285 |
| 7 | 51 | 55 | 0 | 106 |
| Empty Deck | 272 | 263 | 74 | 609 |
| **Total Wins** | **468** | **458** | **74** | **1,000** |

Table 4: Results of 1,000 Flat Monte Carlo-based games of CuBirds. The table shows the number of wins per win condition for each player, as well as ties and totals per row and column.

| Win Condition | Player 1 | Player 2 | Tie | Total |
|---|---|---|---|---|
| 2x3 | 264 | 282 | 0 | 546 |
| 7 | 212 | 233 | 0 | 445 |
| Empty Deck | 2 | 4 | 3 | 9 |
| **Total Wins** | **478** | **519** | **3** | **1,000** |

Table 5: Results of 1,000 heuristic-based games of CuBirds. The table shows the number of wins per win condition for each player, as well as ties and totals per row and column.

| Win Condition | Player 1 | Player 2 | Tie | Total |
|---|---|---|---|---|
| 2x3 | 10 | 8 | 0 | 18 |
| 7 | 504 | 468 | 0 | 972 |
| Empty Deck | 5 | 5 | 0 | 10 |
| **Total Wins** | **519** | **481** | **0** | **1,000** |

### 7.1.1 Advantage of the starting Player

To assess whether a starting player has a statistical advantage in CuBirds, a two-sided binomial test is performed on the data in Section 7.1. The win counts of Player 1 and Player 2 for each decision-making method are taken into account, excluding tied games.
For the random agent, Player 1 won 50.5% of the games while Player 2 won 49.5%. The resulting p-value of 0.767 indicates that there is no statistically significant difference, suggesting that random play does not favor either player.
Similarly, for the Flat Monte Carlo agent, Player 1 won 47.9% of games compared to 52.1% for Player 2, with a p-value of 0.205. Although Player 2 won slightly more often, this difference is not statistically significant at the 0.05 level.
The heuristic agent shows a reverse pattern with respect to the Flat Monte Carlo agent: Player 1 won 51.9% of the games compared to 48.1% for Player 2, with a p-value of 0.242, which is also not statistically significant.
In general, these results do not indicate evidence of a first- or second-player advantage in any of the methods tested, which suggests the starting player does not have an advantage in the game CuBirds.

### 7.1.2 Game length

Table 6 presents the average time per turn and the average number of turns for each decision-making method. These results provide insight into the efficiency of the agents. The random agent spends the lowest average time per turn compared to the other two agents but requires the highest number of turns to complete a game.
The Flat Monte Carlo method, on the other hand, has the highest computational cost per move, this is mainly due to the large number of playouts per move, namely 1,000 per possible move on a turn. However, the method completes games in fewer turns than the random agent.
The heuristic agent achieves a balance between the two: it requires significantly less time per move than the Flat Monte Carlo method and completes games in the fewest turns on average.
Based on the average game length, it can be concluded that the heuristic agent offers the most efficient trade-off between move quality and computational cost.

Table 6: Average time per turn and average number of turns per game for each agent.

| **Agent** | **Avg. Time per Turn (s)** | **Avg. Turns per Game** |
|---|---|---|
| Random | 0.0001 | 77.3564 |
| Flat Monte Carlo | 338.3955 | 53.1940 |
| Heuristic | 0.1416 | 17.0380 |

### 7.1.3 Configuration of Winning Collection

An agent using a random method wins mostly through two sets of three birds rather than seven unique birds. From this, the conclusion could be drawn that when no strategy is applied, it is more likely to win with two sets of three bird cards than to win with seven unique bird cards.
The Flat Monte Carlo agent shows a nearly even split between the two ways to create a collection. This suggests that by planning ahead it is nearly as likely to win with two sets of three bird cards as it is with seven unique bird cards.
In Section 6.2, it was already concluded that the heuristic player wins more often if it focuses on a collection of seven unique bird cards. This shows that when applying a certain strategy, it is more profitable to focus on seven unique bird cards than on two sets of three bird cards.

## 7.2 Effect of Starting Bird Card on the Winning Rate

In order to investigate the effect of the starting bird card in a player's initial collection, the starting bird of the collection of both players is initialized to a specific bird. The effect of the starting bird card is analyzed using all three decision-making methods.

### 7.2.1 Effect of Starting Bird Card with the Random Method

Table 7 shows the results of playing 1,000 games using the random method with different starting bird cards.
To test whether the starting bird has an effect on the win count for either player 1 or 2, a chi-squared test is performed on the data in Table 7. This results in a chi-squared statistic of 0.203 with 49 degrees of freedom and a p-value of 1.0. From these values it is concluded that the starting bird of a player has no significant effect with the random decision-making method.

Table 7: Win count results of playing 1,000 random games per starting bird for P1/P2. On the horizontal axes every column corresponds to a different starting bird for player 2 (using the indexation from Table 1). On the vertical axes every row corresponds to a different starting bird for player 1 using the same indexation. Every cell first states the amount of times player 1 won, then the amount of times player 2 won, the remaining games ended in a tie.

| Starting Bird | P2:0 | P2:1 | P2:2 | P2:3 | P2:4 | P2:5 | P2:6 | P2:7 |
|---|---|---|---|---|---|---|---|---|
| P1:0 | 463/464 | 464/464 | 465/464 | 466/463 | 466/463 | 463/467 | 465/464 | 463/465 |
| P1:1 | 464/464 | 465/464 | 465/465 | 468/462 | 468/462 | 466/464 | 469/461 | 468/463 |
| P1:2 | 463/465 | 466/465 | 463/465 | 465/466 | 468/462 | 466/465 | 468/463 | 468/464 |
| P1:3 | 463/466 | 460/470 | 462/466 | 465/465 | 465/465 | 464/467 | 466/463 | 465/465 |
| P1:4 | 464/465 | 464/466 | 464/467 | 465/465 | 465/465 | 463/467 | 467/465 | 464/467 |
| P1:5 | 464/466 | 464/466 | 461/468 | 466/464 | 467/464 | 465/465 | 467/463 | 468/462 |
| P1:6 | 463/465 | 460/470 | 464/466 | 463/467 | 464/465 | 462/469 | 467/464 | 465/465 |
| P1:7 | 466/464 | 462/470 | 461/470 | 464/466 | 465/466 | 462/470 | 462/469 | 465/466 |

### 7.2.2 Effect of Starting Bird Card with the Flat Monte Carlo Method

Table 8 shows the results of playing 1,000 games using the Flat Monte Carlo method with different starting bird cards. Although the win count does seem to deviate between different cells in the table, no specific patterns can be found. To illustrate the data in Table 8, a corresponding heatmap is shown in Figure 6.

Table 8: Win count results of playing 1,000 Flat Monte Carlo games per starting bird for P1/P2.

| Starting Bird | P2:0 | P2:1 | P2:2 | P2:3 | P2:4 | P2:5 | P2:6 | P2:7 |
|---|---|---|---|---|---|---|---|---|
| P1:0 | 448/552 | 507/474 | 531/457 | 526/474 | 527/461 | 556/444 | 521/467 | 551/449 |
| P1:1 | 554/446 | 502/498 | 491/509 | 412/588 | 456/544 | 580/420 | 498/502 | 503/497 |
| P1:2 | 606/394 | 481/519 | 506/494 | 462/526 | 597/403 | 423/577 | 439/561 | 547/453 |
| P1:3 | 470/530 | 541/459 | 593/407 | 429/571 | 522/466 | 481/519 | 543/457 | 584/404 |
| P1:4 | 516/484 | 410/590 | 503/497 | 508/492 | 490/510 | 440/560 | 513/487 | 482/518 |
| P1:5 | 517/483 | 545/455 | 520/480 | 468/532 | 531/469 | 417/583 | 489/511 | 525/475 |
| P1:6 | 580/420 | 475/525 | 507/493 | 505/495 | 603/397 | 482/518 | 500/500 | 556/444 |
| P1:7 | 449/551 | 468/532 | 592/408 | 453/547 | 477/523 | 502/498 | 465/535 | 454/546 |

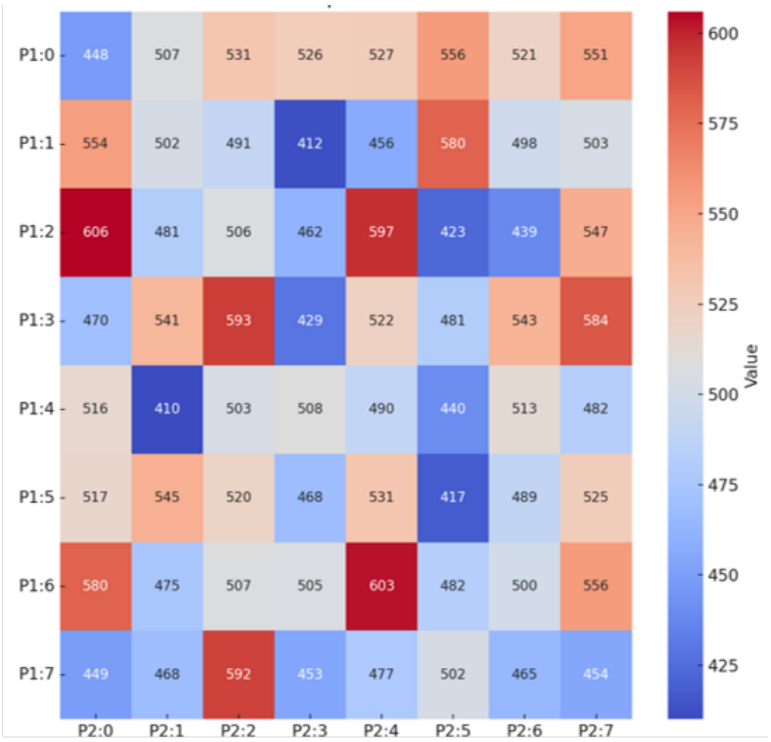



Figure 6: A heatmap of the data from Table 8. Every cell contains the wincount of Player 1.

### 7.2.3 Effect of Starting Bird Card with Heuristic Method

Table 9 shows the results of the win rate for different starting bird cards with a heuristic decision-making method. To illustrate the data in Table 9, a corresponding heatmap is shown in Figure 7.

Table 9: Win count results of playing 1,000 heuristic games per starting bird for P1/P2.

| Starting Bird | P2:0 | P2:1 | P2:2 | P2:3 | P2:4 | P2:5 | P2:6 | P2:7 |
|---|---|---|---|---|---|---|---|---|
| P1:0 | 512/480 | 508/492 | 457/540 | 493/505 | 488/510 | 454/543 | 457/542 | 490/509 |
| P1:1 | 505/492 | 460/537 | 496/503 | 503/496 | 507/492 | 496/501 | 465/532 | 478/520 |
| P1:2 | 505/493 | 525/472 | 506/491 | 501/497 | 485/514 | 488/508 | 498/501 | 506/491 |
| P1:3 | 546/452 | 513/486 | 523/475 | 523/475 | 493/504 | 519/480 | 494/506 | 479/521 |
| P1:4 | 547/452 | 507/493 | 508/490 | 510/490 | 510/489 | 514/484 | 511/489 | 468/529 |
| P1:5 | 535/460 | 510/488 | 513/485 | 510/484 | 503/497 | 522/475 | 501/499 | 505/494 |
| P1:6 | 582/416 | 535/463 | 531/467 | 511/488 | 535/463 | 523/476 | 506/492 | 505/493 |
| P1:7 | 555/443 | 525/474 | 532/467 | 499/500 | 503/493 | 528/472 | 505/493 | 480/517 |

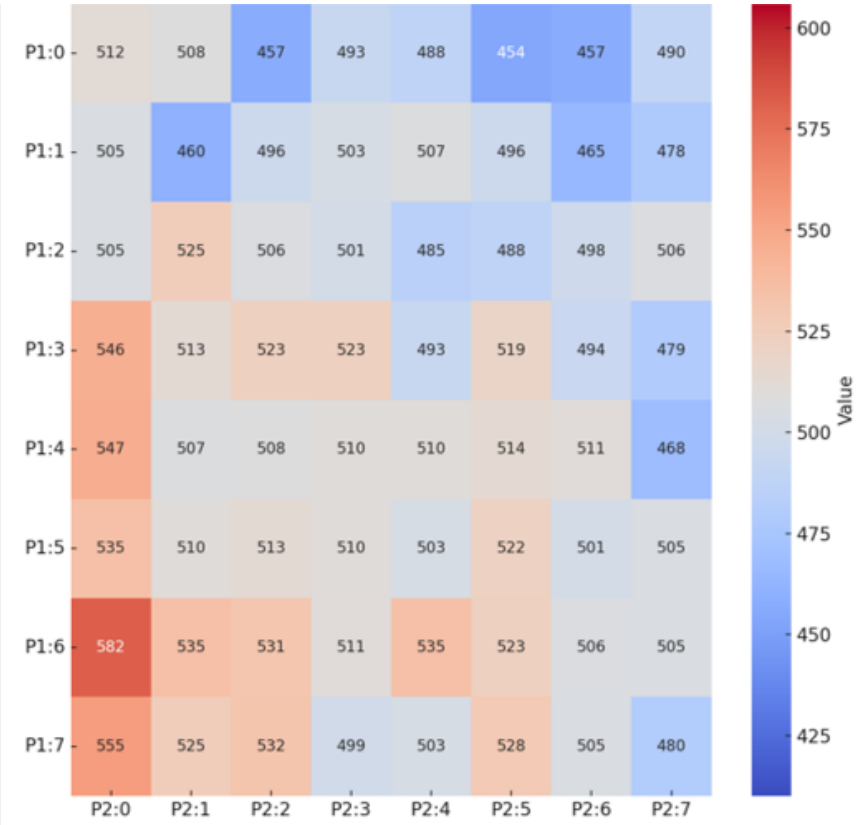



Figure 7: A heatmap of the data from Table 9. Every cell contains the wincount of Player 1.

An analysis of the strength of the starting bird shows that certain starting bird cards offer a statistically significant advantage. A chi-square test confirms that the win distributions differ significantly by starting bird for player 1 and player 2 ($p = 0.0009$). This means that the starting bird matters. Residual and Z-score analysis further identify bird 6 as the strongest starting option for both players, with win counts well above the expected average and Z-scores of +2.79 (P1) and +1.91 (P2). Bird 7 and bird 5 also consistently perform above expectations. In contrast, bird 0 is a notably weak starting bird, with significantly fewer wins than expected ($Z = –3.01$ for P1 and –3.93 for P2). These findings indicate that common bird cards provide a measurable strategic advantage at the start of the game.

Using a chi-square test on the tie counts in all 64 matches, a p-value = 0.9999 is found. This means that the starting bird does not affect the number of ties.

# 8 Conclusions and Further Research

In this thesis, we analyzed CuBirds using three decision-making agents to answer two main research questions: does a certain starting position offer a measurable advantage to a player and are there particular game strategies that are especially effective in a game of CuBirds? Sections 6.2 and 7 provide valuable information on the performance of different strategies in CuBirds. By tuning the weight factors with the genetic algorithm, the conclusion can be drawn that a good strategy is to balance a score that features key components of the state of the game to make optimal decisions. This optimal balance consists of focusing on the cards that are needed to complete the collection. Almost as important is to take into account the possible importance of the cards for the next turn and the quantity of the cards. Slightly less important is checking the desirability of the cards for the opponent. It is of minimal importance to take into consideration the rarity of the cards.
The experiments show that a random player wins about 20 percentage points more by completing a collection of two triplets over a collection of seven unique bird cards. For the Flat Monte Carlo method, this differs by about ten percentage points. However, when tuning with a genetic algorithm, the heuristic method displays a strong preference for collecting seven unique bird cards over two triplets.
The results also show that the starting player does not have an advantage and a player that starts with a common bird card (e.g. a reed warbler) in its collections does have a slight advantage.

Building on the implementation and experiments described in this thesis, several opportunities for further exploration are identified. We list some directions in more detail below.

**Dynamic weight factors in the Genetic Algorithm.** The current reward function uses fixed weight factors throughout the game. Another approach could be to change the weight factor based on the state of the game. For example, focus more on the opponent at the end of the game than at the start of the game. Another dynamic component that could be added is to choose a configuration goal at a certain point in the game, for example after a set amount of turns or when a set gamestate is reached.

**Monte Carlo Tree Search (MCTS) instead of Flat Monte Carlo.** The current approach treats all legal moves equally. Implementing Monte Carlo Tree Search (MCTS) would allow for more efficient exploration of the most promising move sequences, especially when combined with a good evaluation function or learned policy.

**Support for More Than Two Players.** The current simulation framework is limited to two players. Extending the implementation to handle three or more players would allow analysis of strategic dynamics in larger games, where opponent modeling and turn order may have a more significant impact.

**Parallelization for Performance Optimization.** In most experiments, multiple games are run sequentially to obtain results. These games are independent of each other and could be run in parallel to improve performance. Using multi-threading or GPU acceleration would significantly reduce training and evaluation time, making it feasible to perform larger simulations.

# References


[Ale18] Stefan Alexander. Cubirds. BoardGameGeek, 2018. boardgamegeek.com/boardgame/245476/cubirds.

[Bak24] Mika Bakker. Creating agents for the card game rikken. Bsc thesis, Leiden Institute of Advanced Computer Science (LIACS), Leiden University, Leiden, Netherlands, 2024.

[Bla20] Marijn Blaauw. Strategy analysis in card games using monte carlo simulation. Bachelor's Thesis, Utrecht University, 2020.

[Boe25] Noëlle Boer. Cubirds: C++ implementation for strategy analysis. https://github.com/NoelleBoer/CuBirds, 2025. Accessed: 2025-05-22.

[BPW+12] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[Bä96] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, 1996.

[Cat25] Catch Up Games. CuBirds Rulebook, 2025. Accessed: 2025-02-17.

[CST+18] Rodrigo Canaan, Haotian Shen, Ruben Rodriguez Torrado, Julian Togelius, Andy Nealen, and Stefan Menzel. Evolving agents for the hanabi 2018 cig competition. *arXiv preprint arXiv:1809.09764*, 2018.

[God22] Lukasz Godlewski. *Monte Carlo Tree Search and Reinforcement Learning Methods for The Lord of the Rings: The Card Game*. Phd thesis, AGH University of Science and Technology, 2022.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[GSTFLC24] Pablo García-Sánchez, Alberto Tonda, Antonio J. Fernández-Leiva, and Carlos Cotta. Optimizing hearthstone agents using an evolutionary algorithm. *arXiv preprint arXiv:2410.19681*, 2024.

[Kel16] Joseph Kelly. Comparison of monte carlo tree search methods in the imperfect-information game cribbage. Honors thesis, University of Tennessee, 2016.

[KM20] Jakub Kowalski and Radosław Miernik. Evolutionary approach to collectible card game arena deckbuilding using active genes. *arXiv preprint arXiv:2001.01326*, 2020.

[MGPL+20] Raul Montoliu, Raluca D. Gaina, Diego Perez-Liebana, Daniel Delgado, and Simon M. Lucas. Efficient heuristic policy optimisation for a challenging strategic card game. In *International Conference on the Applications of Evolutionary Computation (EvoStar)*, volume 12104, pages 403–418. Springer, 2020.

[MK22] Radosław Miernik and Jakub Kowalski. Evolving evaluation functions for collectible card game ai. In *14th International Conference on Agents and Artificial Intelligence (ICAART)*, volume 3, pages 253–260, 2022.

[PW03] Matthew Pratola and Thomas Wolf. Optimizing gotools' search heuristics using genetic algorithms. *arXiv preprint cs/0302002*, 2003.

[Sha03] Alexander Shapiro. Monte carlo sampling methods. In *Handbooks in Operations Research and Management Science*, volume 10, pages 353–425. Elsevier, 2003.

[Spe24] Spellenhuis. CuBirds Card Game Image, 2024. Accessed: 2025-02-17.

[Yuc25] Yucata. CuBirds Flamingo Card Image, 2025. Accessed: 2025-02-17, Image modified by the author for illustrative purposes.

[ZR15] Alexander Zook and Mark O. Riedl. Monte carlo tree search for simulation-based strategy analysis. In *Proceedings of the 10th International Conference on the Foundations of Digital Games (FDG)*, 2015.