Okey and Random Combinatorial Games

Nuray Kutlu

October 2025

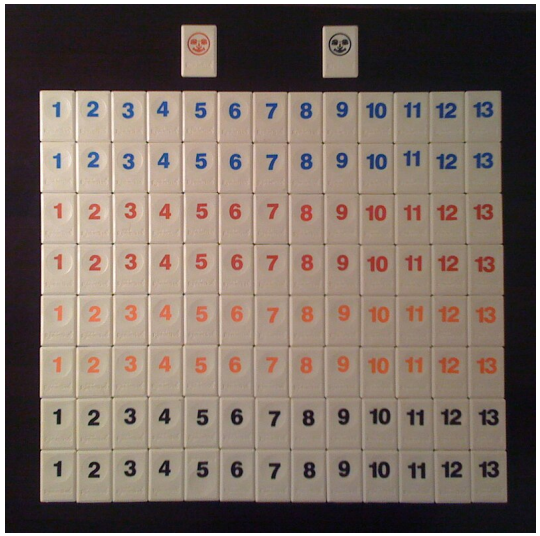
Table of Contents

1. The Game of Okey
2. Experimenting with Generalized Okey
3. Random Combinatorial Games
4. TicTacToe
5. Statistical Analysis and Evolution

The Game of Okey

The Game of Okey: Setup

- Okey is a Turkish tile game, similar to the game Rummikub.
- There are 106 tiles in the game: 104 numbered tiles (numbered 1–13 in 4 different colors, with 2 copies of each tile) plus 2 jokers.
  - Normally, there's a process to pick a number as a joker and replace the joker tiles with that number — for our sake, we can ignore this.
- The game is played with 4 players. One player receives 15 tiles while the rest of the players receive 14, *completely at random*.
  - There's also a process to randomly distribute the tiles, but we can ignore this too for our sake.



The Game of Okey: Gameplay

- The player with the 15 tiles starts.
- Every turn, the player discards a tile and puts it in a stack to their right. The next player can choose to take that tile, if it suits their board, or pick a random tile from the middle.
- During each turn, a player's objective is to take a tile (either from the middle or from the previous player) and form runs (a sequence of three or more tiles in the same color) or sets (three or more tiles of the same number and different colors) with their tiles.
- If a player manages to put 14 tiles into runs or sets on their turn, they throw the 15th tile in the middle and win the game. If not, they discard one tile into the pile to their right, and the game continues.

Experimenting with Generalized Okey

Generalized Okey: Setup

- Okey is normally played with 106 tiles, 4 players, and 14 tiles per player. This requires a lot of intensive computation to simulate. Instead, we can generalize both the number of players and the tile deck:
  - We can define a general tile deck as follows: there are r different colors (suits) and k different numbers (ranks), with two of each tile and two jokers. This means the deck is of size $r \times k \times 2 + 2$.
  - We call the number of players p.
  - We call the number of tiles per player s.
- Generalizing Okey also allows us to experiment with different types of game states with pre-defined rules!

Generalized Okey: Game Simulation

- In order to simulate a Generalized Okey game, we need:
  - A game state: A random dealing of tiles to p players, along with middle tiles, current player, and any discarded tiles (initially none).
  - Functions to determine which tile to take/discard, a function which determines whether a player can put all their tiles into runs and sets (a winning state), and a function which determines which tile to discard.
- The Maple package Okey.txt has functions to initialize a game state (take in r, k, s, p parameters and randomly deal tiles), find runs and sets, decide between picking a random tile or a newly discarded tile, determine winning states, and simulate an entire *random game of Okey*.
- The simulation returns the winning player, along with the number of moves played. If there are no middle tiles left and no winner, this is considered a tie.

Generalized Okey: Analyzing Games using Game States and Markov Chains

- The approach was to model Okey as a Markov chain where each hand is a state.
- The function **GenerateStates(r, k, s):** creates all possible hands with r colors, k numbers, s tiles.
- The function **TerminalStates(r, k, gs, s):** identifies winning states (hands with 0 ungrouped tiles).
  - These become END states in the Markov chain (absorbing states).

Generalized Okey: Analyzing Games using Game States and Markov Chains (cont.)

- **StateProbabilities(r, k, s)**: Computes the initial probability vector V_0
- **ChildrenStates(S, States, r, k, gs):** For state S, find all reachable next states.
  - All reachable next states: remove each ungrouped tile, add each available tile from deck.
- **TransitionMatrix(r, k, gs, s):** Build matrix **M** where $M_{ij} = P(\text{state } j \mid \text{state } i)$.
  - Also, all terminal states are collapsed into one absorbing END state.
- **TransitionMul(V, M):** This function does the matrix multiplication on the vector V and the matrix M: $\mathbf{V}_{n+1} = \mathbf{V}_n \cdot \mathbf{M}$.
- Track winning probability over time: $\mathbf{V}_n[T]$.
- We realized that a game as complicated as Okey needs tons of computation to analyze, even when generalized to smaller games! So, we took our experimentation over to Random Combinatorial Games, along with TicTacToe.

Random Combinatorial Games

Random Combinatorial Games: Definition and Set-Up

- A Random Combinatorial Game consists of:
  - n states (numbered 1 to n)
  - k players, each assigned a subset of states
  - Random transitions defined between states
  - Random winning states for each player
- **RandomCG(n,k)** generates a Random Combinatorial Game for n states and k players:
  - Divides n states (numbered 1 to n) roughly equally among k players. Picks one state as an initial state.
  - Generates random transitions between states.
  - Generates win states as subsets of each player's states.
  - Generates tie states as a subset of states that are not in any player's winning states.
- The output for **RandomCG(n,k)** is of the format [Initial State, WinStates, Table of States/Transitions].

TicTacToe

Generalized TicTacToe: Definition and Set-Up

- A Generalized TicTacToe game is a $k \times k$ board where players aim to get k in a row.
- The package CombGames.txt has a function **TicTacToe(k)** that generates a complete TicTacToe game for a $k \times k$ board:
  - Starts with an empty $k \times k$ board as the initial state represented as a matrix of zeros.
  - Systematically generates all possible valid board configurations and stores transitions.
  - Stores win states by checking if a player has achieved k consecutive marks in any row or column.
  - Stores tie states (fully filled boards with no winner).
- **TicTacToeD(k)** is a variant that also includes diagonal wins.

Statistical Analysis and Evolution

Random Combinatorial Games: Statistical Analysis

- **SimulateGameT(G,n,k)** simulates $G := \text{RandomCG}(n, k)$ and returns [Winner, Game Duration].
- **SimuStatDuration(G,n,k,N,t)** runs **SimulateGameT** N times and returns the average as a polynomial:

$$\frac{1}{N} \sum_{i=1}^{N} t^{\text{duration}_i}$$

  where duration_i is the length of the ith simulated game, and the coefficient of t^d is the average frequency of games lasting d moves.
- **SimuStatWin(G,n,k,N)** runs **SimulateGameT** N times and returns probabilities for each player's win outcomes.

Random Combinatorial Games: Game Evolution

- *"What is the probability that player 2 wins on move 5?"*
- **Oper1(G, i, X)** for G:= RandomCG(n,k) will output a linear combination of states reachable from state i.
- **Oper(G, X)** gives us a list, where each element is the evolution (linear combination of states reachable) of each state. It's the "transition operator".

$$T = [T_1, \ldots, T_n], \qquad T_i := \text{Oper1}(G, i, X).$$

- **ApplyOper(T, f, X)** applies T to a linear combination f. If f is a linear combination of states $f = \sum_{i=1}^{n} a_i X_i$, then you substitute each X_i by $T_i(X)$.

$$\text{ApplyOper}(T, f, X) = \sum_{i=1}^{n} a_i \, T_i(X).$$

- **EvolutionGameT(G,n,k,N,x)** returns a table of all probabilities that player i in T[i] will win on a given turn N by applying the Operator repeatedly N many times.

Random Combinatorial Games: Generating Functions

- We can also use generating functions to do similar analysis.
- **GenFunDuration(G,k,N,x)** returns a polynomial $\sum_{i=1}^{N} p_i \cdot x^i$ where the coefficient p_i of x^i is the probability that the game ends exactly after i rounds.
- **GenFunWin(G,n,k,N,x,r)** for a specific player r returns a polynomial $\sum_{i=1}^{N} p_i \cdot x^i$ where the coefficient p_i of x^i is the probability that player r wins after exactly i moves.
- These functions use the evolution operator and encode probabilities as polynomial coefficients.

Thank You!

Thank you all for listening and for your interest in my project!

A special thanks to Dr.Z for working with me on this project and inviting me to speak at the Experimental Mathematics Seminar.