



Lesson: Mathematical Analysis and Model Design for Logic and Puzzle Games

Pathum Sookaromdee¹, Viroj Wiwanitkit²

¹Private Academic Consultant, Bangkok, Thailand.

²Joseph Ayobabalola University, Ikeji-Arakeji, Nigeria.

Abstract

Mathematical logic and puzzle games provide structured environments for analyzing problem-solving strategies and the underlying principles that govern them. This article examines the essential components of mathematical puzzle games, including game states, transitions, constraints, and solution objectives. It further explains the mathematical foundations behind these games-such as graph theory, invariants, combinatorics, and algebraic models-and demonstrates how these principles shape solvability and strategy. Finally, a systematic framework for designing mathematical solution models is presented, offering methods for representing game mechanics, identifying solvable structures, and constructing efficient algorithms. This integrative approach supports deeper understanding of both the theory and practice of mathematical puzzle design and analysis.

Keywords: Mathematical Puzzles, Problem-Solving Models, Game Structure Analysis.

Components of Mathematical Logic Games

When analyzing a puzzle from a mathematical perspective, the game can be decomposed into fundamental components:

State (Configuration)

A formal representation of the game's current position.

Examples:

- Arrangement of tiles in a 15-puzzle
- Number of stones in each pile in a Nim game
- Sudoku board configuration

Actions / Transitions (Moves)

Rules that govern how one state can transform into another.

Initial and Goal States

Starting configuration and the target condition for success.

Objective / Cost Function

What the game aims to optimize:

Minimum number of steps

Maximum score

Constraints

Restrictions on allowed moves (e.g., no repeated numbers in a Sudoku row).

State Space

A graph in which:

- Nodes = states
- Edges = moves

Structural Properties

Invariants, symmetry, parity, monotonicity—properties that remain unchanged.

Game Type

Deterministic vs. stochastic, single-player vs. two-player, impartial vs. partisan—determines appropriate mathematical tools.

Mathematical Principles Behind Logic Games

Below are common mathematical foundations used to analyze puzzle games:

Graph Theory & Search Algorithms

State space as a graph →

Use BFS, DFS, Dijkstra, or A for finding shortest solutions.

Heuristics (admissible) are essential for efficient A search.

Invariants & Proof Techniques

Finding properties that never change (parity, symmetry) helps prove:

A puzzle is solvable

Certain moves are impossible

Combinatorics & Permutations

Used to count:

Possible states

Permutation parity

Number of legal arrangements

Combinatorial Game Theory

For impartial two-player games like Nim.

Tools:

Grundy numbers (nimbers)

Sprague–Grundy theorem

XOR operations

Linear Algebra Over Finite Fields

Puzzles like Lights Out can be modeled as systems of linear equations over $GF(2)$.

Solve via Gaussian elimination mod 2.

Computational Complexity

Determine whether a puzzle is:

Easy (P)

Hard (NP-complete)

Very hard (PSPACE-complete)

This helps choose which solution methods are feasible.

Designing Mathematical Models for Puzzle Games

Three examples illustrate how to construct models for solving puzzles.

Example A — 15-Puzzle (Sliding Tiles)

Components

State: permutation of 16 tiles (including blank).

Moves: slide adjacent tile into blank.

Goal: tiles ordered 1–15, blank at end.

Mathematical Background

Solvability depends on parity of permutation (inversion count) + blank row parity.

Exactly half of all permutations are solvable.

Model

Represent puzzle states as nodes in a graph.

Use A with heuristic = sum of Manhattan distances (admissible).

Check inversion count before solving.

Example B — Nim (Multiple Piles of Stones)

Components

State: sizes of piles ($[n_1, n_2, \dots]$)

Moves: remove stones from one pile

Winner: player making the final move

Mathematical Background

Nim is an impartial combinatorial game.

Key rule:

If

...

$n_1 \text{ XOR } n_2 \text{ XOR } n_3 \dots = 0 \rightarrow \text{losing position}$

else $\rightarrow$ winning position

...

Model

Compute the nim-sum (XOR of pile sizes).

If non-zero $\rightarrow$ reduce one pile to make XOR = 0 (winning move).

If zero $\rightarrow$ opponent has a forced win (with perfect play).

Example C — Lights Out (Press Button to Toggle Lights)

Components

State: ON/OFF grid (represented by 0/1).

Move: pressing a cell toggles itself + neighbors.

Mathematical Background

The game can be expressed as:

$$\begin{bmatrix} A x = b \pmod{2} \end{bmatrix}$$

Where:

A = effect matrix

x = vector of presses

b = initial state

Model

Construct matrix (A) based on toggle rules.

Solve using Gaussian elimination modulo 2.

Consider multiple solutions via nullspace.

General Framework for Designing Mathematical Game Models

To analyze or design a new puzzle game:

1. Formally define states and moves.
2. Draw or conceptualize state-space as a graph.
3. Identify invariants or symmetry properties.
4. Estimate state-space size to choose between exact or heuristic methods.
5. Choose appropriate mathematical tools:
 - Permutations/parity
 - XOR/Grundy for impartial games
 - Linear algebra for toggle-based games
 - Graph algorithms for shortest-path problems
6. Design the solving algorithm (A , DP, greedy, algebraic).
7. Prove correctness and analyze complexity.

Exercises Aligned With Learning Objectives

For Objective 1 (Analysis)

Identify state, actions, constraints for:

- 8-puzzle

- Tic-Tac-Toe
- Maze with teleporters

For Objective 2 (Mathematical Principles)

Compute inversion count for a given 15-puzzle state → determine solvability.

For Objective 3 (Model Design)

Construct the GF(2) system ($Ax = b$) for a 3×3 Lights Out instance.

Integrated Task

Design an algorithm for a variation of Nim with a maximum-k removal limit using Grundy numbers.

Summary

- Break a puzzle into state / moves / goal / constraints.
- Look for invariants, often the key to solvability.
- Use A for shortest-path puzzles.
- Use XOR / Grundy for impartial two-player games.
- Use linear algebra over GF(2) for toggle-type puzzles.
- Consider computational complexity before selecting solution methods.

References

Signature and destiny. Available online at <https://www.cuvip.gened.chula.ac.th/Timetable-Detail.php?id=918>