Master's Thesis

Research on the Computational Complexity of the Crush Ice Game

Yanqiu TAN

Supervisor Ryuhei UEHARA

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
(Information Science)

March 2025

Abstract

In this thesis, we investigate the computational complexity of the Crush Ice Game, a physics-based puzzle game played on a hexagonal grid. We present a formal analysis of the game's mechanics, focusing particularly on the interplay between block stability and gravitational effects within its hexagonal grid system. The research transforms the original multiplayer game into a deterministic single-player decision problem while preserving its core mechanical and strategic elements.

Our contribution is twofold. First, we develop a deterministic algorithm for simulating block stability and falling mechanics in a hexagonal grid system, providing a systematic method for evaluating game states. This result implies that the block stability problem of the crush ice game can be solved in polynomial time. Second, we prove that the crush ice puzzle decision problem is NP-complete through a polynomial-time reduction from Planar Monotone 3-SAT. This reduction employs a series of carefully designed gadgets—including variable, clause, and anti-backflow mechanisms—that encode Boolean logic within the game's physical constraints.

The research demonstrates that despite its simple ruleset, the crush ice puzzle exhibits significant computational complexity. Our findings contribute to the broader understanding of complexity theory in physics-based puzzle games and provide insights into the algorithmic challenges of simulating gravity-based mechanics in non-traditional grid systems. Additionally, we identify promising directions for future research.

Keywords: computational complexity, NP-completeness, hexagonal grid systems, physics-based puzzles

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Background

Crush Ice Game is a tabletop game suitable for 2 to 4 players. Due to its varying names and themes, it is difficult to ascertain the original inventor and the time of invention. Figure 1.1 shows the version released by YOU & I TOYS CO., LTD. in Japan in 2016[1]. The name of the game in the paper originates from this version.

Figure 1.1: Crush Ice Game released by YOU & I TOYS CO., LTD. in 2016 (photo by author)

The crush ice game combines strategic depth and interactive fun through its unique mechanics. The player's preparation involves filling the game board with hexagonal ice blocks (usually in 2 colors). There are no requirements for the order in which they are filled or the color arrangement. Then place a penguin on the central ice block. Subsequently, players take turns spinning the roulette. And based on the result of the roulette, knock down the corresponding number and color of ice blocks. Once the penguin drops, that player fails. Therefore, players must think carefully during each round. Players must ensure the stability of the penguins while also trying to affect their opponents as

much as possible. This shows the strategic depth and interactive fun of the game.

The rules of puzzles and board games are complex and diverse. Different rules have their own computational complexities. Research related to this began very early, and the computational complexity of more and more games has been revealed. Based on this, more efficient solving algorithms can be developed.

However, research on gravity-related games is more focused in the field of video games. In the field of board games, research on games that possess gravity elements is still very limited. The crush ice game may seem easy to play, but it possesses elements such as gravity and a hexagonal board. These elements are not common in traditional board games like shogi. From the perspective of computational complexity, it shows potential research value.

1.2 Research Purposes

In this research, we focus on proving the computational complexity of the game under specific rule conditions. As well as how to embody the physical characteristics of the game through algorithms.

The original game rules, designed for multiplayer entertainment with random elements introduced through roulette spins, present significant challenges for formal analysis[6]. To facilitate our theoretical investigation, we have systematically simplified these rules. At the same time, preserving the essential mechanical and strategic elements that make the game interesting from a computational perspective. Specifically, we have transformed the multiplayer gameplay into a deterministic single-player decision problem. This transformation allows us to apply formal methods of complexity analysis while maintaining the core challenge of managing block stability under gravity. In the following text, this single-player version will be referred to as crush ice puzzle.

Through this research, we aim to contribute not only to the understanding of this specific game. As well as to the broader field of computational complexity analysis for physics-based puzzle games. The unique combination of hexagonal grid geometry and gravity-based mechanics in the crush ice game provides an opportunity to explore complexity characteristics that differ from those found in traditional board games and typical video game puzzles.

1.3 Organization

In Chapter 2, we introduced the hexagonal grid system and the modified game rules.

We transformed the game into a decision problem and defined the game stage and objectives.

In Chapter 3, we introduced the physical mechanism for determining whether blocks will fall and developed a deterministic algorithm to simulate the reaction of blocks to gravity in a hexagonal grid system.

In Chapter 4, we proved the NP-completeness of the crush ice puzzle problem by reducing Planar Monotone 3-SAT to the game. We introduced tools such as variable gadgets, clause gadgets, and anti-backflow gadgets.

In Chapter 5, we summarized the research results, pointed out the computational complexity of the crush ice puzzle problem, and looked forward to future research directions, such as finding algorithms for minimum stable configurations.

Chapter 2
Preliminaries

2.1 Hexagonal grid system

The hexagonal grid system presents unique challenges for computational representation and manipulation compared to traditional square grids. While square grids naturally align with two primary axes, hexagonal grids require three primary axes for comprehensive spatial representation. This section introduces the coordinate systems used in our research and their mathematical properties.

2.1.1 Cube Coordinate System

The cube coordinate system provides a solution for representing hexagonal grids. It is derived by intersecting a three-dimensional cubic lattice with the diagonal plane defined by $x + y + z = 0$ [2]. This approach yields several advantageous properties:

1. **Vector Operations:** Standard vector operations from orthogonal coordinate systems can be directly applied to cube coordinates. Including coordinate addition, subtraction and scalar multiplication, division.
2. **Algorithmic Compatibility:** Existing algorithms for distance calculation, rotation, reflection, and line drawing from orthogonal coordinate systems can be adapted to the cube coordinate system with minimal modification.
3. **Unique Representation:** The constraint condition $q + r + s = 0$ ensures that each hexagonal cell has a unique coordinate representation.

2.1.2 Axial Coordinate System

The axial coordinate system is a simplified variant of the cube coordinate system that reduces memory requirements while maintaining the essential properties[2]. In this system:

1. Only two coordinates (q, r) are explicitly stored.
2. The third coordinate s can be derived when needed using the relation $s = -q - r$.
3. All geometric operations remain valid with appropriate transformations.

This representation offers a more efficient storage solution while preserving the mathematical properties necessary for game mechanics implementation.

2.2 Modification of game rules

The initial game rules were designed for a scenario involving multiple players. Players take turns removing ice blocks according to the result of the roulette, while maintaining the stability of the penguin. To facilitate computational complexity analysis, we transform these rules into a deterministic single-player decision problem. While retaining the basic physical mechanisms and strategic elements that make the game challenging.

In modified version, we have eliminated the turn-based structure and random elements. We present the game as a puzzle, where all decisions must be made at the beginning. The game board retains a hexagonal grid structure, which creates unique geometric constraints for the stability of the ice. Players can still knock down ice blocks. But instead of following the instructions of a roulette, they must strategically choose which ice blocks to remove in order to achieve specific goals, while preventing the penguin from falling.

The frame blocks are immutable boundary conditions that provide structural support and define the play area. These blocks cannot be knocked down. The penguin blocks represent failure conditions—their stability must be maintained at all times in any solution. This adds an important constraint to the problem.

2.3 Problem definition

Our modified formal definition of the game revolves around a decision problem with clearly defined inputs and success conditions. Game stage N represents the initial configuration, including a complete description of the position and type of all blocks. This configuration represented by:

- Ice block set $I = \{(q, r) \mid$ an ice block exists at coordinate $(q, r)\}$,
- Immutable frame blocks $F \subseteq I$,
- Penguin block position $p \in I$,
- Target number $k_1, k_2 \in \mathbb{Z}^+$ $(k_1 < k_2)$.

Then the problem can be formally described as a decision problem as follows:

Input: A game stage $N\ (I, F, p, k_1, k_2)$.

Output: Determine whether there exists a removal sequence $S \subseteq I \setminus (F \cup \{p\})$ to achieve:

1. $|S| \leq k_1$ (number of directly removed blocks does not exceed k_1).
2. At least k_2 ice blocks fall in total.
3. The penguin block remains stable.

This formulation transforms the original game's entertainment-focused rules into a

well-defined computational problem suitable for complexity analysis. In the following text, this definition will also be referred to as crush ice puzzle problem.

Chapter 3

Decision on block falling

The physical mechanism of falling blocks is an important component of the game's complexity. We have developed a deterministic algorithm to simulate the reaction of blocks to gravity in a hexagonal grid system. This algorithm captures the fundamental physical principles of the game.

Every block can maintain contact with six adjacent blocks in the hexagonal grid system. These adjacent blocks provide structural support against gravitational force. The spatial arrangement and quantity of these supporting blocks determine the stability state of the central block.

By exhausting all possible adjacent scenarios, we can discover the underlying patterns. When the number of adjacent blocks is between 4 and 6, as shown in Figure 3.1, the block will keep stable. When the number of adjacent blocks is 1, the block will definitely fall.

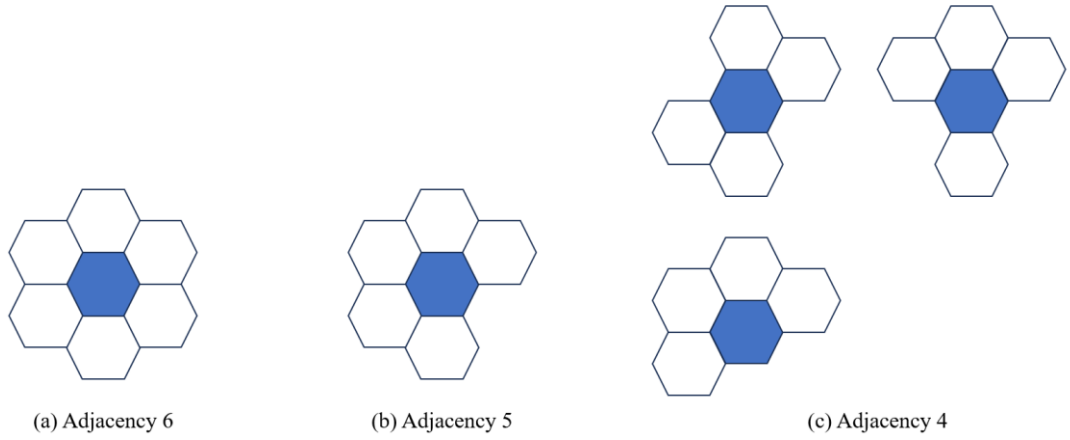
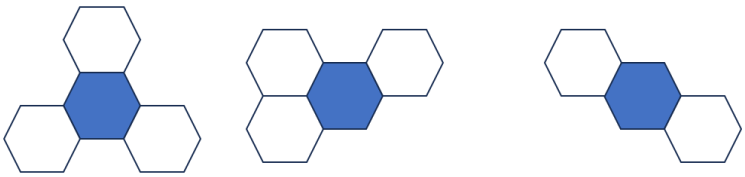


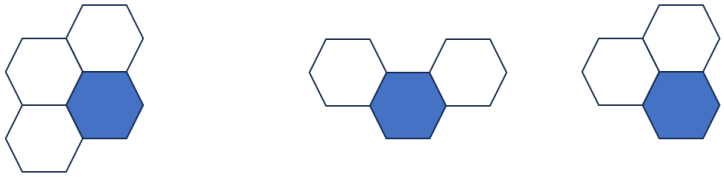
Figure 3.1: Stable structures

When the number of adjacent blocks is 2 or 3, the situation becomes slightly more complex. Depending on the arrangement of the adjacent blocks, the block may fall or may not fall. When the arrangement of adjacent blocks is as shown in Figure 3.2, the block will not fall. However, when the arrangement is as shown in Figure 3.3, the block will fall. Stability depends on the specific arrangement of these neighbors.



(a) Adjacency 3 (b) Adjacency 2

Figure 3.2: Stable situations



(a) Adjacency 3 (b) Adjacency 2

Figure 3.3: Unstable situations

To systematically determine block stability, we have developed a two-phase algorithm that analyzes the local neighborhood of each block. The first phase involves building a comprehensive list of neighbors for any given block. This is implemented through the `build_neighbors function`:

Algorithm 1 build_neighbors function

```
Input: grid // the game board state
       x, y // coordinates of the target block
Output: List of Boolean values indicating presence of neighbors
1  neighbors ← empty list
2  directions ← [(0,1), (1,0), (0,-1), (-1,0), (1,-1), (-1,1)]
3  for each (dx, dy) in directions do
4      if grid[x + dx][y + dy] ≠ EMPTY then
5          append TRUE to neighbors
6      else
7          append FALSE to neighbors
8  return neighbors
```

This function examines the six adjacent positions in the hexagonal grid, recording the presence or absence of blocks in each position. The directions array defines the relative coordinates for each adjacent position, enabling systematic traversal of the neighborhood.

The second phase analyzes the neighborhood configuration to determine stability. This

is implemented in the will_fall function:

Algorithm 2 will_fall function

```
Input: grid // the game board state
       x, y // coordinates of the target block
Output: TRUE or FALSE
1  neighbors ← BUILD_NEIGHBORS(grid, x, y)
2  count ← 0
3  neighbors_extended ← neighbors + [neighbors[0], neighbors[1]]
4  for each neighbor in neighbors_extended do
5      if neighbor = FALSE then
6          count ← count + 1
7          if count = 3 then
8              return TRUE
9      else
10         count ← 0
11 return FALSE
```

This function implements critical stability analysis features. It extends the neighborhood configuration to account for cyclic adjacency. It quantifies consecutive structural voids. It determines instability through the identification of three consecutive unsupported positions.

Based on the above, when the support density of the blocks is relatively high, that is, when there are 4 to 6 adjacent blocks, a stable structure can be maintained and the blocks will not fall. Even when the support density is lower, but the arrangement meets specific conditions, 2 to 3 adjacent blocks are also sufficient to provide stable support.

Conversely, there are two forms of unstable configurations. The first is the minimal support condition for a single adjacent block. The second is an uneven distribution of support, where the positioning of 2 to 3 adjacent blocks is not conducive to gravitational support. Thus, the stability of a block under reduced support density depends on the spatial arrangement of its neighbors.

The algorithmic implementation incorporates several computational considerations. The neighborhood analysis maintains constant temporal complexity through fixed-size examination. The deterministic nature ensures computational consistency across identical configurations. The modular structure permits adaptation to modified physical constraints.

This computational framework for stability determination provides the foundation for

analyzing complex game configurations. It enables systematic evaluation of strategic possibilities within the crush ice puzzle problem.

Chapter 4

NP-Completeness of Crush Ice Puzzle

To prove the NP-completeness of Crush Ice Game problem, we provide a polynomial-time reduction from Planar Monotone 3-SAT to the game. Our reduction demonstrates that any instance of Planar Monotone 3-SAT can be transformed into an equivalent instance of the game. The existence of a solution is guaranteed if and only if the original Planar Monotone 3-SAT formula is satisfiable[4]. This construction relies on gadgets that encode Boolean logic within the physical mechanics of the game.

4.1 Gadgets

The reduction relies on a series of gadgets: variable, clause, anti-backflow, wire, split and parity. Each type of gadget has a specific role in transforming the logical structure of Planar Monotone 3-SAT into the physical constraints of the game.

Variable gadget: The variable gadget is the foundation of Boolean representation. For each variable x_i in the Planar Monotone 3-SAT formula, we construct a device consisting of two selected blocks, as shown in Figure 4.1. These two blocks are arranged in a vertical configuration. The upper block represents the TRUE assignment of the variable, as shown in Figure 4.2, while the lower block represents the FALSE assignment as shown in Figure 4.3.

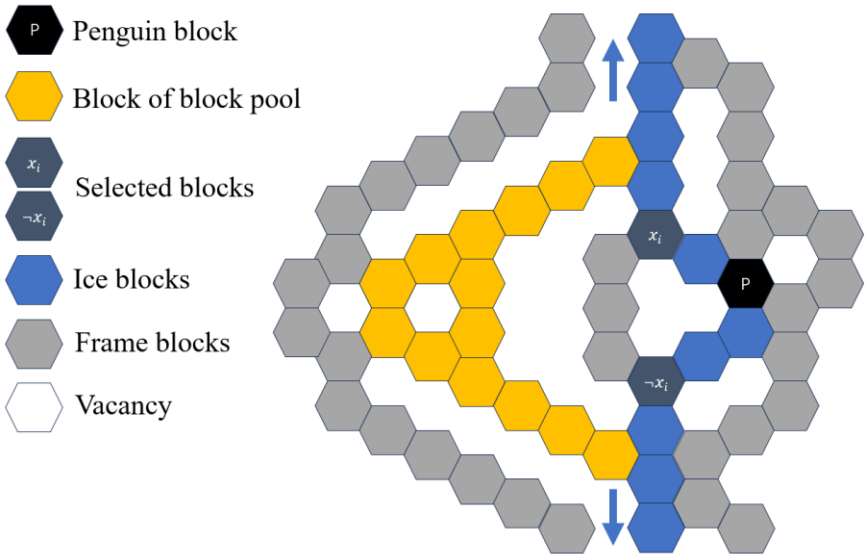


Figure 4.1: Variable gadget

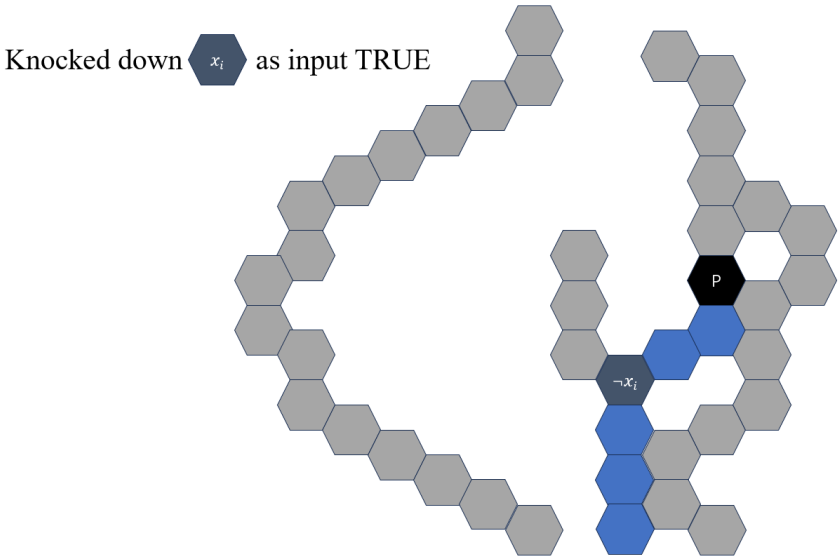


Figure 4.2: TRUE value input

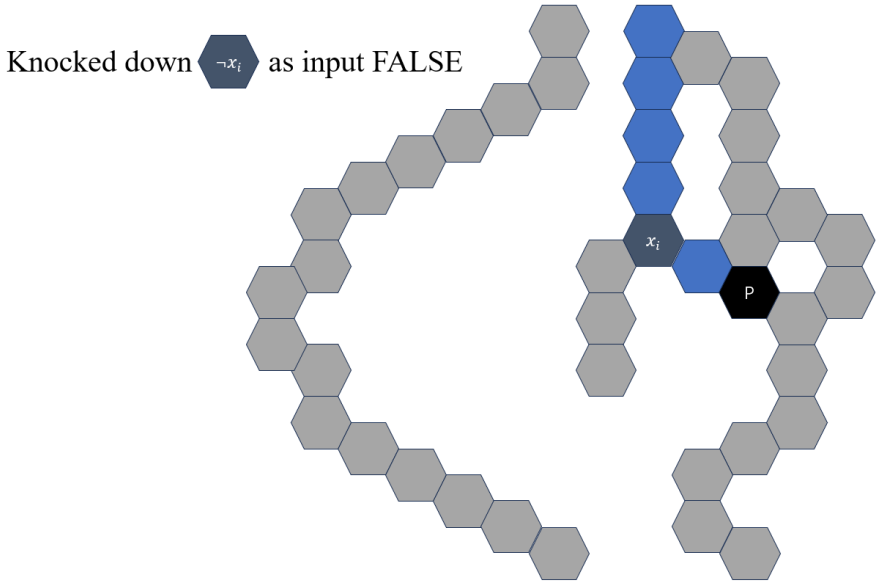


Figure 4.3: FALSE value input

An important feature of this gadget is its block pool. The blocks that belong to the block pool are represented in yellow in Figure 4.1. It contains a very large number of blocks. They are arranged in such a way that they will only fall when one of the two selected blocks is knocked down. The large number of building blocks serves two purposes: First, it forces players to make a choice for each variable. If not, it is impossible to reach the target k. Second, it ensures that choosing both TRUE and FALSE for the same variable will result in the penguin falling, thereby maintaining logical consistency.

The block pool itself adopts a scalable design pattern, allowing it to be extended according to specific rules. As shown in Figure 4.4, this is an example of an extension. This scalability is crucial for ensuring that we can create pools of any size to meet the constraints requirements. The structure of block pool carefully manages the propagation of falling blocks, ensuring that they contribute to the target k while also preventing any unintended effects on other parts of the construction.

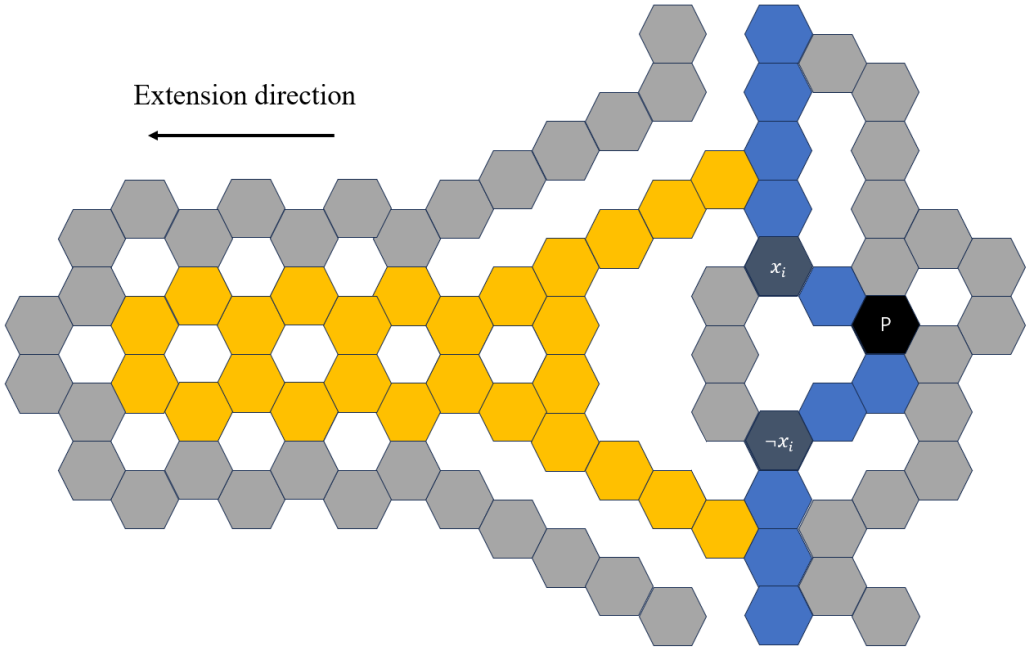


Figure 4.4: An example of a block pool extension

Clause gadget: The clause gadget implements the logical OR operation within a 3-SAT clause. Each clause gadget connects to three variable gadgets, corresponding to the literals in the clause, as shown in Figure 4.5.

The clause gadget is designed such that if any of its input literals is satisfied, it triggers the falling of another large pool of blocks. Here, the block pool has the same function and properties as the block pool in the variable gadget. The internal structure of this clause gadget ensures that the satisfaction of any literal is sufficient to release its pool of blocks, thereby accurately simulating the OR operation. The truth table of the clause gadget shows that it corresponds exactly to the logical OR operation, as shown in Table 4.1. Except when all literals are false, any input combination will trigger the blocks in the brick pool to fall.

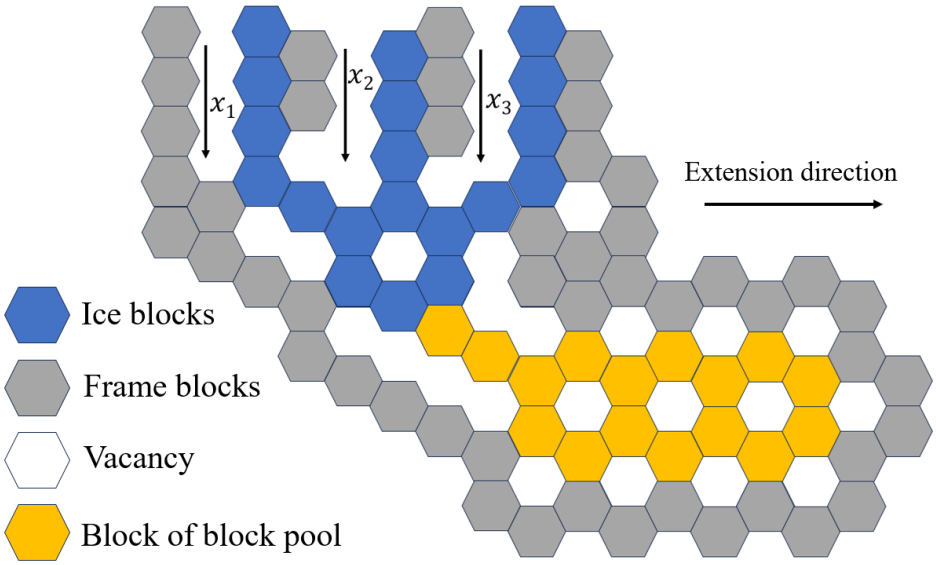


Figure 4.5: Clause gadget

x_1	x_2	x_3	**Formula Satisfied**
1	1	1	1
1	1	0	1
1	0	1	1
0	1	1	1
0	0	1	1
0	1	0	1
1	0	0	1
0	0	0	0

Table 4.1: Truth Table of Clause Gadget

Anti-backflow gadget: The anti-backflow gadget serves as a critical supporting structure that maintains the integrity of our logical structure, as shown in Figure 4.6. These gadgets are strategically placed in various parts of the structure to ensure that the blocks fall and propagate only in the predetermined direction. Without anti-backflow protection, the falling of blocks in one part of the structure could trigger unexpected effects in other parts, thereby compromising the logical consistency of our reduction.

The structure of the anti-backflow gadget consists of channel A and channel B. When

the blocks in channel A fall in sequence, a chain reaction will cause the blocks in channel B to fall as well. However, when the blocks in channel B fall in reverse, this fall will stop at the connection part between the two channels. The block where the fall stops are marked in Figure 4.6. The anti-backflow gadget, through this block structure, allows signals to propagate in only one direction, effectively realizing a physical diode.

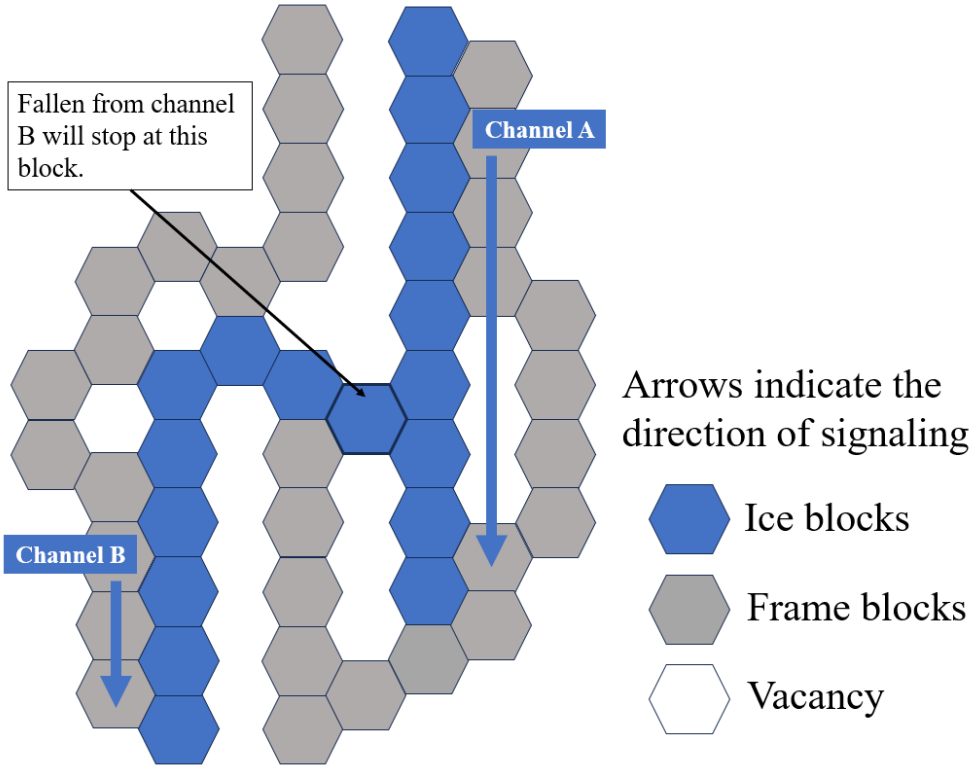


Figure 4.6: Anti-backflow gadget

Other gadgets: In addition to the three main gadgets, there are also some other small gadget structures to implement necessary functions.

As shown in Figure 4.7, the wire gadget demonstrates how the blocks that transmit signals are arranged in both longitudinal and lateral directions, as well as how to switch between the two directions (i.e., make a turn).

The split gadget, as shown in Figure 4.8, can enhance the output degree of variable gadgets[5].

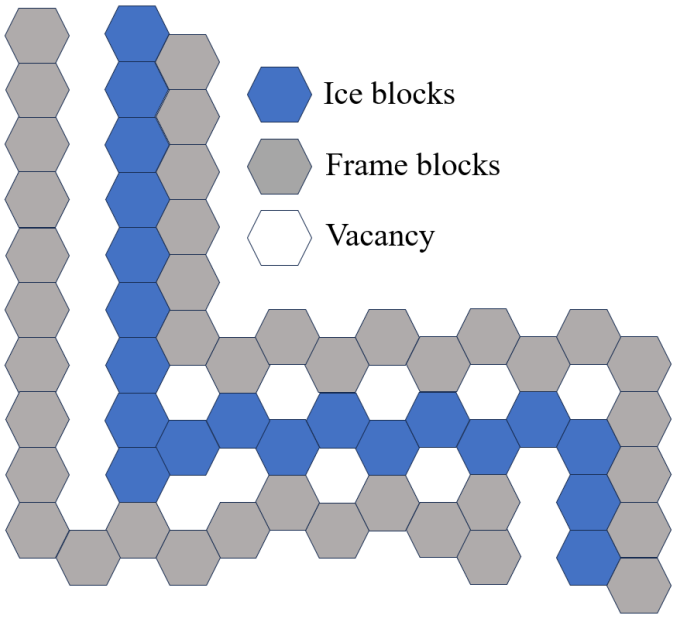


Figure 4.7: Wire gadget

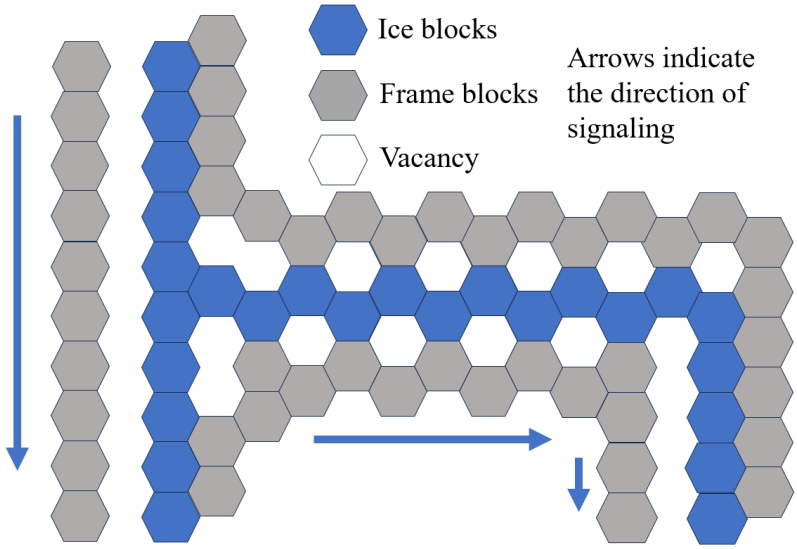


Figure 4.8: Split gadget

Finally, to accommodate the input of other gadgets, we may need to change the position of the signal in the wire[5]. The Parity gadget, as shown in Figure 4.9, can easily achieve this purpose.

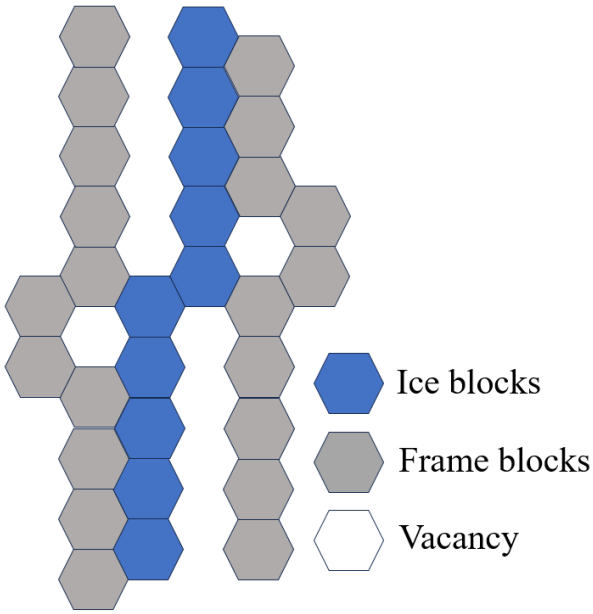


Figure 4.9: Parity gadget

4.2 Reduction from 3-SAT to Crush Ice Puzzle

With the previously mentioned gadgets, we can complete the proof.

Theorem 1. *Crush ice puzzle problem is NP – complete*.

Proof.

To prove NP-completeness, we demonstrate two properties:

1. **NP Membership:** Crush ice puzzle problem∈ NP.
2. **NP-Hardness:** Crush ice puzzle problem is NP-hard via a polynomial-time reduction from Planar Monotone 3-SAT.

First, for NP membership, a decision problem belongs to NP if a proposed solution can be verified in polynomial time. For the crush ice puzzle, a candidate solution is a subset of blocks S to be removed. The verification process involves two steps:

1. **Simulate Block Stability:** Apply the deterministic algorithm from Chapter 3 to check if removing S (where $|S| \leq k_1$) causes at least k_2 ice blocks to fall.
2. **Check Penguin Stability**: Ensure no penguin blocks become unstable during the simulation.

Both steps can be executed in polynomial time: The stability-checking algorithm (Algorithm 1 and 2) iterates over each block's neighbors with constant per-block operations. For a board of size n this requires $O(n)$ time. The penguin stability check is a subset of the simulation and adds no extra complexity.

Thus, Crush Ice Puzzle ∈ NP.

The proof of NP-Hardness is done by a reduction from Planar Monotone 3-SAT. Planar Monotone 3-SAT is a special variant of the classical 3-SAT problem. It has been proven to be NP-complete[3]. Its characteristic is that while maintaining NP-completeness, it adds additional structural constraints[3]. In this variant, the Boolean formula φ must satisfy three key properties:

1. **Monotonicity:** Each clause contains either only positive literals or only negative literals. For example, $(x_1 \vee x_2 \vee x_3)$ and $(\neg x_1 \vee \neg x_2 \vee \neg x_4)$ are valid clauses, while $(x_1 \vee \neg x_2 \vee x_3)$ is not permitted.
2. **Planarity:** The formula's associated graph $G(\varphi)$ must be planar. This graph is constructed as follows:
   Each variable x_i is represented by a rectangle, arranged side by side in the middle; each clause is represented by a rectangle; edges connect variables to the clauses in which they appear; the graph must be drawable on a plane without any edges crossing.
3. **Bipartite Structure:** All positive clauses appear on one side of the variable vertices, while all negative clauses appear on the other side, maintaining planarity.

Figure 4.10 shows an example of a drawing of an instance of a Planar Monotone 3SAT. It describes a formula $\varphi(x_1, x_2, x_3, x_4, x_5, x_6) = (x_1 \vee x_2 \vee x_6) \wedge (x_2 \vee x_4 \vee x_6) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_4 \vee \neg x_5) \wedge (\neg x_1 \vee \neg x_4 \vee \neg x_6)$

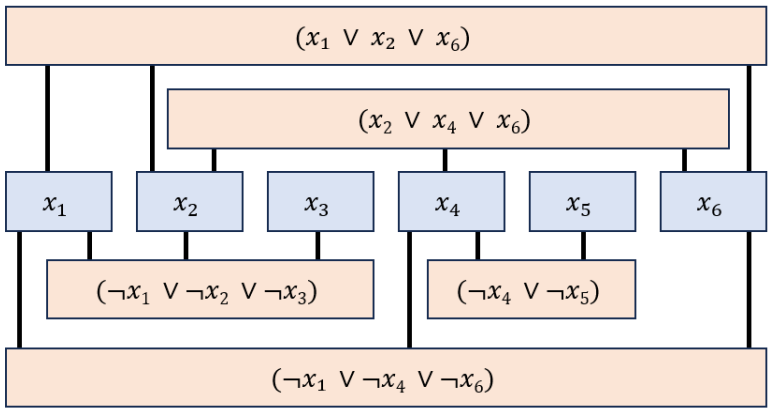


Figure 4.10: An example of a planar representation of an instance of a Planar Monotone 3SAT

The complete reduction transforms a Planar Monotone 3-SAT formula with n variables and m clauses into a crush ice puzzle instance through a systematic construction process. Each variable in the formula is represented by a variable gadget,

and each clause is represented by a clause gadget. The gadgets are interconnected according to the logical structure of the formula, with anti-backflow gadgets protecting the integrity of the logical relationships. Figure 4.11 shows a corresponding crush ice puzzle board.

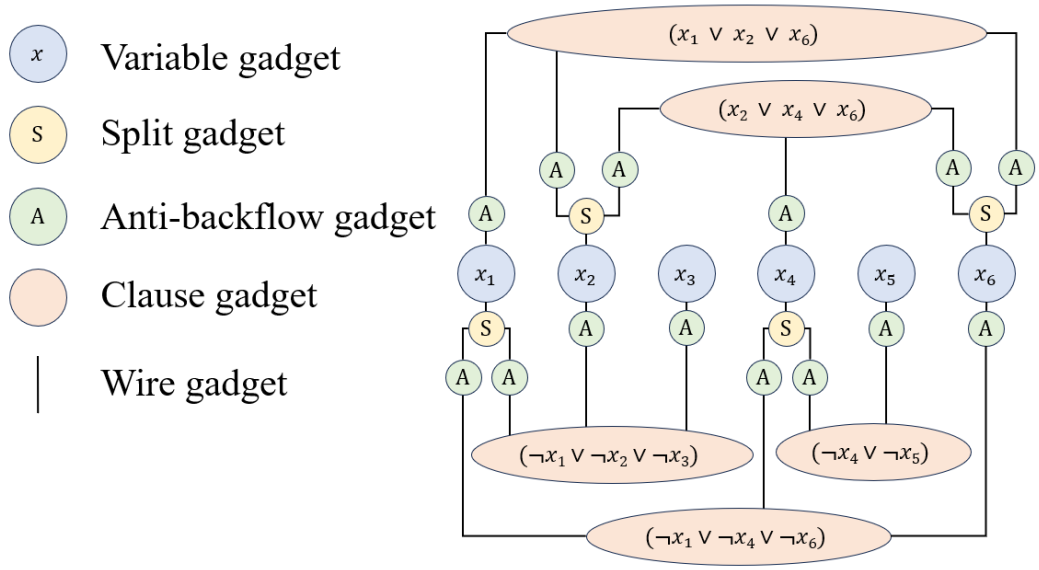


Figure 4.11: A corresponding crush ice puzzle board

Assume that in the previous gadgets, the number of blocks in the block pool is b. Set $k_1 = n$ (where n is the number of variables) and $k_2 = b \times (n + m)$ (where m is the number of clauses). These values enforce the necessary constraints. The k_1 constraint ensures that the solution must accurately knock down at most one choice block from each variable gadget, while k_2 ensures that all variable and clause block pools must fall. This construction ensures that the crush ice puzzle problem has a solution if and only if the φ is satisfiable.

The correctness of our reduction can be proven from two directions. Firstly, if the Planar Monotone 3-SAT formula has a satisfying assignment, we can construct a solution to crush ice puzzle problem by knocking down n choice blocks corresponding to the truth assignment (satisfying k_1). The satisfaction of each clause ensures that all clause gadgets will trigger, causing their block pools to fall along with the variable gadget block pools, reaching the target $k_2 = b \times (n + m)$ fallen blocks.

Conversely, if there exists a solution to crush ice puzzle problem, the k_1 constraint means we can only directly remove n blocks. To achieve k_2 fallen blocks without any penguins dropping, these n removals must correspond exactly to a valid variable

assignment (selecting exactly one ice block from each variable gadget) that satisfies all clauses (triggering all clause gadgets). These choices directly correspond to a satisfying assignment of the Planar Monotone 3-SAT formula.

By demonstrating NP membership and NP-hardness, we conclude that the Crush Ice Puzzle decision problem is NP-complete. ■

This result highlights the intrinsic computational complexity of the game, despite its simple rule set.

In addition, by letting k_1 and k_2 be appropriately set, we have the following corollary:

Corollary 4. 1 *Crush ice puzzle problem is NP− complete if the number of penguins is zero.*

Proof.

To establish the NP-completeness of the crush ice puzzle when the number of penguins is zero, we must demonstrate two properties: NP membership and NP-hardness.

For NP membership, the problem of determining whether a given configuration of the crush ice puzzle problem with zero penguins can result in at most k_1 directly removed blocks causing at least k_2 ice blocks to fall is in NP. This is because, given a candidate solution, we can verify in polynomial time whether removing at most k_1 blocks lead to at least k_2 blocks falling. The verification process involves simulating the block stability and falling mechanics using the stability-checking algorithm (Algorithm 1 and 2). Since the simulation can be done in polynomial time, the problem is in NP.

To prove NP-hardness, we can reduce the crush ice puzzle problem, which we have already shown to be NP-complete, to the crush ice puzzle problem with zero penguins. The key idea is that the presence of penguins in the original problem adds an additional constraint (ensuring that no penguin falls), but if we remove this constraint by setting the number of penguins to zero, the k_1 constraint still maintains the problem's computational complexity.

Given an instance of the crush ice puzzle problem with penguins, we can construct an equivalent instance of the crush ice puzzle problem with zero penguins by simply removing all penguin blocks from the game board. Since the penguin blocks are no longer present, the constraint of ensuring that no penguin falls is automatically satisfied. The problem then reduces to determining whether there exists a sequence of at most k_1 block removals that causes at least k_2 ice blocks to fall, which maintains the challenge of the original problem through these dual constraints.

Since the crush ice puzzle problem is NP-complete, and we have shown that it can be reduced to the crush ice puzzle problem with zero penguins while preserving the essential

complexity through the k_1 and k_2 constraints. The crush ice puzzle with zero penguins must also be NP-hard.

By demonstrating both NP membership and NP-hardness, we conclude that the crush ice puzzle problem with zero penguins is also NP-complete. ■

Chapter 5
Conclusion and Future Works

5.1 Conclusion

This research contributes to our understanding of the computational complexity of physics-based puzzle games through our analysis of the crush ice puzzle. By reducing the Planar Monotone 3-SAT, we have proven the NP-completeness of the game under specific constraints. Additionally, we have developed algorithmic methods for modeling and analyzing the core mechanisms of the game.

Our research began with the fundamental challenge of simulating the physical process of blocks falling in a hexagonal grid system. The developed algorithm provides a deterministic framework for analyzing the stability of the blocks. This framework not only captures the basic physical dynamics of the game, but also maintains computational feasibility. This modeling framework is extendable to other physics-based puzzle games featuring hexagonal grids and gravity.

By demonstrating the NP-completeness through the reduction to Planar Monotone 3-SAT, the profound computational complexity underlying the seemingly simple rule set of the game is revealed. Our construction of the gadget demonstrates how logical constraints can be encoded within the physical mechanics of a game. It provides insights that may be applicable to the complexity analysis of other games with physical element.

5.2 Future Works

In addition, our research has identified several noteworthy directions for future research. There are three main directions.

First, development of algorithms for finding minimum stable configurations. The minimum stable configuration refers to the minimum number of ice blocks and their arrangement required on the board at game stage N to ensure that the penguin does not fall. A game stage N, which consists a complete board of any shape. One edge of the board is labeled as frame blocks, the rest of the board contains a penguin block placed in the center of the board, and the rest of the board is all ice blocks. It is worth noting that at this time, there are no empty spaces on the board.

By observation, a chain of bricks containing penguin bricks tends to be minimal, as shown in Figure 5.1. Therefore, the idea of the algorithm is to traverse the bricks along

the 3 pairs of penguin bricks in the diagonal direction. According to depth-first exploration until the block is frame block. Count the number of ice blocks in the blocks that have been traversed, and choose the smallest one, output as minimum.

However, we have also identified certain specific chessboard configurations for which the optimal solution does not fall within these depth-first traversal patterns. This specific board configuration and its solution are shown in Figure 5.2.

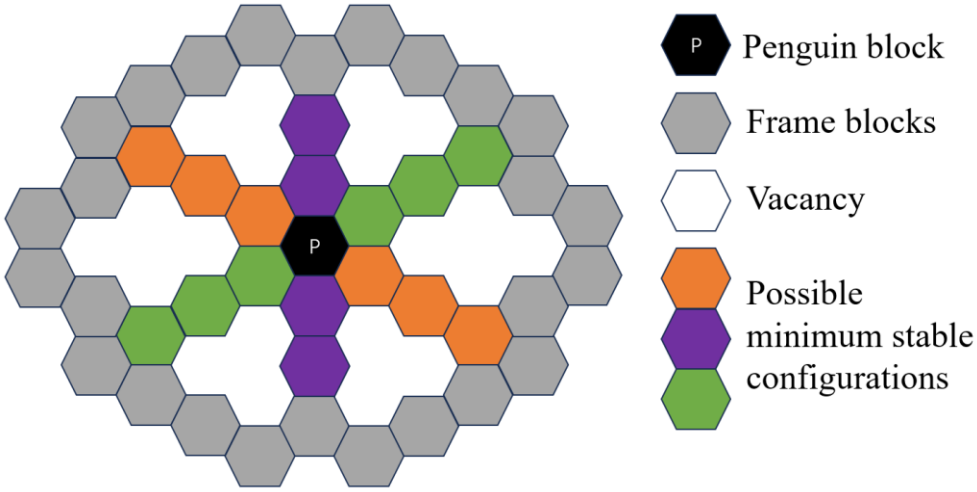


Figure 5.1: Observations on the minimum stable configuration

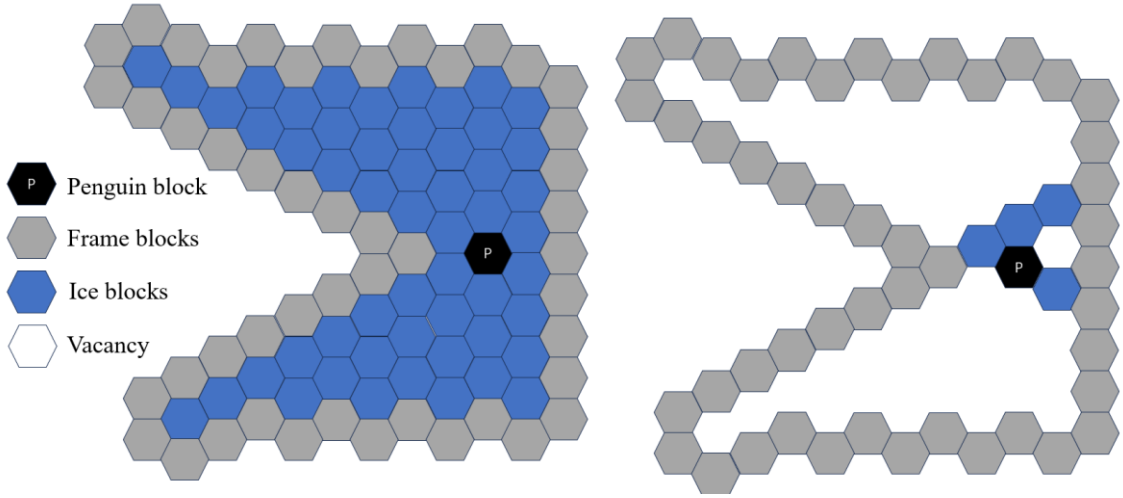


Figure 5.2: Specific board configuration and its solution

Due to the emergence of special cases, the design approach of the algorithm perhaps should shift to a breadth-first exploration. It reveals that the relevant algorithms deserve more in-depth research.

Second, the computational complexity of the crush ice game under its multiplayer rules is also worthy of investigation. As a conjecture, if it can be reduced from Quantified Boolean Formula (QBF), then it would be possible to prove that the computational complexity of its multiplayer variant is PSPACE-complete.

Third, there is another game worth noting called Beehive Game. Its image is shown in Figure 5.3[7]. The rules of Beehive Game are the same as those of Crush Ice Game, with the key difference being that the game board is oriented vertically rather than horizontally. This distinction significantly alters the impact of gravity on the gameplay process. As a result, the computational complexity of the two games may differ. This difference is also worthy of further research.

Figure 5.3: Beehive Game (photo by author)

Bibliography

[1] You & I Toys Co., Ltd. (n.d.). You & I Toys Co., Ltd. [Website]. Retrieved January 22, 2025, from http://www.you-and-i-toys.co.jp/

[2] Red Blob Games. (n.d.). Hexagonal Grids. Retrieved January 22, 2025, from https://www.redblobgames.com/grids/hexagons/#coordinates

[3] de Berg, M., & Khosravi, A. (2010). Optimal binary space partitions in the plane. In Computing and Combinatorics: 16th Annual International Conference, COCOON 2010, Nha Trang, Vietnam, July 19-21, 2010. Proceedings 16 (pp. 216-225). Springer Berlin Heidelberg.

[4] Fekete, S. P., Mitchell, J. S., Rieck, C., Scheffer, C., & Schmidt, C. (2024). Dispersive Vertex Guarding for Simple and Non-Simple Polygons. arXiv preprint arXiv:2406.05861.

[5] Demaine, E. D., Okamoto, Y., Uehara, R., & Uno, Y. (2014). Computational complexity and an integer programming model of Shakashaka. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 97(6), 1213-1219.

[6] Furtak, T., Kiyomi, M., Uno, T., & Buro, M. (2005, July). Generalized Amazons is PSPACE-complete. In IJCAI (pp. 132-137).

[7] Flying Tiger. (n.d.). Women's products. Shop-list. Retrieved from https://shop-list.com/women/flyingtiger/3012523/