

# A Computational Analysis of P-Position Patterns in the $3 \times n$ Game of Chomp

Dan Nguyen

November 24, 2025

## Abstract

The game of Chomp is an impartial combinatorial game played on a grid, where players take turns removing squares and all squares below and to the right, with the objective of forcing the opponent to eat the poisoned top-left square. While the winning strategy for  $2 \times n$  boards is well-known, the  $3 \times n$  case remains unsolved in closed form. This paper bridges the gap between theoretical existence and practical strategy through a computational approach. Using a recursive solver with memoization, I analyzed the state space of  $3 \times n$  Chomp for  $n$  up to 500. Instead of relying on Sprague-Grundy values, this research focuses on identifying structural patterns within the set of P-positions (losing positions). I report the discovery of infinite linear families of P-positions, alongside linear arithmetic progressions for fixed third-row lengths.

## 1 Introduction

The game of Chomp, despite its simple rules and whimsical premise, stands as a significant object of study within combinatorial game theory. Its rules can be taught in seconds, yet its analysis has challenged mathematicians for decades. While widely recognized in its "chocolate bar" form, the game is mathematically equivalent to problems in number theory involving divisors. The central question for Chomp is determining which player has a winning strategy.

For any rectangular board larger than  $1 \times 1$ , a non-constructive proof technique known as a *strategy-stealing argument* establishes that the first player is guaranteed to win. However, this proof gives no indication as to *what* the winning move is. While the  $2 \times n$  case is completely solved, the transition to three rows marks a dramatic leap in complexity. For  $3 \times n$  Chomp, no simple, closed-form winning strategy is known.

### 1.1 The Chocolate Bar Problem

In its most common presentation, Chomp is a two-player abstract strategy game played on a rectangular grid of cells, often visualized as a chocolate bar [1]. The game is defined by the following rules:

- **The Board:** The game begins with a rectangular  $m \times n$  grid of squares. By convention, the top-left square at coordinate  $(1, 1)$  is designated as "poisoned".
- **The Move:** Players take turns selecting any available square  $(i, j)$  on the board. This move, known as a "chomp," "eats" the selected square along with every square in the rectangle formed by  $(i, j)$  as its top-left corner and the bottom-right corner of the board. Formally, selecting  $(i, j)$  removes all squares  $(x, y)$  such that  $x \geq i$  and  $y \geq j$ .
- **The Losing Condition:** The player who is forced to select and eat the poisoned square at  $(1, 1)$  loses the game.

A single move can transform the board from a simple rectangle into a non-rectangular shape. These resulting shapes are known as Young's diagrams (or Ferrers diagrams), which are fundamental objects in combinatorics. Play continues, with the board progressively shrinking, until only the poison square remains, and the player whose turn it is must take it and lose.

## 1.2 Mathematical Origins and the Divisor Game

While the accessible "chocolate bar" formulation has contributed significantly to Chomp's popularity, its mathematical origins are rooted in number theory. The name "Chomp" and the chocolate bar analogy are widely attributed to the mathematician David Gale, who published the game in this form in 1974 [1]. This intuitive presentation was famously popularized by Martin Gardner in his "Mathematical Games" column in *Scientific American* [3], making the game accessible to a broad audience.

However, an isomorphic game known as the *Divisor Game* was published decades earlier by the Dutch mathematician Frederik Schuh in 1952 [2]. The rules of the Divisor Game are played with a fixed natural number  $N$ . Two players take turns choosing a positive divisor of  $N$ , but they cannot choose a multiple of any previously chosen divisor. The player who cannot make a move loses.

The equivalence between the Divisor Game and Chomp is a powerful insight. A number  $N$  with prime factorization  $N = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$  corresponds to a  $k$ -dimensional Chomp board with dimensions  $(a_1 + 1) \times (a_2 + 1) \times \cdots \times (a_k + 1)$ . A move in the Divisor Game corresponds to selecting a square in the Chomp grid, and the rule against choosing multiples corresponds to the removal of all squares "above and to the right" [2]. This formulation inherently establishes Chomp not merely as a recreational puzzle, but as a specific instance of a game played on a partially ordered set (poset).

## 1.3 The Question of a Winning Strategy

The central question for any combinatorial game is determining which player has a winning strategy and what that strategy is. For Chomp, the existence of a winning strategy for the first player is remarkably easy to prove, yet the nature of that strategy remains one of the most challenging open problems in the field.

### 1.3.1 The Strategy-Stealing Argument

For any rectangular board larger than  $1 \times 1$ , the first player is guaranteed to have a winning strategy. This is established by a non-constructive proof technique known as a *strategy-stealing argument* [1].

The argument proceeds by contradiction. Assume, for the sake of argument, that the *second* player has a winning strategy against any opening move.

1. Player 1 begins by chomping only the single square at the bottom-right corner,  $(m, n)$ .
2. By our assumption, Player 2 must have a winning response to this move. Let this winning response be to chomp at square  $(i, j)$ .
3. However, since Player 1's initial move only removed the single square  $(m, n)$ , the square  $(i, j)$  was available to Player 1 from the very start.
4. Therefore, Player 1 could have simply chomped at  $(i, j)$  as their *first* move. By "stealing" Player 2's winning response, Player 1 would be in the winning position.

This contradicts the assumption that Player 2 has a winning strategy. Thus, Player 1 must have a winning strategy. Crucially, this proof is **non-constructive**: it proves *that* a winning move exists, but gives no indication as to *what* it is.

### 1.3.2 Constructive Strategies for Solved Cases

While the general strategy is unknown, explicit constructive strategies have been found for specific board shapes that exhibit symmetry.

**Square ( $n \times n$ ) Boards:** For a square board of size  $n \times n$  ( $n \geq 2$ ), the winning strategy relies on diagonal symmetry.

- **Opening Move:** Player 1 chomps the square at  $(2, 2)$ , leaving an L-shaped board with equal horizontal and vertical arms.
- **Response:** Player 1 adopts a mirroring strategy. If Player 2 chomps  $k$  squares from the horizontal arm, Player 1 chomps  $k$  squares from the vertical arm (and vice versa). This ensures the board remains symmetric after Player 1's turn, eventually forcing Player 2 to take the poison [1].

**Two-Row ( $2 \times n$ ) Boards:** The case of a  $2 \times n$  board is also completely solved using a different form of symmetry equivalent to the game of Nim.

- **Opening Move:** Player 1 chomps the bottom-right square  $(2, n)$ .
- **Response:** This move leaves the two rows with a difference of one. This state is combinatorially equivalent to a game of Nim with two equal piles. Player 1 mirrors Player 2's moves on the opposite row. The losing positions (P-positions) for the 2-row game are precisely those of the form  $(k, k - 1)$  [2].

### 1.3.3 The $3 \times n$ Frontier

The transition from two rows to three marks a dramatic leap in complexity. The symmetries that simplify the  $n \times n$  and  $2 \times n$  cases break down, and the game tree explodes in size. As noted by Zeilberger, the problem walks a "tightrope between the trivial and the impossible" [5]. For  $3 \times n$  Chomp, no simple, closed-form winning strategy is known, necessitating the computational approach detailed in the following section.

## 1.4 Thesis: A Computational Analysis of Patterns

While the  $3 \times n$  case lacks a simple human-executable strategy, it is not entirely unsolved. The game is known to be "computationally solved" thanks to deep theoretical work. It has been demonstrated that the sequence of winning and losing positions (known as N-positions and P-positions, respectively) for  $3 \times n$  Chomp is **ultimately periodic** [4]. This theorem provides a powerful algorithm for determining the status of any  $3 \times n$  position, even if it does not provide a simple, closed-form strategy.

This paper leverages this theoretical foundation to bridge the gap between abstract existence proofs and practical play. The thesis of this capstone project is as follows:

By implementing a recursive solver with memoization, I analyze the state space of  $3 \times n$  Chomp for  $n$  up to 500. Instead of relying on Sprague-Grundy values, this research focuses on identifying structural patterns within the set of P-positions (losing positions). I report the discovery of infinite linear families of P-positions, alongside linear arithmetic progressions for fixed third-row lengths, providing a partial constructive strategy for the  $3 \times n$  game.

This paper will first detail the necessary combinatorial game theory, then describe the methodology of the solver's implementation, present the results (the list of P-positions), and analyze these findings in the context of known periodicity theorems.

## 2 Methodology: Building a $3 \times n$ Chomp Solver

To verify theoretical predictions and explore patterns in the distribution of winning and losing positions, I implemented a computational solver in Python. This solver uses a recursive backtracking algorithm enhanced with memoization to exhaustively map the game's state space.

### 2.1 Representing the Game State

While Chomp is played on a rectangular grid, the board quickly loses its rectangular shape as players remove pieces. The remaining structure always satisfies a key property: the number of blocks in any row is less than or equal to the number of blocks in the row above it. In combinatorics, such a shape is known as a **Young's Diagram** (or Ferrers diagram).

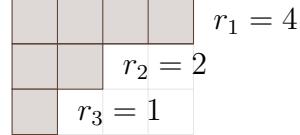
To represent these shapes in code, I do not need to store the status of every single cell in a 2D array. Instead, I represent the board state efficiently as a tuple of integers corresponding to the lengths of the rows.

A state  $S$  for a  $3 \times n$  board is defined as a tuple:

$$S = (r_1, r_2, r_3)$$

where  $r_i$  represents the number of remaining blocks in row  $i$ , satisfying the condition  $n \geq r_1 \geq r_2 \geq r_3 \geq 0$ .

For example, a standard  $3 \times 4$  board begins in the state  $(4, 4, 4)$ . If a player chomps at coordinate  $(1, 2)$  (row 1, column 2), all blocks to the right and below are removed, resulting in the state  $(4, 2, 1)$ , visualized below:



**State Representation:**  $(4, 2, 1)$

Figure 1: A  $3 \times 4$  Chomp board reduced to a Young's Diagram represented by the tuple  $(4, 2, 1)$ .

## 2.2 Algorithm for Generating Successor States

The core of the solver is a recursive function, `is_winning_state(state)`, which determines if a given board configuration is a Winning (N-position) or Losing (P-position) state. The algorithm relies on the standard definition of impartial games: a state is losing if and only if all moves lead to winning states for the opponent, and a state is winning if there exists at least one move leading to a losing state for the opponent.

The algorithm proceeds in four logical steps:

1. **Input:** The function accepts the current state tuple  $(r_1, r_2, r_3)$ .
2. **Base Case:** The empty board is an N-position, as it is the only available move from the most trivial P-position,  $(1, 0, 0)$ .
3. **Recursive Step:** To determine the value of the current state, the algorithm generates all possible successor states. A successor is generated by simulating a "chomp" at every valid coordinate  $(i, j)$  on the current board.
  - The solver iterates through all valid moves.
  - For each move, it computes the resulting tuple  $S'$ .
  - It recursively calls `is_winning_state(S')`.
  - **Logic:**
    - If **any** successor  $S'$  is found to be a P-position (i.e., `is_winning_state(S') == False`), then the current state allows the player to force a win. The function immediately returns **True** (N-position).

- If the loop completes and **all** successors are found to be N-positions (i.e., `is_winning_state( $S'$ ) == True`), then the current player has no escape. The function returns **False** (P-position).
4. **Memoization (Caching):** Because the game of Chomp allows for multiple move sequences to reach the same board state (transposition), the state space is a directed acyclic graph rather than a tree. To prevent redundant calculations and exponential runtime, I utilize a hash map (dictionary) to store the result of every computed state. Before performing the recursive step, the algorithm checks if the current state is already in the cache.

This approach allows us to efficiently determine the status of  $3 \times n$  boards for  $n$  up to 500, a range sufficient to detect long-term periodic patterns.

## 3 Results and Analysis

### 3.1 Analysis of Periodicity in P-Positions

#### 3.1.1 Scarcity and Growth of P-Positions

An initial macroscopic analysis of the generated data reveals important trends regarding the distribution of P-positions. Figure 2 presents two views of the P-position data for boards up to width  $n = 500$ .

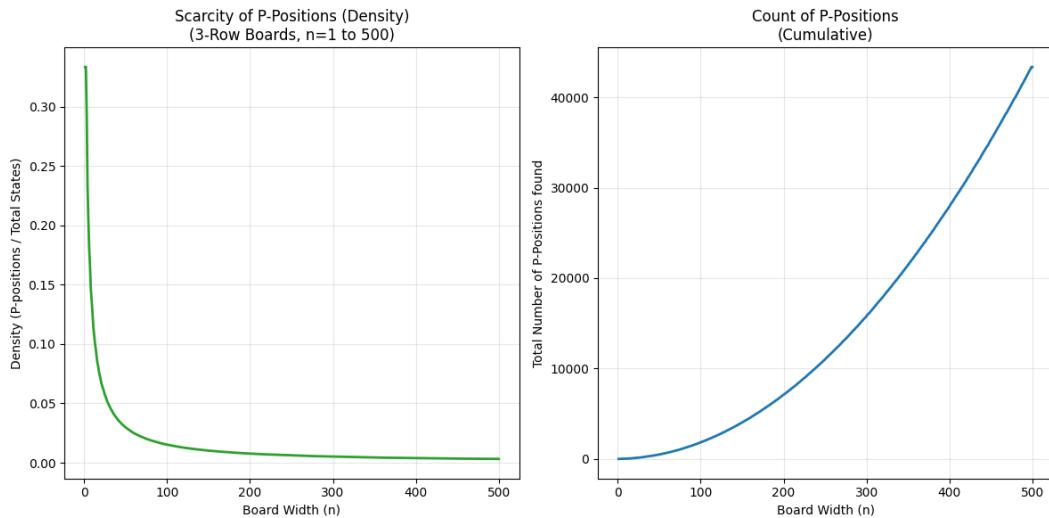


Figure 2: Statistical analysis of P-positions for  $3 \times n$  Chomp,  $n \leq 500$ . Left: The density of P-positions relative to the total state space. Right: The cumulative count of identified P-positions.

The plot on the left illustrates the density of P-positions, defined as the ratio of P-positions to the total number of states for a given  $n$ . The density drops sharply from an

initial high, quickly stabilizing at a very low value as  $n$  increases, indicating that P-positions are exceedingly rare within the vast state space.

Conversely, the plot on the right shows the cumulative count of P-positions found. The curve exhibits a clear upward trend with an increasing slope, suggesting that while sparse, the absolute number of P-positions continues to grow significantly.

These combined observations suggest that P-positions are not distributed randomly. This realization directly prompted the more granular investigation into structural patterns and infinite families detailed in the subsequent sections.

### 3.1.2 Periodicity in the optimal first move

I analyze the optimal opening move for a rectangular board of size  $3 \times n$ , represented by the state  $(n, n, n)$ . By the Strategy Stealing Argument, we know  $(n, n, n)$  is an N-position for all  $n > 0$ . However, the argument is non-constructive.

My computational results indicate that the unique winning move always occurs on either the second or the third row, never both, at least for the boards with size up to  $3 \times 500$ . Specifically, for each  $n$ , the optimal move is either to position  $(n, n, k)$  (a move on the third row) or to position  $(n, k, k)$  (a move on the second row).

We partition the positive integers  $n$  into two sets based on the location of this winning move:

- Let  $A$  be the set of  $n$  for which the winning move is on the third row (index 2). For these  $n$ , the move is to  $(n, n, f(n))$ .

$$A = \{2, 5, 7, 9, 12, 14, 17, 19, 22, 23, 26, 29, 31, 33, 36, 38, 41, 43, 46, 47, 50, \dots\}$$

The corresponding target columns  $f(n)$  are:

$$F = \{1, 3, 4, 6, 8, 10, 12, 13, 15, 16, 18, 20, 21, 23, 25, 27, 29, 30, 32, 33, 35, \dots\}$$

The corresponding sequence of  $n - f(n)$  for all  $n \in A$  is:

$$\{1, 2, 3, 3, 4, 4, 5, 6, 7, 7, 8, 9, 10, 10, 11, 11, 12, 13, 14, 14, 15, 15, 16, 17, 18, 18, 19, \dots\}$$

- Let  $B$  be the set of  $n$  for which the winning move is on the second row (index 1). For these  $n$ , the move is to  $(n, g(n), g(n))$ .

$$B = \{1, 3, 4, 6, 8, 10, 11, 13, 15, 16, 18, 20, 21, 24, 25, 27, 28, 30, 32, 34, 35, \dots\}$$

The corresponding target columns  $g(n)$  are:

$$G = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, \dots\}$$

The corresponding sequence of  $n - g(n)$  for all  $n \in B$  is:

$$\{2, 2, 3, 4, 5, 5, 6, 7, 7, 8, 9, 9, 11, 11, 12, 12, 13, 14, 14, 15, 15, 16, 17, 17, 18, 18, 19, 19, 21, 21, \dots\}$$

**Observations and Conjecture** From this data, I observe structural properties regarding the uniqueness of the optimal move. Based on my exhaustive computational analysis of the state space for  $n \leq 500$ , I formulate the following conjecture:

**Conjecture 1.** *The sets A and B are complementary in the set of positive integers  $\mathbb{Z}^+$ . That is, they form a disjoint partition of  $\mathbb{Z}^+$  such that  $A \cap B = \emptyset$  and  $A \cup B = \mathbb{Z}^+$ . In game-theoretic terms, this implies that the winning move from the rectangular configuration  $(n, n, n)$  is unique.*

My computational verification confirms that for every integer  $n \leq 500$ ,  $n$  belongs to precisely one of these sets. While  $A \cup B = \mathbb{Z}^+$  is a consequence of the Strategy Stealing Argument (which guarantees existence) combined with the fact that row 1 moves are losing, the disjointness ( $A \cap B = \emptyset$ ) implies uniqueness.

I support the uniqueness claim by introducing a conjecture derived from my computational results, visualized in Figure 3.

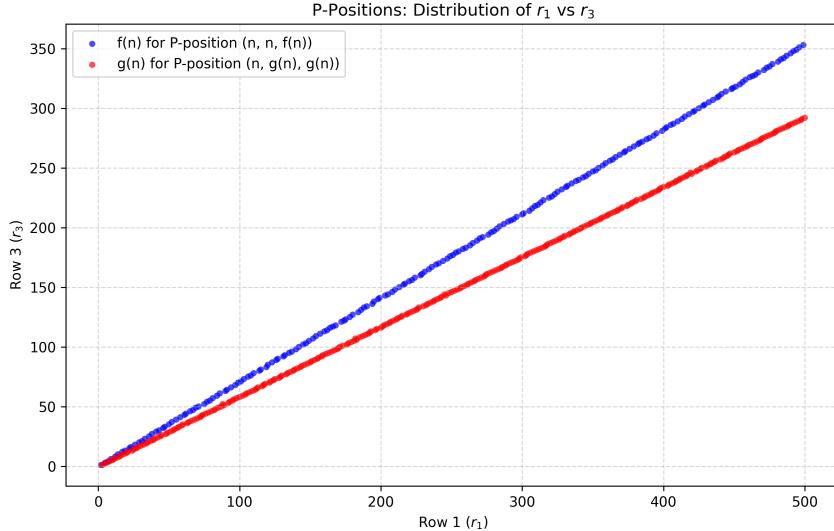


Figure 3: Scatter plot of  $r_3$  values for P-positions in sets A and B. The blue points represent  $x = f(n)$  for positions  $(n, n, x)$ , and the red points represent  $y = g(n)$  for positions  $(n, y, y)$ . The separation shows  $x > y$  for all  $n$ .

**Conjecture 2** (Ordering of Target Columns). *Let  $P_A = (n, n, f(n))$  and  $P_B = (n, g(n), g(n))$  be potential P-positions associated with board size  $n$ . Based on the distinct linear trends observed in my data (Figure 3), for any fixed  $n$ , the third-row value in set A is strictly greater than the second-row value in set B. That is:*

$$f(n) > g(n)$$

or, using the simplified notation  $x = f(n)$  and  $y = g(n)$ :

$$x > y$$

*Argument for Uniqueness.* Assuming Conjecture 2 holds, We prove that for any  $n$ , the optimal move is unique, implying  $A$  and  $B$  are disjoint. Suppose, for contradiction, that  $n \in A \cap B$ . This implies that both  $(n, n, x)$  and  $(n, y, y)$  are P-positions for some  $x, y < n$ , where  $x = f(n)$  and  $y = g(n)$ .

By Conjecture 2, we have  $x > y$ . This inequality allows a specific transition between the two states: From the state  $(n, n, x)$ , a player can choose to chomp at row 2, column  $y$  (where rows are 1-indexed).

1. The move reduces row 2 from  $n$  to  $y$ .
2. Since  $y < x < n$ , the move also forces a reduction in row 3 from  $x$  to  $\min(x, y) = y$ .

The resulting board state is exactly  $(n, y, y)$ . Since we assumed  $(n, n, x)$  is a P-position, any state reachable from it must be an N-position. However, we also assumed  $(n, y, y)$  is a P-position. This implies a P-position can reach another P-position in a single move, which is a contradiction.

Therefore, the assumption that  $n \in A \cap B$  is false. The sets must be disjoint.  $\square$

Furthermore, we propose the following properties regarding the monotonicity of the target function  $F$  and the structure of set  $G$ :

**Conjecture 3.** *The function  $f : A \rightarrow F$  is strictly increasing. That is, for any  $n_1, n_2 \in A$  with  $n_1 < n_2$ , we have  $f(n_1) < f(n_2)$ .*

**Conjecture 4.** *The set of target values  $G$  is equal to the set of natural numbers  $\mathbb{N}$ . That is,  $G = \{0, 1, 2, \dots\}$ .*

My computational verification confirms that Conjecture 3 holds for all integers  $n \leq 500$ , however the similar conjecture doesn't hold for the function  $g : B \rightarrow G$ , as both 88 and 89 are parts of  $B$ , however,  $g(88) = 52$  and  $g(89) = 51$ , that is  $(88, 52, 52)$  and  $(89, 51, 51)$  are both P-positions.

### 3.1.3 Periodicity in P-Positions by number of third-row-blocks

Regarding the periodicity in P-positions based on the number of blocks in the third row, I analyze the game position represented by  $(r_1, r_2, r_3)$  with  $r_1 \geq r_2 \geq r_3$ . My computational results reveal the following P-positions for small fixed values of  $r_3$ :

- If  $r_3 = 0$ , the P-positions are  $(a + 1, a, 0)$  for nonnegative integers  $a$ .
- If  $r_3 = 1$ , the P-positions are  $(3, 1, 1)$  and  $(2, 2, 1)$ .
- If  $r_3 = 2$ , the P-positions are  $(a + 4, a + 2, 2)$  for nonnegative integers  $a$ .
- If  $r_3 = 3$ , the P-positions are  $(6, 3, 3)$ ,  $(7, 4, 3)$ , and  $(5, 5, 3)$ .
- If  $r_3 = 4$ , the P-positions are  $(8, 4, 4)$ ,  $(9, 5, 4)$ ,  $(10, 6, 4)$ , and  $(7, 7, 4)$ .
- If  $r_3 = 5$ , the P-positions are  $(10, 5, 5)$ ,  $(9, 6, 5)$ , and  $(a + 11, a + 7, 5)$  for nonnegative integers  $a$ .

- If  $r_3 = 6$ , the P-positions are  $(9, 9, 6)$ ,  $(11, 6, 6)$ ,  $(12, 7, 6)$ , and  $(13, 8, 6)$ .
- If  $r_3 = 7$ , the P-positions are  $(12, 9, 7)$ ,  $(13, 7, 7)$ ,  $(14, 8, 7)$ , and  $(a + 15, a + 10, 7)$  for nonnegative integers  $a$ .
- If  $r_3 = 8$ , the P-positions are  $(12, 12, 8)$ ,  $(14, 9, 8)$ ,  $(15, 8, 8)$ ,  $(16, 10, 8)$ , and  $(17, 11, 8)$ .
- If  $r_3 = 9$ , the P-positions are  $(14, 11, 9)$ ,  $(16, 9, 9)$ ,  $(17, 10, 9)$ , and  $(a + 18, a + 23, 9)$  for nonnegative integers  $a$ .
- If  $r_3 = 10$ , the P-positions are  $(14, 14, 10)$ ,  $(18, 10, 10)$ ,  $(19, 11, 10)$ ,  $(20, 12, 10)$ , and  $(21, 13, 10)$ .
- If  $r_3 = 11$ , the P-positions are  $(17, 14, 11)$ ,  $(19, 12, 11)$ ,  $(20, 11, 11)$ ,  $(22, 13, 11)$ ,  $(21, 18, 11)$ , and  $(a + 23, a + 15, 11)$  for nonnegative integers  $a$ .

A structural dichotomy becomes apparent when examining these results. For any fixed third row  $r_3$ , the P-positions constitute either a finite set or an eventual infinite linear family (a "robust" family). Specifically, we observe linear families for the set of third-row values:

$$\mathcal{L} = \{0, 2, 5, 7, 9, 11, 14, 17, 19, 22, 24, 26, 28, \dots\}$$

The composition of this set suggests two structural properties governing the game. First, we note that  $\mathcal{L}$  contains no consecutive integers. This observation motivates my first conjecture regarding the spacing of these infinite families.

**Conjecture 5** (Non-Adjacency / The Gap Principle). *If  $r_3 = k$  admits an infinite linear family of P-positions, then  $r_3 = k + 1$  cannot have a linear family.*

Second, to predict exactly when a linear family emerges, we look to the structure of the preceding row value. The data suggests that specific geometric configurations in row  $k$  necessitate a linear response in row  $k + 1$ .

**Conjecture 6** (Existence of Linear Families). *If, for a fixed third row  $k$ , we have:*

1. *P-positions of the form  $(x, x, k)$  and  $(y, k, k)$ , AND*
2. *No P-position of the form  $(z, z, k + 1)$ ,*

*then there exists a linear robust family for  $r_3 = k + 1$ .*

While a universal proof for these conjectures remains an open problem, we can substantiate the pattern by rigorously proving the existence of the families identified in my initial data. In the following subsections, I provide formal proofs for the linear families observed at  $r_3 = 2$  and  $r_3 = 5$ .

## 3.2 Recursive Proof of cases (a+4, a+2, 2) and (a+11, a+7, 5)

### 3.2.1 The P-positions family (a+4, a+2, 2)

I have identified a robust infinite family of P-positions when the third row has length 2.

**Theorem 1.** *For any nonnegative integer  $a$ , the position  $(a + 4, a + 2, 2)$  is a P-position.*

*Proof.* We proceed by mathematical induction on  $a$ . Let  $\mathcal{P}$  denote the set of P-positions and  $\mathcal{N}$  denote the set of N-positions.

**Base Case ( $a = 0$ ):** For  $a = 0$ , the position is  $(4, 2, 2)$ . I verify that every legal move leads to a position in  $\mathcal{N}$ . Based on the trivial results, the following reachable positions are known P-positions:  $(3, 1, 1)$ ,  $(2, 2, 1)$ ,  $(3, 2, 0)$ , and  $(1, 0, 0)$ .

We analyze the possible moves from  $(4, 2, 2)$ , assuming that the player does not "chomp" the toxic square:

- **Row 1 moves:** A move to  $(3, 2, 2)$  allows a response to  $(3, 1, 1) \in \mathcal{P}$ . A move to  $(2, 2, 2)$  allows a response to  $(2, 2, 1) \in \mathcal{P}$ . A move to  $(1, 1, 1)$  allows a response to  $(1, 0, 0) \in \mathcal{P}$ .
- **Row 2 moves:** A move to  $(4, 1, 1)$  implies the response  $(3, 1, 1) \in \mathcal{P}$ . A move to  $(4, 0, 0)$  implies the response  $(1, 0, 0) \in \mathcal{P}$ .
- **Row 3 moves:** A move to  $(4, 2, 1)$  allows a response to  $(2, 2, 1) \in \mathcal{P}$ . A move to  $(4, 2, 0)$  allows a response to  $(3, 2, 0) \in \mathcal{P}$ .

Since every move from  $(4, 2, 2)$  leads to an N-position,  $(4, 2, 2)$  is itself a P-position.

**Inductive Step:** Assume that  $P_k = (k+4, k+2, 2)$  is a P-position for all integers  $0 \leq k < n$ . We must show that  $P_n = (n+4, n+2, 2)$  is a P-position. We consider all moves from  $P_n$  to a state  $S'$  and demonstrate a winning response  $S' \rightarrow P_{\text{target}} \in \mathcal{P}$ .

**Case 1: Opponent reduces Row 1.** If the opponent moves to  $(3, 3, 3)$ ,  $(2, 2, 2)$ , or  $(1, 1, 1)$ , these are all known N-positions (a rectangular state is always a Next-Player-win state). If the opponent moves to  $(x, \min(x, n+2), 2)$ , where  $x \geq 4$ . We respond with  $(x, x-2, 2)$ . This matches the form  $(k+4, k+2, 2)$  with  $k = x-4$ , which is a P-position by the inductive hypothesis. If the opponent moves to  $(3, 3, 2)$ , we respond with  $(3, 1, 1)$ .

**Case 2: Opponent reduces Row 2.** The opponent moves to  $(n+4, y, \min(y, 2))$ .

- If  $y \geq 2$ : The state is  $(n+4, y, 2)$ . Since  $y < n+2$ , we can reduce Row 1 to match the pattern. We move to  $(y+2, y, 2)$ . Letting  $k = y-2$ , we see that since  $y \geq 2$ ,  $0 \leq k < n$ . Thus, this position is in  $\mathcal{P}$  by hypothesis.
- If  $y < 2$ :
  - If  $y = 1$ , the state is  $(n+4, 1, 1)$ . We move to the known P-position  $(3, 1, 1)$ .
  - If  $y = 0$ , the state is  $(n+4, 0, 0)$ . We move to  $(1, 0, 0)$ .

**Case 3: Opponent reduces Row 3.** The opponent moves to  $(n+4, n+2, z)$  with  $z < 2$ .

- If  $z = 1$ , we move to  $(2, 2, 1) \in \mathcal{P}$ .
- If  $z = 0$ , we move to  $(n+3, n+2, 0) \in \mathcal{P}$ .

In all cases, a winning response exists. Thus,  $(n+4, n+2, 2)$  is a P-position.  $\square$

### 3.2.2 The P-positions family ( $a+11, a+7, a+5$ )

I extend my analysis to the case where the third row has length 5. I identify the infinite family  $(a+11, a+7, 5)$ .

**Theorem 2.** *For any nonnegative integer  $a$ , the position  $(a+11, a+7, 5)$  is a P-position.*

*Proof.* We use mathematical induction on  $a$ .

**Base Case ( $a = 0$ ):** For  $a = 0$ , the position is  $(11, 7, 5)$ . We must show that every legal move leads to an N-position. We rely on the following known P-positions for small third rows ( $r_3 < 5$ ) and exceptional cases for  $r_3 = 5$ :

- $r_3 = 4$ :  $(10, 6, 4), (9, 5, 4), (8, 4, 4), (7, 7, 4)$
- $r_3 = 3$ :  $(7, 4, 3), (6, 3, 3), (5, 5, 3)$
- $r_3 = 2$ :  $(k+4, k+2, 2)$  for  $k \geq 0$
- $r_3 = 1$ :  $(3, 1, 1), (2, 2, 1)$
- $r_3 = 5$  exceptions:  $(10, 5, 5), (9, 6, 5)$

Moves from  $(11, 7, 5)$ :

- **Row 1 moves:**
  - If  $x = 10$ :  $(10, 7, 5) \rightarrow (10, 5, 5) \in \mathcal{P}$ .
  - If  $x = 9$ :  $(9, 7, 5) \rightarrow (9, 7, 2) \in \mathcal{P}$ .
  - If  $x = 8$ :  $(8, 7, 5) \rightarrow (8, 4, 4) \in \mathcal{P}$ .
  - If  $x = 7$ :  $(7, 7, 5) \rightarrow (7, 7, 4) \in \mathcal{P}$ .
  - If  $x = 6$ :  $(6, 6, 5) \rightarrow (6, 3, 3) \in \mathcal{P}$ .
  - For cases  $x \leq 5$ , the resulting game state would be rectangular, so it will be an N-position.
- **Row 2 moves:** The opponent moves to  $(11, y, \min(y, 5))$ :
  - If  $y = 6$ :  $(11, 6, 5) \rightarrow (9, 6, 5) \in \mathcal{P}$ .

- For cases  $y \leq 5$ , the resulting game state would be that of the form  $(11, y, y)$ , which means the next move could be one of  $(10, 5, 5)$ ,  $(8, 4, 4)$ ,  $(6, 3, 3)$ ,  $(4, 2, 2)$ ,  $(3, 1, 1)$ , or  $(1, 0, 0)$ .
- **Row 3 moves:** The opponent reduces  $r_3$  to  $z < 5$ .
  - If  $z = 4$ :  $(11, 7, 4) \rightarrow (7, 7, 4) \in \mathcal{P}$ .
  - If  $z = 3$ :  $(11, 7, 3) \rightarrow (5, 5, 3) \in \mathcal{P}$ .
  - If  $z = 2$ :  $(11, 7, 2) \rightarrow (9, 7, 2) \in \mathcal{P}$ .
  - If  $z = 1$ :  $(11, 7, 1) \rightarrow (2, 2, 1) \in \mathcal{P}$ .
  - If  $z = 0$ :  $(11, 7, 0) \rightarrow (8, 7, 0) \in \mathcal{P}$ .

All moves lead to an N-position, so  $(11, 7, 5)$  is a P-position.

**Inductive Step:** Assume  $P_k = (k+11, k+7, 5)$  is a P-position for all integers  $0 \leq k < n$ . We show that  $P_n = (n+11, n+7, 5)$  is a P-position.

**Case 1: Opponent reduces Row 1.** If the opponent moves to a state where all three rows are equal, it will be an N-position. The opponent moves to  $(x, \min(x, n+7), 5)$ :

- If  $x \geq 11$ , let  $k = x - 11$ . We respond with  $(x, x - 4, 5)$ . This matches  $P_k = (k+11, k+7, 5)$ . Since  $x < n+11$ , we have  $k < n$ , so this is a known P-position. The move is valid because  $x - 4 \geq 7$  and  $x - 4 < \min(x, n+7)$ .
- If  $7 < x < 11$ :
  - $(10, 10, 5), (10, 9, 5)$ , or  $(10, 8, 5) \rightarrow (10, 5, 5) \in \mathcal{P}$ ,
  - $(9, 9, 5)$  or  $(9, 8, 5) \rightarrow (9, 6, 5) \in \mathcal{P}$ ,
  - $(8, 8, 5) \rightarrow (8, 4, 4) \in \mathcal{P}$ .
- If  $x \leq 7$  We use the specific responses derived in the base case.

**Case 2: Opponent reduces Row 2.** The opponent moves to  $(n+11, y, \min(y, 5))$ :

- If  $y \geq 7$ , let  $k = y - 7$ . We respond with  $(y+4, y, 5)$ . This matches  $P_k = (k+11, k+7, 5)$ . Since  $y < n+7$ ,  $k < n$ , so this is a P-position. The move is valid because  $y+4 < n+11$ .
- If  $y < 7$ :
  - If  $y = 6$ , we move to  $(9, 6, 5)$ .
  - For cases  $y \leq 5$ , the resulting game state would be that of the form  $(n+11, y, y)$ , which means the next move could be one of  $(10, 5, 5)$ ,  $(8, 4, 4)$ ,  $(6, 3, 3)$ ,  $(4, 2, 2)$ ,  $(3, 1, 1)$ , or  $(1, 0, 0)$ .

**Case 3: Opponent reduces Row 3.** The opponent moves to  $(n+11, n+7, z)$  with  $z < 5$ .

- If  $z = 4$ , we move to  $(7, 7, 4)$ .

- If  $z = 3$ , we move to  $(5, 5, 3)$ .
- If  $z = 2$ , we move to  $(n + 9, n + 7, 2)$ .
- If  $z = 1$ , we move to  $(2, 2, 1)$ .
- If  $z = 0$ , we move to  $(n + 8, n + 7, 0)$ .

In all cases, a winning response exists. Thus,  $(n + 11, n + 7, 5)$  is a P-position.  $\square$

## 4 Discussion and Future Work

### 4.1 Computational Limitations and State Space

The computational results presented in this paper were generated using a Python-based recursive solver with memoization. While this approach successfully identified P-positions for board widths up to  $n = 500$ , the analysis was constrained by significant memory and processing limitations.

The primary bottleneck encountered was the management of the state space. For a  $3 \times n$  board, the number of possible states corresponds to the number of partitions of integers into at most three parts, with each part less than or equal to  $n$ . Combinatorially, the size of this state space grows as a cubic function, specifically  $\binom{n+3}{3} \approx O(n^3)$ . While this is polynomial rather than exponential growth, the constant factors involved in a high-level language like Python resulted in substantial resource consumption.

For the maximum analyzed width of  $n = 500$ , the memoization cache (stored as a Python dictionary serialized to a .pk1 file) grew to approximately 132 MB. Because the solver relies on keeping the entire history of computed states in memory to ensure  $O(1)$  lookup times for transpositions, the memory footprint expands rapidly. Furthermore, Python’s Global Interpreter Lock (GIL) and object overhead limit the efficiency of the recursive backtracking algorithm compared to lower-level implementations. Consequently, extending the analysis beyond  $n = 500$  to verify the conjectures for larger  $n$  would require hardware with significantly higher RAM capacity or a fundamental change in the storage architecture.

### 4.2 Future Work and Potential Optimizations

To overcome the limitations described above and to further substantiate the theoretical conjectures proposed in this paper, future research should focus on both algorithmic optimization and theoretical proofs.

**Algorithmic and Storage Optimizations** The most immediate improvement would be to transition the solver from Python to a compiled language with manual memory management, such as C++ or Rust. This would eliminate the overhead associated with Python objects and allow for more efficient data structures.

Crucially, the current storage method (serializing a hash map of full state tuples) is inefficient. Since the status of any game state is binary (winning or losing), the cache could be optimized using a bit-array or a bitmap. By mapping each unique state tuple  $(r_1, r_2, r_3)$

to a unique integer index via a ranking function (a perfect hash), the entire state space could be stored in a flat binary array. This would reduce the storage requirement from hundreds of megabytes to mere bits per state, potentially allowing the computation of  $n$  into the thousands on standard consumer hardware.

Additionally, the algorithm could be refactored from a recursive approach to an iterative dynamic programming approach. By iterating through states in topological order (from the smallest total number of blocks to the largest), we could eliminate the recursion stack entirely, further reducing memory overhead.

**Theoretical Proofs** While the computational data provides strong evidence for the existence of linear families, rigorous mathematical proofs are required to elevate the "Gap Principle" and the "Existence of Linear Families" from conjecture to theorem. Future work should focus on constructing inductive proofs similar to those provided in Section 3.2.2 for the  $r_3 = 2$  and  $r_3 = 5$  cases. Specifically, proving the non-adjacency of linear families would be a significant contribution to the understanding of Chomp's structural properties.

**Generalization** Finally, the methods developed here could be extended to  $m \times n$  boards where  $m > 3$ . While  $4 \times n$  Chomp is significantly more complex, determining if similar "robust families" of P-positions emerge in higher dimensions remains an open and compelling question in combinatorial game theory.

## 5 Conclusion

The game of Chomp has long served as a deceptively simple entry point into the complexities of combinatorial game theory. As noted by Zeilberger, the  $3 \times n$  variation walks a "tightrope between the trivial and the impossible" [5]. While the Strategy Stealing Argument guarantees a first-player win, the specific winning moves have historically resisted simple closed-form analysis. This paper bridges the gap between abstract existence theorems and practical strategy through a rigorous computational analysis of the state space for  $n \leq 500$ .

By implementing a recursive solver with memoization, I moved beyond the reliance on Sprague-Grundy values to identify structural patterns directly within the set of P-positions. The central contribution of this work is the identification and formal proof of infinite linear families of P-positions. Specifically, I proved by mathematical induction that positions of the form  $(a + 4, a + 2, 2)$  and  $(a + 11, a + 7, 5)$  constitute robust families for all nonnegative integers  $a$ . These proofs validate the hypothesis that while the game tree is complex, pockets of linear predictability exist within the chaotic distribution of losing states.

Furthermore, my macroscopic analysis of optimal opening moves revealed a strict dichotomy between third-row and second-row winning responses. The empirical data strongly suggests that the sets of board sizes requiring these respective moves form a disjoint partition of the positive integers, supported by the conjectured ordering  $f(n) > g(n)$ . I also introduced the "Gap Principle," hypothesizing that infinite linear families for a fixed third row  $k$  and  $k + 1$  are mutually exclusive. This observation aligns with the theoretical periodicity established by Byrnes [4], providing concrete examples of the ultimate periodicity inherent in poset games.

While computational limitations constrained this study to  $n = 500$ , the clear emergence of periodic behavior provides a strong foundation for further inquiry. Future work requires transitioning to optimized low-level implementations to expand the search horizon and formulating rigorous proofs for the non-adjacency of linear families. Ultimately, this research provides the necessary empirical evidence to elevate the understanding of  $3 \times n$  Chomp from computational observation to generalized theorem.

## References

- [1] D. Gale, "A curious Nim-type game," *American Mathematical Monthly*, vol. 81, no. 8, pp. 876–879, 1974.
- [2] F. Schuh, "Spel van delers," *Nieuw Tijdschrift voor Wiskunde*, vol. 39, pp. 299–304, 1952.
- [3] M. Gardner, "Mathematical Games: Sim, Chomp and Race Track," *Scientific American*, vol. 228, no. 1, pp. 108–115, 1973.
- [4] S. Byrnes, "Poset game periodicity," *Integers: Electronic Journal of Combinatorial Number Theory*, vol. 3, p. G3, 2003.
- [5] D. Zeilberger, "Three-rowed Chomp," *Advances in Applied Mathematics*, vol. 26, no. 2, pp. 168–179, 2001.
- [6] J. H. Conway, *On Numbers and Games*, 2nd ed., A K Peters/CRC Press, 2000.
- [7] E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning Ways for Your Mathematical Plays*, vol. 1, 2nd ed., A K Peters/CRC Press, 2001.

## A Solver Source Code

The core of the Chomp P-position identification is a recursive function with memoization, implemented in Python. The solver determines if a state is a Winning (N) or Losing (P) position by checking if there is any move to an opponent's P-position. The results are cached using the `pickle` module to avoid re-computation and facilitate data analysis across sessions. The code used 0-indexed columns and rows, which are standard for Python.

```
1 import pickle
2 import os
3 import sys
4 from typing import Tuple
5
6 # CONFIGURATION
7 CACHE_FILENAME = "chomp_smart_cache.pkl"
8 win_cache = {} # Stores {state_tuple: is_win (bool)}
9
10 def load_cache():
11     """Loads the win_cache from a pickle file if it exists."""
12     global win_cache
13     if os.path.exists(CACHE_FILENAME):
14         try:
15             with open(CACHE_FILENAME, "rb") as f:
16                 win_cache = pickle.load(f)
17                 print(f"[System] Loaded {len(win_cache)} states from {CACHE_FILENAME}")
18         except (EOFError, pickle.UnpicklingError):
19             print("[System] Cache file corrupted or empty. Starting fresh.")
20         win_cache = {}
21     else:
22         print(f"[System] No existing cache found. Starting fresh.")
23         win_cache = {}
24
25 def save_cache():
26     """Saves the current win_cache to a pickle file."""
27     try:
28         with open(CACHE_FILENAME, "wb") as f:
29             pickle.dump(win_cache, f, protocol=pickle.HIGHEST_PROTOCOL)
30             print(f"[System] Saved {len(win_cache)} states to {CACHE_FILENAME}")
31     except Exception as e:
32         print(f"[System] Error saving cache: {e}")
33
34 def get_next_state(current_state: Tuple[int, ...], r: int, c: int) ->
35     Tuple[int, ...]:
36     """Calculates the new state tuple after a move (r, c)."""
37     next_state_list = list(current_state)
38
39     # A move at (r, c) truncates all rows >= r at column c
40     for i in range(r, len(next_state_list)):
41         next_state_list[i] = min(next_state_list[i], c)
```

```

42     # State Normalization: Remove trailing zeros
43     while next_state_list and next_state_list[-1] == 0:
44         next_state_list.pop()
45
46     return tuple(next_state_list)
47
48 def is_winning_state(state: Tuple[int, ...]) -> bool:
49     """
50     Recursively determines if a state is a winning (N) or losing (P)
51     position using memoization.
52     """
53
54     # 1. Check the Cache first
55     if state in win_cache:
56         return win_cache[state]
57
58     # 2. Base Case: The empty board () is a WIN (N-position).
59     if state == ():
60         win_cache[state] = True
61         return True
62
63     # 3. Iterate through all possible moves
64     can_move_to_losing_state = False
65
66     # Iterate through each row 'r'
67     for r in range(len(state)):
68         # Iterate through each column 'c' in that row (0-indexed)
69         for c in range(state[r]):
70
71             # The (0, 0) move (the poisoned cookie) is an immediate loss.
72             if r == 0 and c == 0:
73                 continue
74
75             # Get the resulting state from this move.
76             next_state = get_next_state(state, r, c)
77
78             # Recursively check if the opponent LOSES from that next_state
79             if not is_winning_state(next_state):
80                 # Found a move to a P-position for the opponent.
81                 # Current state is a WINNING (N) position.
82                 can_move_to_losing_state = True
83                 break
84
85             if can_move_to_losing_state:
86                 break
87
88     # 4. Cache and Return Result
89     win_cache[state] = can_move_to_losing_state
90     return can_move_to_losing_state
91
92 def run_analysis(n):
93     """
94     Main function to run the solver up to board width n.
95     In the $3 \times n$ context, this checks all (i, j, k) states

```

```

95     where i, j, k <= n.
96     """
97     print("\n--- Running Analysis ---")
98
99     # Example: Check the sequence of square boards
100    print(f"Checking 3-row boards (i, i, i) up to i={n}...")
101   for i in range(1, n + 1):
102     board = (i, i, i)
103     # Call the recursive function to compute the state if not cached
104     is_winning_state(board)
105
106   # Example: Analyze cached results
107   print(f"\nScanning cache for P-positions (Losing States)...")
108   p_positions = 0
109   for state, is_win in win_cache.items():
110     if not is_win and len(state) == 3 and state[0] <= n:
111       p_positions += 1
112
113   print(f"Found {p_positions} three-row P-positions up to width {n}.")
114
115 # EXECUTION BLOCK
116 if __name__ == "__main__":
117   load_cache()
118
119   # Increase recursion limit for deep searches
120   sys.setrecursionlimit(5000)
121
122   try:
123     # Run the analysis up to the limit N
124     run_analysis(500)
125
126   except RecursionError:
127     print("\n[System] Recursion depth exceeded. Increase sys.setrecursionlimit().")
128
129   except KeyboardInterrupt:
130     print("\n[System] Process interrupted by user.")
131
132   finally:
133     # ALWAYS save the cache at the end
134     save_cache()

```

Listing 1: Recursive Chomp Solver in Python

## B Full Table of P-Positions for a $3 \times 30$ board

The following table lists the identified P-positions  $(r_1, r_2, r_3)$  where  $r_3 \leq 10$ . The positions are ordered lexicographically.

P-Positions $(r_1, r_2, r_3)$					
(1, 0, 0)	(2, 1, 0)	(2, 2, 1)	(3, 1, 1)	(3, 2, 0)	(4, 2, 2)
(4, 3, 0)	(5, 3, 2)	(5, 4, 0)	(5, 5, 3)	(6, 3, 3)	(6, 4, 2)
(6, 5, 0)	(7, 4, 3)	(7, 5, 2)	(7, 6, 0)	(7, 7, 4)	(8, 4, 4)
(8, 6, 2)	(8, 7, 0)	(9, 5, 4)	(9, 6, 5)	(9, 7, 2)	(9, 8, 0)
(9, 9, 6)	(10, 5, 5)	(10, 6, 4)	(10, 8, 2)	(10, 9, 0)	(11, 6, 6)
(11, 7, 5)	(11, 9, 2)	(11, 10, 0)	(12, 7, 6)	(12, 8, 5)	(12, 9, 7)
(12, 10, 2)	(12, 11, 0)	(12, 12, 8)	(13, 7, 7)	(13, 8, 6)	(13, 9, 5)
(13, 11, 2)	(13, 12, 0)	(14, 8, 7)	(14, 9, 8)	(14, 10, 5)	(14, 11, 9)
(14, 12, 2)	(14, 13, 0)	(14, 14, 10)	(15, 8, 8)	(15, 10, 7)	(15, 11, 5)
(15, 13, 2)	(15, 14, 0)	(16, 9, 9)	(16, 10, 8)	(16, 11, 7)	(16, 12, 5)
(16, 14, 2)	(16, 15, 0)	(17, 10, 9)	(17, 11, 8)	(17, 12, 7)	(17, 13, 5)
(17, 14, 11)	(17, 15, 2)	(17, 16, 0)	(17, 17, 12)	(18, 10, 10)	(18, 12, 9)
(18, 13, 7)	(18, 14, 5)	(18, 16, 2)	(18, 17, 0)	(19, 11, 10)	(19, 12, 11)
(19, 13, 9)	(19, 14, 7)	(19, 15, 5)	(19, 16, 12)	(19, 17, 2)	(19, 18, 0)
(19, 19, 13)	(20, 11, 11)	(20, 12, 10)	(20, 14, 9)	(20, 15, 7)	(20, 16, 5)
(20, 18, 2)	(20, 19, 0)	(21, 12, 12)	(21, 13, 10)	(21, 15, 9)	(21, 16, 7)
(21, 17, 5)	(21, 18, 11)	(21, 19, 2)	(21, 20, 0)	(22, 13, 11)	(22, 14, 12)
(22, 15, 13)	(22, 16, 9)	(22, 17, 7)	(22, 18, 5)	(22, 19, 14)	(22, 20, 2)
(22, 21, 0)	(22, 22, 15)	(23, 13, 12)	(23, 14, 13)	(23, 15, 11)	(23, 16, 14)
(23, 17, 9)	(23, 18, 7)	(23, 19, 5)	(23, 20, 15)	(23, 21, 2)	(23, 22, 0)
(23, 23, 16)	(24, 13, 13)	(24, 15, 12)	(24, 16, 11)	(24, 18, 9)	(24, 19, 7)
(24, 20, 5)	(24, 22, 2)	(24, 23, 0)	(25, 14, 14)	(25, 16, 13)	(25, 17, 11)
(25, 19, 9)	(25, 20, 7)	(25, 21, 5)	(25, 23, 2)	(25, 24, 0)	(26, 15, 14)
(26, 16, 15)	(26, 17, 13)	(26, 18, 16)	(26, 19, 11)	(26, 20, 9)	(26, 21, 7)
(26, 22, 5)	(26, 23, 17)	(26, 24, 2)	(26, 25, 0)	(26, 26, 18)	(27, 15, 15)
(27, 17, 14)	(27, 18, 13)	(27, 20, 11)	(27, 21, 9)	(27, 22, 7)	(27, 23, 5)
(27, 25, 2)	(27, 26, 0)	(28, 16, 16)	(28, 17, 15)	(28, 18, 14)	(28, 21, 11)
(28, 22, 9)	(28, 23, 7)	(28, 24, 5)	(28, 26, 2)	(28, 27, 0)	(29, 17, 16)
(29, 18, 15)	(29, 19, 17)	(29, 20, 14)	(29, 21, 18)	(29, 22, 11)	(29, 23, 9)
(29, 24, 7)	(29, 25, 5)	(29, 26, 19)	(29, 27, 2)	(29, 28, 0)	(29, 29, 20)
(30, 17, 17)	(30, 19, 15)	(30, 20, 16)	(30, 21, 14)	(30, 23, 11)	(30, 24, 9)
(30, 25, 7)	(30, 26, 5)	(30, 28, 2)	(30, 29, 0)		