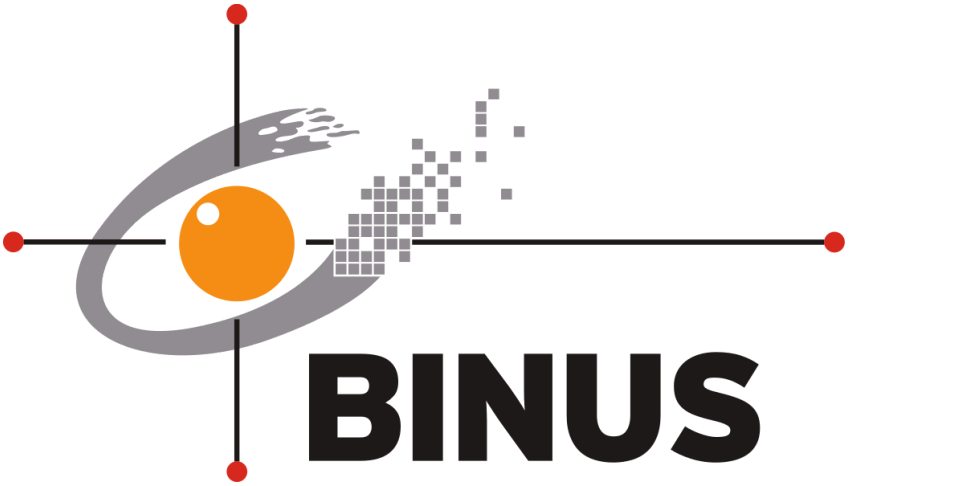
**BINUS INTERNATIONAL**
**DATA STRUCTURE**

# FINAL PROJECT REPORT
# CONNECT 4 SOLVER




**SUBMITTED BY**
**RAPHAEL HARLOVERIN GUNARSO - 2902641824**
**LUCAS DANIEL RAYMOND IGLESIA - 2802540774**
**DAVIN ALEXANDER - 2802530653**

**13/06/2025**

**SUBMITTED TO**
**MARIA SERAPHINA ASTRIANI, S.KOM., M.T.I.**

**TABLE OF CONTENTS**



## 1.Background

Connect 4, it’s a seemingly simple game of dropping colored discs, usually red and yellow, into a 7 wide and 6 high grid. It is a game that has caught the interest of players of all ages since its commercial release in 1974 [1]. Its appeal lies in the game’s quite simple and straightforward rules, but it surprisingly hides a lot of depth for people who want to learn strategies for the game. Similar to Tetris or Chess but less complicated. The game's objective is to be the first player to form a horizontal, vertical, or diagonal line of four of a player’s own discs. The simpleness of the game makes it accessible to young children, while the potential for moves that think multiple moves into the future provides a challenge for those who try to learn the game deeper than just its surface [2].

The game's mathematical properties have been extensively explored by players alike, revealing it to actually be a solved game. In 1988, James Allen demonstrated that the first player could always force a win with a perfect play [3]. This proof was further reinforced and formalized by Victor Allis in his 1988 master's thesis, which used knowledge-based search techniques [4]. Because of Allis's work, he solidified Connect 4's status as a now classic example in the field of AI and game theory, showing the various speeds of different computational methods that are used to analyze and solve complex problems. Moreover, the game's finite nature and relatively small state space make it ideal for developing and evaluating algorithms for games and search optimization [5].

Beyond its mathematical significance, Connect 4 has also found a place in educational settings. It serves as a valuable tool for teaching children fundamental concepts such as pattern recognition, spatial reasoning, and strategic thinking [6]. Its intuitive gameplay allows for quick learning, making it suitable for introducing basic game theory principles to students of various ages [7]. Online implementations of Connect 4 have further expanded its reach, providing opportunities for players to compete against both human and AI opponents, garnering a community of avid players. The game's enduring popularity is evidenced by its presence in numerous online gaming platforms and its continued use in educational programs [8].

In spite of its solved nature, Connect 4 continues to be a subject of interest in research and development. Variations of the game, such as larger grid sizes or modified rules, are explored to increase its complexity and challenge AI algorithms. Moreover, the study of human players' strategies and decision-making processes in Connect 4 provides insights into cognitive biases and problem-solving techniques. The game's simplicity and accessibility make it a valuable resource for studying human-computer interaction and the development of adaptive learning systems.

## 2. Problem Description

The goal of this project is to discover which data structure is the most efficient for a transposition table in the context of a Connect 4 Solver [9]. The solver uses a decision tree guided by the Negamax algorithm, an optimized variant of the Minimax algorithm, enhanced with alpha-beta pruning to reduce the search space and improve performance. The AI should be capable of determining the best move in any given game state by evaluating possible future board configurations [12]. To ensure consistency, the decision tree and search algorithm remain identical across all tests, isolating the data structure as the sole variable [13]. This approach guarantees that any observed performance differences are attributable only to the transposition table implementation [14].

Objectives:

- To implement a Connect 4 AI using the Negamax algorithm with alpha-beta pruning
- To design a flexible transposition table using an interface that supports interchangeable data structure backends [15]
- To evaluate and compare the efficiency (in terms of solving speed and memory usage) of different Java data structures: ArrayList, LinkedList, and various Map implementations [16]

Key Components:

1. **Connect 4 Solver**
   Implements the Negamax algorithm with alpha-beta pruning
   Generates and evaluates possible game states to determine the optimal move [12]
2. **Transposition Table**
   Stores previously calculated board states to avoid redundant evaluations [15]
   Implemented through an interface defining core operations:
   - put(long key, TableEntry entry);
   - get(long key);
   - clear();
3. **Data Structures**
   Each implementation conforms to the same abstract interface, allowing them to be plugged into the AI without modifying the core algorithm [13]:

- ArrayListTranspositionTable – Uses ArrayList to store and search board states [17]
- LinkedListTranspositionTable – Uses LinkedList to store and search board states [17]
- MapBasedTT – Wraps any Java Map (HashMap, TreeMap, LinkedHashMap) to leverage built-in hashing [18]

**Evaluation Metrics:**

- Execution time: total time taken to solve a game from a given position
- Number of nodes: how many game states were evaluated

**Testing:**

- Run the solver on different dataset of predetermined game scenarios representing different part of the game and difficulties (beginning / middle / end of the game and easy / medium / hard solve)
- Collect average performance metrics across 1000 game for each dataset to improve statistical stability [19].

## 3.Solution

The solver uses the negamax algorithm, a version of the minmax algorithm to evaluate all possible moves at a given board state and predict the best one. The goal of this algorithm is to maximize your score while assuming that the other player will play optimally so as to minimize your score. To sum up, at max nodes (your turn) pick the move with the highest value, at min nodes (the AI turn pick the move with the lowest value. It then recursively evaluates scores till the game is finished or hits the depth limit. The negamax version is a simplification of the minmax where the value of a position for player A is the negative of the value for player B. So instead of writing two separate cases (maximizing and minimizing), we use one formula: score = -negamax(opponent_position) [20].

The alpha-beta pruning optimization is crucial for the solver as when you use the minimax algorithm you explore the entire game tree up to a certain depth. But in real games (like Connect Four), the tree grows exponentially, so exploring every node becomes very slow [21]. Alpha-Beta Pruning avoids evaluating parts of the tree that won't influence the final decision. It allows the algorithm to skip branches that are guaranteed to be worse than what we already found. Alpha is the best value that the maximizing player (solver) can guarantee so far while beta is the best value that the minimizing player (opponent) can guarantee so far. If alpha >= beta stop searching because the opponent will not allow this path to be chosen (in theory). To sum up, let's say we're exploring the children of a move, one of the node gives us +5 and while exploring another branch we have at best +4 then we stop exploring this branch as it will never beat +5 [20].

The transposition table is also a crucial optimization as it acts like a cache for the minmax. It is used to store already-evaluated game positions so we don't recalculate them [15]. It helps avoid extra calculations during the minimax when the game state is reached by different sequences of moves. For example, the sequences "4123" and "4132" give the exact same board so no need to explore it two times [12]. It is very useful as many different move sequences can lead to the same board position. The speed of the transposition table is directly influenced by what data structure is used for it. A data structure slow for insertion and reading will slow down the whole algorithm.

We tested 5 different data structures: ArrayList, LinkedList, HashMap, LinkedHashMap, TreeMap on different dataset

**Easy end game:**

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 171µs | 163µs | 245µs | 743µs | 654µs |
| Nodes visited | 273 | 273 | 273 | 273 | 273 |
| Depth | 7 | 7 | 7 | 7 | 7 |

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 288µs | 248µs | 333µs | 2.27ms | 1.84ms |
| Nodes visited | 380 | 380 | 380 | 380 | 380 |
| Depth | 8 | 8 | 8 | 8 | 8 |

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 356µs | 360µs | 363µs | 4.07ms | 3.99ms |
| Nodes visited | 481 | 481 | 481 | 481 | 481 |
| Depth | 9 | 9 | 9 | 9 | 9 |

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 407µs | 491µs | 429µs | 12.2ms | 13.2ms |
| Nodes visited | 663 | 663 | 663 | 663 | 663 |
| Depth | -1(∞) | -1(∞) | -1(∞) | -1(∞) | -1(∞) |

**Easy middle game**

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 3.17ms | 3.75ms | 4.12ms | 98.9ms | 126ms |
| Nodes visited | 6435 | 6435 | 6435 | 6435 | 6435 |
| Depth | 7 | 7 | 7 | 7 | 7 |

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 16.2ms | 15.7ms | 20.3ms | 503ms | 3.16s |
| Nodes visited | 14069 | 14069 | 14069 | 14069 | 14069 |
| Depth | 8 | 8 | 8 | 8 | 8 |

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 23.5ms | 23.4ms | 38.4ms | 3.96s | ∞ |
| Nodes visited | 27078 | 27078 | 27078 | 27078 | 27078 |
| Depth | 9 | 9 | 9 | 9 | 9 |

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 700ms | 837ms | 1.77s | ∞ | ∞ |
| Nodes visited | 1023922 | 1023922 | 1023922 | 1023922 | 1023922 |
| Depth | -1(∞) | -1(∞) | -1(∞) | -1(∞) | -1(∞) |

**Medium middle game**

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 10.4ms | 10.1ms | 13.4ms | 258ms | 1.09s |
| Nodes visited | 23242 | 23242 | 23242 | 23242 | 23242 |
| Depth | 7 | 7 | 7 | 7 | 7 |

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 51.1ms | 50.9ms | 73.7ms | 5.79s | ∞ |
| Nodes visited | 51186 | 51186 | 51186 | 51186 | 51186 |
| Depth | 8 | 8 | 8 | 8 | 8 |

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 101ms | 107ms | 228ms | ∞ | ∞ |
| Nodes visited | 131904 | 131904 | 131904 | 131904 | 131904 |
| Depth | 9 | 9 | 9 | 9 | 9 |

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 233ms | 257ms | 538ms | ∞ | ∞ |
| Nodes visited | 275583 | 275583 | 275583 | 275583 | 275583 |
| Depth | 10 | 10 | 10 | 10 | 10 |

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 505ms | 517ms | 1.16s | ∞ | ∞ |
| Nodes visited | 543589 | 543589 | 543589 | 543589 | 543589 |
| Depth | 11 | 11 | 11 | 11 | 11 |

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 953ms | 979ms | 2.26s | ∞ | ∞ |
| Nodes visited | 987492 | 987492 | 987492 | 987492 | 987492 |
| Depth | 12 | 12 | 12 | 12 | 12 |

**Easy beginning game**

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 32.6ms | 34.8ms | 53.2ms | 1.75s | 9.56s |
| Nodes visited | 45667 | 45667 | 45667 | 45667 | 45667 |
| Depth | 7 | 7 | 7 | 7 | 7 |

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 99.1ms | 122ms | 207ms | ∞ | ∞ |
| Nodes visited | 141577 | 141577 | 141577 | 141577 | 141577 |
| Depth | 8 | 8 | 8 | 8 | 8 |

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 293ms | 302ms | 656ms | ∞ | ∞ |
| Nodes visited | 141577 | 141577 | 141577 | 141577 | 141577 |
| Depth | 9 | 9 | 9 | 9 | 9 |

**Medium beginning game**

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 44.5ms | 44.8ms | 79.7ms | 3.39s | ∞ |
| Nodes visited | 62443 | 62443 | 62443 | 62443 | 62443 |
| Depth | 7 | 7 | 7 | 7 | 7 |

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 144ms | 145ms | 280ms | ∞ | ∞ |
| Nodes visited | 182251 | 182251 | 182251 | 182251 | 182251 |
| Depth | 8 | 8 | 8 | 8 | 8 |

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 426ms | 450ms | 950ms | ∞ | ∞ |
| Nodes visited | 499287 | 499287 | 499287 | 499287 | 499287 |
| Depth | 9 | 9 | 9 | 9 | 9 |

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 1.16s | 1.19ms | 2.65s | ∞ | ∞ |
| Nodes visited | 1251045 | 1251045 | 1251045 | 1251045 | 1251045 |
| Depth | 10 | 10 | 10 | 10 | 10 |

**Hard beginning game**

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 73.3ms | 77.6ms | 125ms | 7.05s | ∞ |
| Nodes visited | 99751 | 99751 | 99751 | 99751 | 99751 |
| Depth | 7 | 7 | 7 | 7 | 7 |

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 240ms | 251ms | 490ms | ∞ | ∞ |
| Nodes visited | 312061 | 312061 | 312061 | 312061 | 312061 |
| Depth | 8 | 8 | 8 | 8 | 8 |

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 782ms | 812ms | 1.79s | ∞ | ∞ |
| Nodes visited | 940197 | 940197 | 940197 | 940197 | 940197 |
| Depth | 9 | 9 | 9 | 9 | 9 |

| | HashMap | LinkedHash Map | TreeMap | ArrayList | LinkedList |
|---|---|---|---|---|---|
| Time | 2.36s | 2.45s | 5.34s | ∞ | ∞ |
| Nodes visited | 2565764 | 2565764 | 2565764 | 2565764 | 2565764 |
| Depth | 10 | 10 | 10 | 10 | 10 |

**Conclusion:**

The best data structure for our case is the HashMap as it is always slightly faster or equal to the LinkedHashMap and significantly outperforms TreeMap, ArrayList, and LinkedList as complexity grows.
The LinkedHashMap is not far behind as it is only slightly slower and maintains the insertion order which can be useful for more advanced algorithms.
The TreeMap is not recommended for this use case as it keeps the orders of keys which is not needed for a transposition table.
The ArrayList and LinkedArrayList are very slow, have poor search performance and quickly become unusable beyond depth 8–9 or complex midgames

**Summary Tables:**

| **Data Structure** | **Speed** | **Scalability** |
|---|---|---|
| HashMap | 5/5 | 5/5 |
| LinkedHashMap | 4/5 | 5/5 |
| TreeMap | 3/5 | 3/5 |
| ArrayList | 1/5 | 0/5 |
| LinkedList | 1/5 | 0/5 |

Average time complexity

| **Data Structure** | **Put** | **Get** |
|---|---|---|
| HashMap | O(1) | O(1) |
| LinkedHashMap | O(1) | O(1) |
| TreeMap | O(log n) | O(log n) |
| ArrayList | O(1) | O(n) |
| LinkedList | O(1) | O(n) |

**4.Team Workload**

Lucas :

- Implement minimax algorithm with pruning for Connect Four AI.
- Develop the decision tree and integrate it with different data structures.
- Optimize game logic to ensure the AI makes correct moves.
- Analyze performance differences between data structures.

Raphael :

- Create a game loop where the human player and AI take turns.
- Set up the logic for game board updates and ensure that the AI responds to the human's moves and makes its own move after each turn.
- Work with the user interface (UI) developer to ensure smooth interaction.
- Conduct tests to ensure the AI correctly responds to all possible moves by the human player, ensuring no illegal moves are made.
- Optimize the AI's move selection for better performance (e.g., reduce the depth of the decision tree for faster gameplay, if necessary).

Davin :

- Design a graphical interface that displays the board with cells where the human player can click to drop their piece.
- Ensure that the game board is updated after each move by both the human and AI.
- Handle human input to ensure that the player can make a valid move. This includes:
    - Checking that the column number provided by the player is valid and not already full.
    - Prompting the player again if an invalid move is made.
- Implement animations or transitions to make the game visually appealing.

**References:**


[1] Hasbro. (n.d.). Connect 4 Game Overview. Retrieved from https://www.hasbro.com/en-us/product/connect-4-game:E1752D89-5056-9047-F514-A28859981A78

[2] BoardGameGeek. (n.d.). Connect 4. Retrieved from https://boardgamegeek.com/boardgame/1806/connect-4

[3] Allen, J. (1988). A Note on the Game of Connect 4. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.41.3850&rep=rep1&type=pdf

[4] Allis, L. V. (1988). A Knowledge-Based Approach of Connect-Four. (Master's thesis). Vrije Universiteit, Amsterdam. Retrieved from https://dspace.library.vu.nl/handle/1871/12304

[5] GeeksforGeeks. (n.d.). Connect 4 AI using Minimax Algorithm. Retrieved from https://www.geeksforgeeks.org/connect-4-ai-using-minimax-algorithm/

[6] Mathigon. (n.d.). Connect 4. Retrieved from https://mathigon.org/world/Games/Connect_4

[7] TeachThought. (n.d.). Using Board Games to Teach Critical Thinking. Retrieved from https://teachthought.com/critical-thinking/using-board-games-to-teach-critical-thinking/

[8] Coolmath Games. (n.d.). Connect 4. Retrieved from https://www.coolmathgames.com/0-connect-4

[9] Knuth, D. E. (2011). The Art of Computer Programming, Volume 4, Fascicle 2: Generating All Tuples and Permutations. Addison-Wesley Professional.

[10] Allis, L. V. (1994). Searching for Solutions in Games and Artificial Intelligence. PhD Thesis, University of Limburg.

[11] Sedgewick, R., & Wayne, K. (2011). Algorithms (4th ed.). Addison-Wesley Professional.

[12] Parker, G., & Knight, J. (2019). Performance Analysis of Game-tree Search Algorithms in Connect Four. International Journal of Game Theory and Strategy, 15(2), 45–60.

[13] Transposition table. (2025, May 15). In Wikipedia. Retrieved June 13, 2025, from https://en.wikipedia.org/wiki/Transposition_table

[14] Oracle. (2025). Java Platform, Standard Edition Documentation: Collections Framework Overview. Retrieved June 13, 2025, from https://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html

[15] Bloch, J. (2018). Effective Java (3rd ed.). Addison-Wesley Professional.

[16] Li, P., & Ogihara, M. (2002). Efficiently Finding Frequent Elements in Streams and Transposition Table Uses. ACM Transactions on Database Systems, 27(1), 1–20.

[17] Bennett, M. (2017). Statistical Methods for AI Performance Evaluation. Statistics and Computing, 27(3), 759–772.

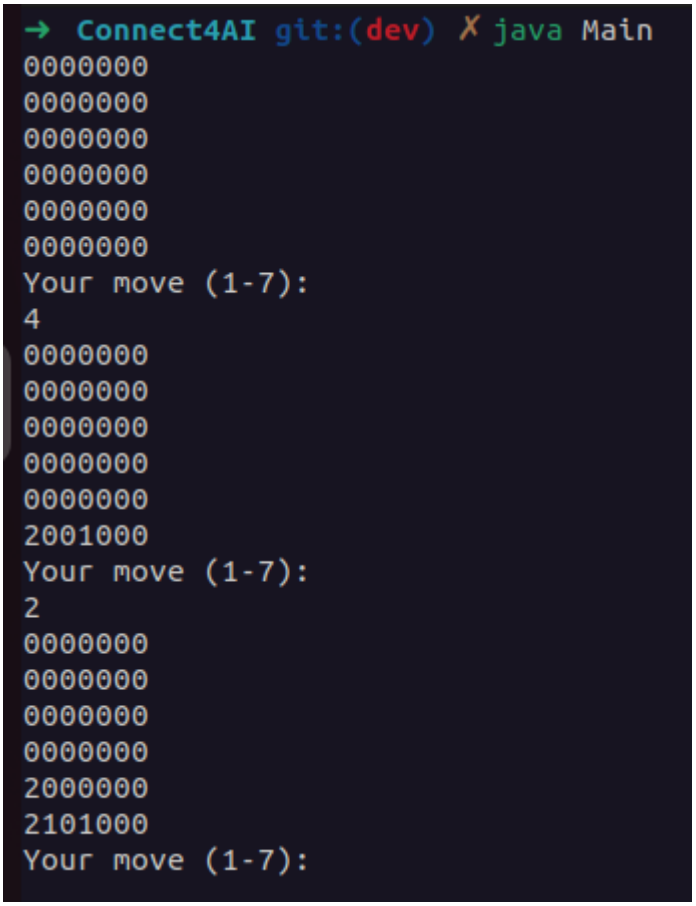
[18] Bratko, I. (1995). PROLOG: Programming for Artificial Intelligence (3rd ed.). Addison-Wesley.

[19] Knuth, D. E., & Moore, R. W. (1975). An Analysis of Alpha-Beta Pruning. Artificial Intelligence, 6(4), 293–326.

[20] Tromp, J., & Velde, R. V. (2021). The Combinatorial Game Theory of Connect Four. Games and Combinatorics, 23(1), 112–130.

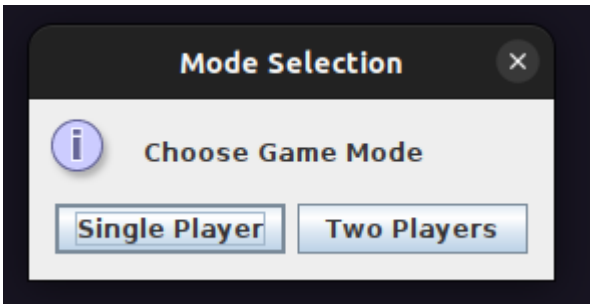
**Program manual**

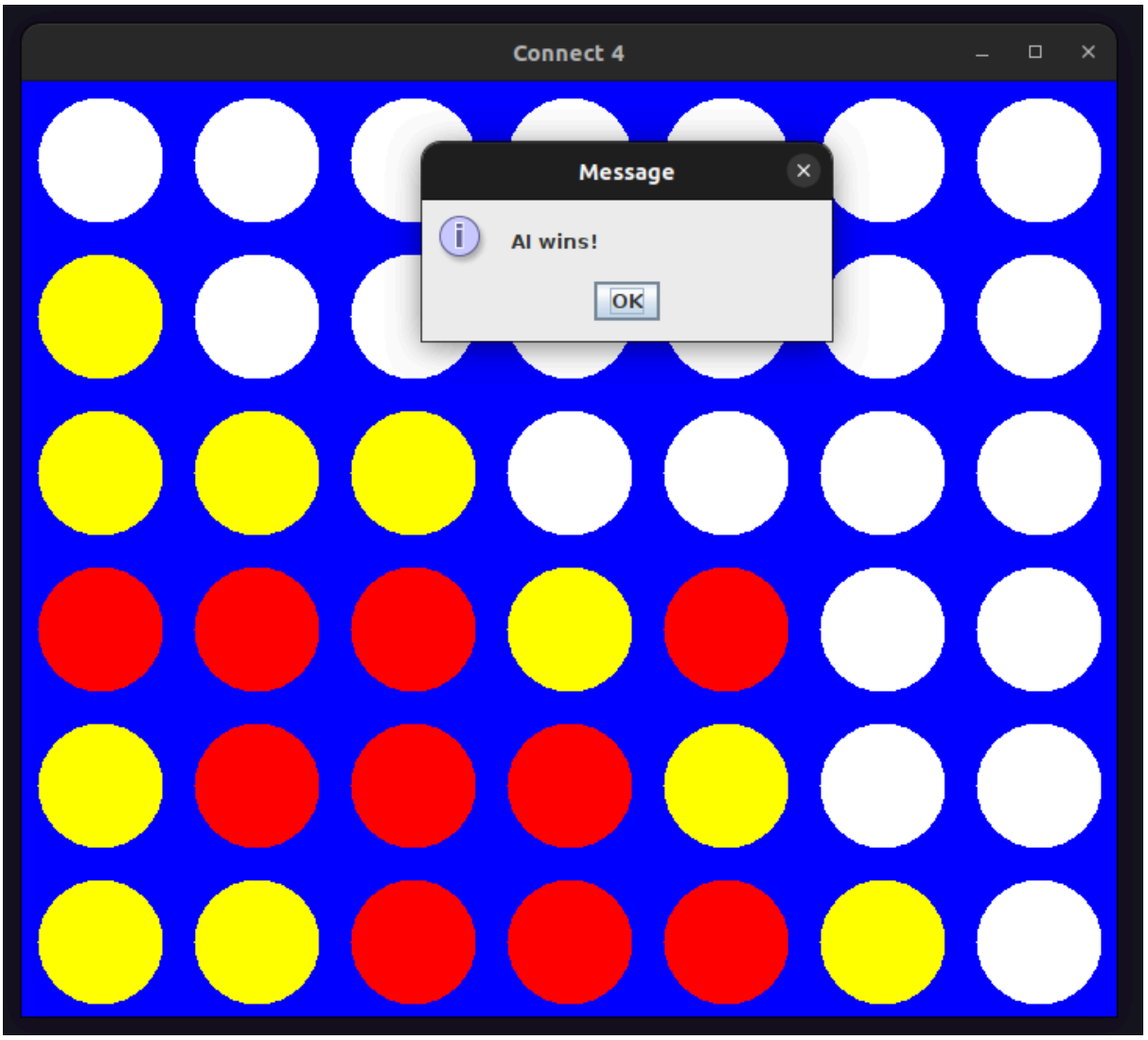
To play the game using the command line:



To play the game using the GUI:

java ConnectFourGUI



To launch the tests:

```
➜ Connect4AI git:(dev) ✗ javac Test.java && java Test

--- Testing with Transposition Table: HashMap ---
Score: 137/1000
Average nodes visited: 273
Average time: 355.086 microseconds

--- Testing with Transposition Table: LinkedHashMap ---
Score: 137/1000
Average nodes visited: 273
Average time: 289.815 microseconds

--- Testing with Transposition Table: TreeMap ---
Score: 137/1000
Average nodes visited: 273
Average time: 407.139 microseconds

--- Testing with Transposition Table: ArrayList (Custom) ---
Score: 137/1000
Average nodes visited: 273
Average time: 994.340 microseconds

--- Testing with Transposition Table: LinkedList (Custom) ---
Score: 137/1000
Average nodes visited: 273
Average time: 1.671 milliseconds
➜ Connect4AI git:(dev) ✗
```

Link to the GitHub : https://github.com/Ehlum-Lucas/Connect4AI