ISSN 2306-5478

УДК 004.832.24:519.83 https://doi.org/10.31713/vt1202613

Poryev G.V. [1; ORCID ID: 0000-0003-4825-0917],
Doctor of Technical Sciences, Associate Professor
Zubyk Ya. Ya. [2; ORCID ID: 0000-0002-0802-3552],
Senior Lecturer
[1] Taras Shevchenko National University of Kyiv, Kyiv
[2] National University of Water Management and Environmental Management, Rivne

EXHAUSTIVE HEURISTIC SEARCH TO SOLVE DETERMINISTIC FINITE ZERO-SUM GAME

Wild-Tic-Tac-Toe (W3T) is a combinatorial extension of the classic Tic-Tac-Toe game, distinguished by the rule that players may place either marks on each turn. This added flexibility introduces considerable strategic complexity and has hitherto left the game's theoretical outcome unresolved. In this study, we rigorously address the solvability of W3T by constructing a complete representation of its state space and all legal move sequences using a specially designed directed acyclic graph structure termed the Atlas. The game's finite and deterministic nature enables exhaustive traversal of all valid positions, culminating in a full labeling of states via retrograde analysis. We define and apply outcome labels—win, loss, or draw—propagated from terminal positions upward through the graph. Our implementation leverages a ternary encoding scheme to uniquely represent board states, with efficient memory access and state reuse facilitated by integer-indexed arrays. The results demonstrate that W3T is not only solvable in the classical sense of perfect-information zero-sum games but is also a forced win for the first player under optimal play. This work contributes a novel methodology for solving similar finite games and underscores the importance of structural optimization in state-space exploration. The proposed framework exemplifies a form of symbolic artificial intelligence applied to exhaustive state-space reasoning, offering a transparent alternative to statistical learning approaches commonly used in decision-making systems for the Internet of Everything.

***Keywords*: heuristic, artificial intelligence, game theory, state-space reasoning, wild-tic-tac-toe, exhaustive search, minimax, alpha-beta pruning**

Introduction. Wild-Tic-Tac-Toe (W3T), a variation of the classic Tic-Tac-Toe game, presents an intriguing yet largely unexplored combinatorial challenge. Unlike its well-known counterpart, which has been extensively analyzed and is trivially solvable under optimal play [1] [2] [3], W3T remains an open question in the realm of computational game theory. Despite its simplicity, no comprehensive effort has been made to determine whether the W3T game can be definitively solved, leaving a gap in the study of small, finite games that are otherwise prime candidates for rigorous analysis.

Small-scale games like W3T provide an ideal proving ground for algorithmic approaches to game solving. Their limited state space—while still complex enough to be nontrivial—allows for exhaustive enumeration, strategic evaluation, and the application of fundamental game-theoretic principles. These characteristics make them particularly useful for demonstrating solving techniques, such as minimax search, alpha-beta pruning, and heuristic evaluation functions. Additionally, studying such games contributes to a broader understanding of decision-making processes in more complex combinatorial environments.

This article presents an investigation into the solvability of W3T, detailing the construction of a complete game tree or similar structure, the application of heuristics, and the strategies used to evaluate optimal play. By addressing a previously unanswered question in game theory, this work not only deepens our understanding of W3T but also reinforces the role of finite games as valuable models for computational problem-solving.

From the perspective of artificial intelligence, W3T serves as a microcosm of autonomous decision making and heuristic search in finite environments. The techniques demonstrated here—state-space enumeration, retrograde labeling, and optimization of successor selection—are foundational to symbolic AI, which underlies intelligent agents and planning systems within the Internet of Everything. Thus, while the experimental domain is a game, the computational principles extend naturally to decision modules operating in distributed AI environments.

Discussion of Related Works. Despite the longstanding attention that classic Tic-Tac-Toe has received in both pedagogical and computational contexts, there appears to be a marked absence of systematic investigations into its Wild variant. The distinction between the two versions, as elaborated in the Materials and Methods section, is not merely cosmetic; it fundamentally alters the strategic landscape of the game. The freedom to choose either mark at each turn introduces a branching factor and combinatorial depth absent in

ISSN 2306-5478

the classic formulation, thereby rendering most of the established analyses inapplicable or insufficient. A survey of existing literature reveals that while numerous studies explore strategies, heuristics, and even evolutionary algorithms for classic Tic-Tac-Toe [3] [1], no work to date has rigorously established the solvability of Wild-Tic-Tac-Toe, nor proposed an exhaustive computational framework for analyzing its full state space. This research therefore fills a critical void by offering what is, to our knowledge, the first comprehensive solution of W3T under the assumption of perfect play.

A substantial body of work demonstrates that small, finite, perfect-information games can be solved by exhaustive enumeration and retrograde analysis. Landmark examples include checkers (English draughts), solved to a draw via massive search coupled with database methods [4], and Nine Men's Morris and its close relatives, for which complete or ultra-strong solutions have been constructed [5, 6]. These studies validate retrograde win/draw/loss labeling at scale and motivate careful indexing of large directed acyclic graphs (DAGs) of positions—methodological choices that align with our Atlas design.

In connection games, the canonical benchmark is Connect Four, solved by knowledge-guided search [7]. Allis and colleagues also introduced proof-number search (PN-search), a best-first method for establishing game-theoretic values in AND/OR trees [8]. While our approach is exhaustive rather than PN-guided, both paradigms share the objective of certifying correctness over complete state graphs.

The endgame-table base literature in chess provides further precedent for space-efficient retrograde analysis on enormous state spaces. Nalimov's indexing schemes and, later, de Man's Syzygy WDL/DTZ split exemplify (i) compact encodings of states, (ii) separation of value classes (win/draw/loss) from distance metrics, and (iii) data structures designed for high-throughput probing—principles that informed our decision to store outcomes separately from successor links in the Atlas [9, 10].

From a search-algorithmic standpoint, the classical line from minimax to alpha-beta pruning remains foundational for perfect-information games, with deep analyses of alpha–beta efficiency by Knuth and Moore [11]. Subsequent best-first alternatives include Stockman's SSS* [12] and practical refinements such as AB–SSS* and MTD(f) [13, 14]. In parallel, transposition tables and Zobrist hashing have become standard to collapse identical positions reached by different sequences into a single node—transforming trees into DAGs [15]. Our ternary encoding and deduplication choices are consistent with these practices.

The distributed/exhaustive-search community has repeatedly shown that large impartial or nearimpartial games are tractable with proper state sharing and indexing. Romein and Bal's strong solution of Awari via parallel retrograde analysis is a representative exemplar of engineering rigor for

enumerating and labeling complete state spaces; the system design choices there (work distribution, memory locality) resonate with our use of cache-friendly, integer-indexed arrays in Atlas [16].

In terms of symmetry handling and canonicalization, the General Game Playing (GGP) community offers general methods for symmetry detection and exploitation during tree/DAG construction [17, 18]. Our reduction of equivalent sequences by dihedral symmetries of the 3×3 board is conceptually aligned with symmetry-aware indexing in GGP and can be further tightened by canonicalization passes.

Regarding Tic-Tac-Toe variants, the most closely related rigorous line is the misère, X-only variant known as Notakto, where Plambeck and Whitehead provide a complete analysis via misère quotient theory [19]. Although Notakto and W3T share a departure from standard alternating-mark assignment, the underlying combinatorial structure and objective differ substantially; existing Notakto analyses do not transfer to W3T's rule set. Our contribution is thus orthogonal to that literature.

We also note pedagogical expositions of W3T strategy (e.g., Elran), which assert a first-player win through illustrative lines [20]. These are valuable for intuition and outreach but do not constitute an exhaustive computational proof or a formal state-space analysis. The present work closes that gap by building, labeling, and querying the complete move DAG.

As for scalability and broader context, modern advances for very large games (Go, chess, shogi) rely on statistical search (Monte-Carlo tree search) guided by deep neural networks (AlphaGo Zero/AlphaZero) [21, 22]. These methods optimize playing strength rather than formal proof. Our objective differs: we compute a certified solution for a finite game by full traversal and retrograde labeling. The two paradigms are complementary; for larger finite impartial games, hybridizing Atlas-style canonicalization with statistical guidance is a promising direction for future work. Finally, contemporary theoretical work on zero-sum random games on directed graphs further motivates principled DAG formulations for game solving [23].

This also situates the present work within the classical AI tradition of symbolic reasoning and search, complementing the data-driven paradigm that dominates modern autonomous systems.

Materials and Methods. In order to provide better understanding of the context or this research, we shall reflect on the differences between the classic Tic-Tac-Toe and W3T. Both games a played on a 3-by-3 grid, with players placing their marks on an empty cell in alternating turns, aiming to form a line of three marks in a row, column, or diagonal, and to prevent the opponent from doing the same. The game ends when one player achieves this goal or when the grid is filled without a winner, resulting in a draw.

 ISSN 2306-5478

The cardinal difference between classic Tic-Tac-Toe and W3T is that in classic game each player is assigned their one of the two marks, usually a nought (0) or cross (X) and can only place that mark on the board. In W3T, however, players can place any of the two marks they choose each turn. This additional freedom introduces a new layer of complexity to the game, as players must now consider not only their own strategy but also the potential strategies of their opponent. This added dimensionality makes W3T a more challenging game to solve than its classic counterpart, as it requires a deeper understanding of the possible interactions between players and the implications of each move.

Ternary encoding. In order to represent the game state in a compact and efficient manner, we have developed a trit-based (Like bit is a binary digit, trit is a ternary digit) encoding scheme for W3T. This encoding uses a base-3 numeral system to represent the state of each cell on the board, with each cell taking on one of three possible values: empty, cross, or nought. To encode a game situation the 3-by-3 grid is flattened into a 9-element array or trits, with each element representing the state of a single cell. This array is then converted into a base-10 integer, which serves as a unique identifier for the game state.

```
type TSituation struct {
 Cell [9]uint8
}

var power3 = [9]uint16{1, 3, 9, 27, 81, 243, 729, 2187, 6561}

func (sit *TSituation) Encode() (code uint16) {
 for pos, trit := range sit.Cell {
  code += power3[pos] * trit
 }
 return code
}

func (sit *TSituation) Decode(code uint16) {
 for pos := range sit.Cell {
  sit.Cell[pos] = uint8((code / power3[pos]) % 3)
 }
}
```

By encoding the state of the board in this way, we can efficiently store and manipulate game states, allowing for fast traversal of the move sequences and evaluation of potential moves. Figure 1 shows an example of the game situation in the encoding scheme.

O 0	X 1	X 2
X 3	O 4	O 5
6	O 7	8

Figure 1: An example of the game situation in the encoding scheme

The situation code for this particular game state would be 3 0 · 2 + 31 · 1 + 32 · 1 + 33 · 1 + 34 · 2 + 3 5 · 2 + 36 · 0 + 37 · 2 + 38 · 0 = 5063.

Modeling. By considering the aforementioned additional freedom of choice in W3T and due to small size of the game board, we can trivially calculate the number of possible states of the game, thus defining the state space to account for all possible combinations of marks that players can place on the board. This expanded state space allows us to model the game as a finite, deterministic, two-player, zero-sum game, in which players take turns to place their marks on the board and aim to achieve a winning position while preventing their opponent from doing the same.

The total number of mathematically possible situations of the 3-by-3 grid is 3 9 = 19683. However, not all of mathematically possible situations are valid in the sense of being achievable by legal moves. For example, the very last situation that goes by the number 19682 decodes to a field filled with all noughts. This is not achievable situation because the victory condition would have been gained long before the last move, which precludes further moves.

Our W3T model has reference tracking whereby it marks the situations that were achieved while building the complete move sequences. The total number of such situations are 18752, which discards 4.7% of the mathematically possible situations. The number of situations that are not achievable with legal moves is thus 931.

We can also trivially calculate an upper contraint on the number of possible moves in a game of W3T. As the game inevitably concludes at 9th or earlier move, the maximum mathematically possible number of move sequences in W3T is 9! = 362880.

In reality, however, a large portion of these moves lead to the same situations mulitple times or are symmetrical and equivalent to one another. By considering the symmetry of the board, the equivalence of certain moves, and by eliminating the duplicating moves, we can reduce the number of unique move sequences to 80150, which is 77% reduction.

We have set out to construct a complete game graph of valid moves and situations for W3T, enumerating all possible moves. This graph can be used to analyze the game's strategic properties, evaluate optimal play, and determine the outcome of any given position. By traversing the graph and applying heuristics to evaluate the desirability of each move, we can identify the best possible move for a given player and predict the outcome of the game under optimal play.

Building the move graph. Before we continue with building the actual complete move graph, an important aspect should be considered. What is the best data structure to represent the complete set of situations and moves between them? The common practice[23] seems to be tree structure, usually

ISSN 2306-5478

general tree (binary trees are not applicable since in W3T there are usually more than two possible moves at each turn).

However, we believe that certain properties of general trees are not suitable to model W3T efficiently.

One such property is that there are the situations which can occur witin entirely different move sequences. For example, the situation in Figure 1 can be reached by placing nought at 7th cell in a situation #689 or by placing cross at 3rd cell in a situation #5036. In total, the situation #5063 can be reached by seven different move sequences, as there are seven marks already placed. Ideally, such kind of situations must be represented by a single node in the tree, making it a DAG rather than a tree. This optimization also helps reduce memory footprint of the model and improve performance of the building and searching algorithms.

Another property of the trees is that they are not indexed by the situation code. This means that the search for a particular situation in the tree to link to it instead of creating new node requires a linear search through all nodes for each node which is suspected to be non-unqiue. This is not only inefficient, as the number of nodes in the tree grows exponentially with the number of moves, but also heavily complicates the implementation of the search algorithm even if the most efficient indexing methods like hash tables are used.

Additionally, the general trees are usually used in a scenarios where the recursion depth is not reasonably limited or the search across the tree is not exhaustive but is directed by some other kind of heuristic. In W3T, the recursion depth is limited by the number of cells in the playfield, and the search is exhaustive, so the tree structure is not the best choice.

To simplify the implementation, reduce memory footprint and performance impact we propose a specially adapted DAG (we call it Atlas) as a structure to model W3T. The Atlas is an array indexed by the situation code, where each array item contains codes of another situations that result from the current one by applying every valid move for the current situation. The Atlas is built by recursively generating all possible moves for a given situation starting from empty field, while keeping track of the situations being referenced, and evaluating the outcome of each move.

```
type (
 TMove struct {
  situation uint16
  outcome   uint8
 }

 TAtlasItem struct {
  nextMove   [uint8][2]*TMove
  referenced bool
 }
)
```

The Atlas is then used to determine the best move for a given situation, by selecting the move that leads to the most favorable outcome. Figure 2 shows a fragment of an Atlas in the form of a graph for the game situation in Figure 1.

As seen in figure 2, the situation has one instant-win move (placing nought at 8th cell) and three intermediate moves resulting in different situations with four instant end moves—two losses and two draws.

The Atlas graph is built with a recursive function analogous to the recursive functions used to build trees. The function maintains current recursion depth (zero for the situation given at the beginning of calculation), because whether a particular move is winning or losing depends on whether it is "our" move or opponent's one, hence the depth is used to determine the player and the winning or losing outcome is checked simply for victory (non-draw) condition.

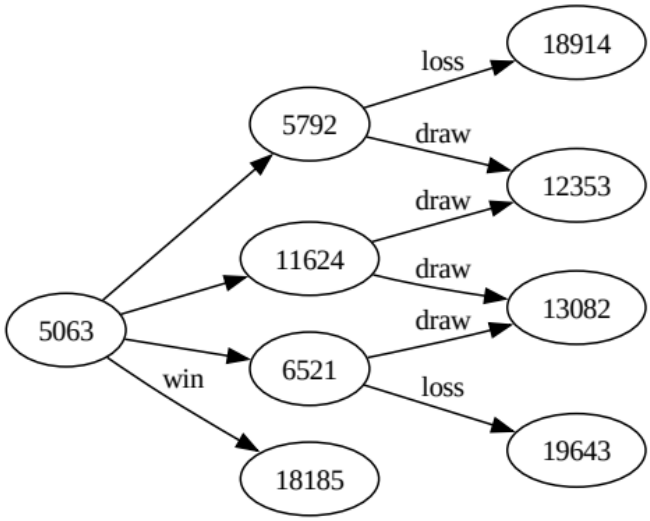

Figure 2: A move tree for the game situation in Figure 1

At the function's entry, the situation is decoded into a playfield and checked for two possible conditions: victory or draw. If the situation is terminal, only the outcome of the move is calculated and returned to the upper level of recursion. If the situation is not terminal, the function iterates over all possible moves, and for each move, it calls itself recursively with the depth increased by one.

The resulting graph, as was mentioned before, is a DAG whose edges are unweighted. However, to implement optimal play strategy, the graph edges should be weighted and an evaluation function should be designed. How it was done is discussed later.

 ISSN 2306-5478

Principal solvability. This section formalizes the logical conditions needed to claim that W3T is "solved" in this sense and explains how these conditions can be verified by traversing the complete move graph.

We define solvability of the game like it was defined in [1] as the ability to generate such chain of moves that is guaranteed to result in an at least a draw regardless of the moves of the opponent. In other words, a game is solvable if there exists a strategy of optimal play that guarantees at least a draw for the player who plays first.

A deterministic, two-player, zero-sum game of perfect information is said to be solved if one can definitively classify the outcome of the initial position (win, loss, or draw) when both players play optimally. In the specific case of W3T, solvability entails demonstrating that the first player can force a result no worse than a draw, regardless of how well the second player responds.

Having constructed the exhaustive game tree, we can now formulate the criteria it must meet to prove the solvability of W3T. To that end, we differentiate between the following kind of nodes:

- **terminal** node whose game situation has no empty cells; as such, it is either winning or drawing situation;
- **pre-terminal** node whose game situation has a single empty cell, therefore all its children nodes are terminal nodes.

As mentioned earlier, the recursion depth is a strategic factor, because it determines whether the same move resulting in a victory situation means win or loss for the player who moved first. As the starting position has zero depth, the player who moved first is generating game situations at odd depths and makes his moves at even depths. Conversely, the player who moved second is generating game situations at even depths and makes his moves at odd depths. Therefore, the winning move for the player who moved first is the move that results in a victory situation at even depth, and the winning move for the player who moved second is the move that results in a victory situation at odd depth.

Retrograde Analysis and Label Assignments. The most direct method to certify solvability is through a retrograde analysis, which systematically labels each position in the move graph (or game tree/DAG) as one of the following from the perspective of the player to move:

- Winning (W): There is at least one move leading to a position that is Losing for the opponent.
- Losing (L): Every possible move leads only to positions that are Winning for the opponent.
- Drawing (D): The position is neither Winning nor Losing, but there is at least one move that forces the game into a Drawing position or otherwise avoids a forced loss.

These labels are determined starting from terminal positions — where the game ends immediately — and proceeding backward through each predecessor state until the entire game graph is labeled.

Terminal positions:

- If the board state already has three in a row for the player to move, it is immediately Winning (W) for that player (though practically, such a terminal state would usually be assigned at the previous ply if the win was just formed).
- If the board state has three in a row for the other player, it is Losing (L) for the current player.
- If the board is completely filled and neither side has three in a row, it is Drawing (D).

Non-terminal positions:

- A position is labeled Winning (W) if there exists at least one legal move to a state labeled Losing (L) for the opponent.
- A position is labeled Losing (L) if all legal moves lead to positions labeled Winning (W) for the opponent.
- A position is labeled Drawing (D) if it is neither Winning nor Losing, but at least one legal move leads to a position labeled Drawing, or otherwise guarantees the current player can avoid an eventual forced loss.

Because W3T is finite, this backward labeling is well-defined: starting from all terminal states (bottom layer) and iterating upward, one obtains a unique label (W, L, or D) for each position. Once labeling is complete, the classification of the initial position (the empty board in the case of W3T) definitively determines whether the first player can force a win, can only force a draw, or is destined to lose against perfect play.

Conditions for "At Least a Draw". To claim that W3T is solvable with an outcome of at least a draw for the first player, two requirements must be satisfied:

- The move graph is fully explored: All achievable board states (those arising from legal sequences of moves) appear as nodes in the graph, and edges correctly reflect every possible move from each state. This ensures no forced winning line or drawing sequence is overlooked.
- The initial position's label is either W or D:
  – If the empty board is labeled Winning (W) for the first player, then there exists at least one forced sequence of moves leading to a guaranteed win, regardless of the opponent's responses.
  – If the empty board is labeled Drawing (D) for the first player, then there is no forced win, but there is a strategy ensuring no worse than a draw.

 ISSN 2306-5478

Once these conditions are verified, one concludes that an optimal strategy exists that prevents the opponent from forcing a loss. By contrast, if the empty board is labeled Losing (L), it implies that, under perfect play by both sides, the second player can force a win.

Practical Procedure. Although the retrograde analysis framework is conceptually straightforward, implementing it efficiently for W3T requires careful handling of the following:

1. Directed Acyclic Graph (DAG) Representation: Because multiple distinct move sequences can arrive at the same board configuration, storing each state once in a DAG (rather than a tree) prevents duplication and reduces memory requirements. The Atlas structure is well-suited for this because every situation is indexed by its encoded integer, and no new node is created if that situation was previously encountered.

2. Depth and Turn Tracking: Since outcome labels are always from the viewpoint of the player to move, it is essential to maintain which player's move is next. Commonly, one can track this by incrementing a "depth" so that one knows whether it is the first or second player's turn. However, once positions are labeled W, L, or D, one no longer needs to store the depth permanently; it is reflected by how one propagate labels upward from the terminal states.

3. Terminal Checks:
   - If a board has a three-in-a-row, label the resulting position accordingly (Winning for the side that completed three in a row; Losing for the other side if that row belongs to the opponent).
   - If the board is fully occupied without a three-in-a-row, label it Drawing.

4. Forward and Backward Sweeps:
   - Forward construction populates the Atlas by enumerating all legal next moves from any situation.
   - Backward labeling visits each situation in reverse (post-order traversal in a sense), assigning it a W, L, or D classification based on its children's labels.

In practice, it seems prudent to intertwine these passes. For instance, a recursive function that builds the full DAG of situations can detect terminal nodes (win or draw) immediately and assign their labels. Then on the way "back up" it can assign the W, L, or D label to each predecessor node. Alternatively, a two-pass algorithm—one pass to collect and store the DAG; a second pass to label nodes in reverse topological order—achieves the same logical result.

Ensuring a Strategy of Optimal Play. Once every position in the DAG is labeled, verifying that the first player can force at least a draw consists of these simple checks:

- Check the label of the empty-board state: If it is W or D, the game is "solved" in favor of the first player not losing.
- Confirm no labeling contradictions exist: For instance, it must never occur that a position is simultaneously assigned more than one label or that a Winning position does not actually have a move leading to a Losing position for the opponent. A straightforward consistency check is part of the retrograde analysis, because each label is derived solely from its children's final labels.

Illustrative Example. Suppose we have a partial Atlas for a mid-game position in W3T. If at least one outgoing edge from that position leads to a Losing position for the opponent, we mark it Winning for the current player. Conversely, if all outgoing edges lead to Winning positions for the opponent, we mark it Losing. Should none of these transitions guarantee a forced loss for the opponent, but at least one leads to a position that is eventually a Draw, we classify it as Drawing.

By systematically applying these rules throughout the entire DAG, we arrive at a single, consistent label for each situation. The label for the initial position conclusively tells us whether W3T is solvable—namely, if the label is Win or Draw, the first player has a strategy that avoids losing.

Suppose we have built part of an Atlas with position that has two empty cells. A player therefore have four possible moves — two possible marks in two empty cells. The Atlas indicates that three out of four of these possible moves are an instant win. The fourth one, however, leads to a definite loss as any possible move in the remaining empty cell results in a win for the opponent. The following is a simple, step-by-step explanation of how the retrograde labeling (or "back-propagation" of outcomes) prevents a player from being forced into a losing move, even if one of the possible moves is losing.

As a first step, we ought to identify the possible moves and their outcomes.

Since there are two empty cells, and each move can be either a cross or a nought, that yields four distinct moves. Call these moves M1, M2, M3, and M4 for convenience. Then we need to check the Atlas (or recursively recompute it for the current situation) for each child's Label. We look up the child position after each move (M1 to Child1, M2 to Child2, etc.) Each child is either:

- An immediate win (the newly placed mark completes three in a row)
- A losing position (the next move will hand victory to the opponent)
- A draw position, or something else.

 ISSN 2306-5478

In our simplified example three of those moves lead to an instant win (so from the next player's viewpoint, those children are losing). The fourth move leads to a position from which the opponent can force a win (so from the current player's viewpoint, that child is Losing). Hence in this case the children's labels might look like:

- Child1: L (because from the next player's perspective, it's a losing position—in other words, we just force them into losing)
- Child2: L
- Child3: L
- Child4: W (since if we make that move, we end up giving our opponent a winning position to play)

Note that "Child is labeled L" means "the child position is losing for the opponent." Conversely, "Child is labeled W" means "the child position is winning for the opponent," which is losing for the player.

Next we should apply the retrograde labeling rule to the parent position. From the perspective of the current player at the parent position, we now evaluate this: do any of the children have label L (meaning if a player moves there, it's losing for the opponent)? If yes then this parent position is labeled W. If no child is L but at least one is D (draw) then label the parent D. Otherwise (if all children are W from the player's perspective) then label the parent L. According to the example scenario, we do indeed have three children labeled L (Child1, Child2, Child3). That means we can choose one of those winning moves. Therefore, the parent position is W for us, the current player.

Because the parent is labeled W, it implies we are not forced to pick the losing move (the one that would lead to Child4). Thus optimal play means the player chooses one of the moves that leads to an L-labeled child (for the opponent).

Even though there is a losing fourth option in the move set, the existence of any winning move is enough for the parent position to be classified as "winning" for the current player. Once it's labeled W, that labeling is also "back-propagated" to any predecessors. If a predecessor node can transition to this parent, it may become a winning or drawing state for that player, and so on, all the way back to the initial position.

An important thing to understand is why this logic prevents forced losses. A core principle in combinatorial game theory is that a player with a winning position can always steer the game into one of its losing children (for the opponent). This is precisely how the tree/DAG labeling ensures the losing move (Child4) is never "forced" upon the player. Because we only pick moves that produce the best outcome (Winning or Drawing), we must ignore any losing children once we discover there is at least one better alternative.

If we imagine a large, complex portion of the Atlas, the same rule generalizes: as soon as we find any child that is losing for the opponent (i.e.,

labeled L from their perspective), our current position is winning for player. Players do not need to worry if other children lead to a forced loss; they simply skip those.

The procedure of upward label propagation is repeated upwards through the DAG—every parent learns its label from its children's labels—until all states, including the initial board state, are assigned a label. The presence of even a single winning move at a position is enough to mark that position as winning, which conveys the idea that the player at that position can (and will) avoid any losing alternatives.

Final Remarks on Solvability. Using the above approach, "principal solvability" in W3T emerges from a complete, consistent labeling of every position in the move graph. If this labeling shows that the initial position is either Win or Draw for the first player, then W3T is demonstrably solvable in the sense that there is an optimal strategy guaranteeing at least a draw.

From a game-theoretic standpoint, these conditions are both necessary and sufficient: if any position can lead inevitably to an opponent's forced win, the labeling will uncover it, and the first player's outcome classification would then be Lose. Conversely, if the empty-board position is labeled Win or Draw, there exists a systematic way to guide each move such that the first player either wins outright or prevents loss, thus establishing the solvability of W3T.

Results. The implementation of the Atlas was done using Golang with the following primitives:

```
const (
        moUndefined = iota
        moDraw
        moWin
        moLoss
)

type (
        TMove struct {
                situation playfield.TSituationCode
                outcome   uint8
        }

        TAtlasItem struct {
                nextMove   [playfield.LenFlatPlayfield][len(playfield.Mark)]*TMove
                referenced bool
        }
)

var (
        Atlas [playfield.CountSituations]TAtlasItem
)
```

As seen in this snippet, for each game situation (uniquely encoded by the situation number using trit-based encoding explaned above) we store a two-dimensional array of possible moves (dimensioned as positions by marks), each of which is a pointer to the next game situation. The outcome of the move (undefined, draw, win, or loss) is also stored. The referenced flag is used to mark the game situations that have been visited during the search.

 ISSN 2306-5478

There are three methods implemented to work with Atlas — building it, finding the best move given the game situation and dumping it in human-readable form. The building method was discussed in great detail in the previous subsections. The move finding method is fairly simple, as it just finds the winning move among those registered for the situation. If a forced winning move cannot be found, this method looks for a move that leads to a draw. If no such move is found, the method returns the random move in the list.

The dumping method produces the dump of Atlas in human-readable form like this:

```
code: 9371
  o |   |
 ---+---+---
  o | x | o
 ---+---+---
    | x | x
 * move : x at 1 => sitcode: 9374 win
 * move : o at 1 => sitcode: 9377 undef
 * move : x at 2 => sitcode: 9380 undef
 * move : o at 2 => sitcode: 9389 undef
 * move : x at 6 => sitcode: 10100 win
 * move : o at 6 => sitcode: 10829 win
```

This dump shows the game situation encoded as 9371, the playfield, and the possible moves with their outcomes. The moves are shown as the mark and the position where it is placed. The situation codes are shown for each move, and the outcomes are shown as well.

It is using this dump the final question of the work can be answered by looking at all the propagated outcomes that lead to the root situation. The root situation is the one where the game starts, and the outcomes are propagated from the leaf situations to the root situation. The Atlas entry for the root situation is this

```
code: 0
   |   |
---+---+---
   |   |
---+---+---
   |   |
 * move : x at 0 => sitcode: 1 win
 * move : o at 0 => sitcode: 2 win
 * move : x at 1 => sitcode: 3 win
 * move : o at 1 => sitcode: 6 win
 * move : x at 2 => sitcode: 9 win
 * move : o at 2 => sitcode: 18 win
 * move : x at 3 => sitcode: 27 win
 * move : o at 3 => sitcode: 54 win
 * move : x at 4 => sitcode: 81 win
 * move : o at 4 => sitcode: 162 win
 * move : x at 5 => sitcode: 243 win
 * move : o at 5 => sitcode: 486 win
 * move : x at 6 => sitcode: 729 win
 * move : o at 6 => sitcode: 1458 win
 * move : x at 7 => sitcode: 2187 win
 * move : o at 7 => sitcode: 4374 win
 * move : x at 8 => sitcode: 6561 win
 * move : o at 8 => sitcode: 13122 win
```

Note that we completely understand that there are only three truly unique starting moves for an empty 3-by-3 playfield, and the rest are just

rotations and reflections of the same situation. We, however, decided not to filter those reflections and rotations out for simplicity of human understanding of the Atlas dump.

Conclusions. As shown in the Atlas dump for the root situation above, all and every possible starting moves have "winning" flavor, which means that the game is at least a first-player win. This is a very interesting result, as it shows that the game is not only solvable but also has a forced winning strategy for the first player.

For completeness, we also provide a quantitative profile of the search space and its reduction under our modeling choices (Table 1). The summary distinguishes the theoretical state space (3^9), the subset of legally reachable positions discovered during Atlas construction, and the combinatorial upper bound of (9!) move sequences. It then reports the effect of transposition handling and canonicalization under the dihedral symmetries of the (3 × 3) board, yielding the deduplicated count of unique sequences and the corresponding reduction ratio. Finally, it clarifies that the Atlas stores each canonicalized situation once, so the number of Atlas nodes coincides with the number of reachable states—highlighting the practical benefit of a DAG representation over a tree for exhaustive analysis.

Table 1

Quantitative summary of W3T state space and search (Atlas).

Metric	Value
Mathematically possible board states 3^9	19,683
Reachable legal states (visited in Atlas)	18,752
Unreachable (illegal) states	931 (4.73% of all states)
Max. possible move sequences (upper bound)	$9! = 362{,}880$
Unique move sequences (after dedup. & symmetry)	80,150
Reduction vs. max sequences	77.9% (unique = 22.1% of max)
First-move options that force a win	18 / 18 (3 by symmetry)
Atlas nodes used (equals reachable states)	18,752

Notes: Percentages are computed relative to the totals shown. Symmetry refers to dihedral symmetries of the

The resulting model functions as an interpretable artificial intelligence system: every decision path, evaluation rule, and outcome label is explicit and verifiable, providing a contrast to opaque statistical methods. This interpretability makes the Atlas framework applicable not only to abstract games but also to deterministic decision processes within IoE and cyber-physical domains where provable correctness is paramount.

Future work. It would also be very interesting to explore the game from the standpoint of the opposing player (who moves second), as the forced winning strategy for the first player is already known. This would involve traversing the Atlas at a modified depth while also reusing the existing code so that the outcome labels (draws versus victories) are either inverted or abstracted from the depth of traversal.

 ISSN 2306-5478

Scalability and Generalization to AI Decision Systems. The Atlas formalism—canonical indexing of positions with successors stored once per state—extends beyond W3T, but practicality depends on branching factor and state-space growth. For a board with (n) cells, the out-degree of a nonterminal node is bounded by ($2e$), where (e) is the number of empty cells (two marks per empty cell). Thus, worst-case local branching grows linearly with remaining empties, while the global number of positions ($|S|$) grows combinatorially with (n). The memory footprint of a full Atlas is ($O(|S| + |E|)$), with ($|E|$) proportional to the sum of out-degrees across all reachable states; in practice ($|E| \approx \Theta(\bar{d}|S|)$), where ($\bar{d}$) is the average out-degree. Canonicalization under the relevant symmetry group (e.g., the dihedral group ($D4$) for square grids) reduces ($|S|$) and ($|E|$) by a factor approaching the group order for sufficiently generic positions, and transposition handling prevents duplication from multiple arrival sequences. Nevertheless, for larger boards or variants with richer move alphabets, ($|S|$) and ($|E|$) may exceed main memory even with aggressive packing.

Two pragmatic directions follow. Engineering improvements: bit-packed encodings, minimal perfect hashing for state indices, external-memory (blocked) Atlas layouts, and reverse topological sweeps for retrograde labeling with frontier compression can extend exact solving to substantially larger instances. Proof-oriented search: hybrid pipelines that use statistical guidance (e.g., best-first or MonteCarlo policies) to prioritize subspaces, and then emit proof certificates (retrograde labels plus cut justifications) only where needed, can deliver correctness without enumerating the entire graph. These strategies preserve the certificate-based nature of Atlas while mitigating the blow-up associated with high branching factors.

These strategies preserve the certificate-based nature of Atlas while mitigating the blow-up associated with high branching factors, and could be adapted as verifiable reasoning modules for deterministic or semi-deterministic agents in AI-driven IoE ecosystems. Declaration on Generative AI During the preparation of this work, the author used GPT-4 series of LLMs in order to fix grammar and check spelling. After using thise service, the author reviewed and edited the content as needed and takes full responsibility for the publication's content.

1. S. Garg, D. Songara, S. Maheshwari, The winning strategy of tic tac toe game model by using theoretical computer science, in: 2017 International Conference on Computer, Communications and Electronics (Comptelix), IEEE, Jaipur, India, 2017, pp. 522–526. doi:10.1109/COMPTELIX. 2017.8004027, accessed: February 19, 2025. **2.** S. W. Golomb, Tic-tac-toe, Mathematical Games and Recreations (1975). URL: https://www. researchgate.net/publication/343148543_Tic-Tac-Toe, available on

ResearchGate; originally published in mathematical literature. Accessed: February 19, 2025. **3.** A. Bhatt, P. Varshney, K. Deb, In search of no-loss strategies for the game of tic-tac-toe using a customized genetic algorithm, in: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO '08), ACM, Atlanta, GA, USA, 2008, pp. 889–890. doi:10.1145/1389095.1389270, accessed: February 19, 2025. **4.** J. Schaeffer, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu, S. Sutphen, Checkers is solved, Science 317 (2007) 1518–1522. doi:10.1126/science.1144079. **5.** R. Gasser, Solving Nine Men's Morris, Ph.D. thesis, ETH Zürich, 1996. **6.** G. E. Gévay, G. Danner, Calculating ultra-strong and extended solutions for nine men's morris, morabaraba, and lasker morris, arXiv preprint arXiv:1408.0032 (2014). URL: https://arxiv.org/abs/ 1408.0032. **7.** L. V. Allis, A Knowledge-Based Approach of Connect-Four, Master's thesis, Vrije Universiteit Amsterdam, 1988. **8.** L. V. Allis, M. van der Meulen, H. J. van den Herik, Proof-number search, Artificial Intelligence 66 (1994) 91–124. doi:10.1016/0004-3702(94)90008-6. **9.** E. V. Nalimov, G. M. Haworth, E. A. Heinz, Space-efficient indexing of chess endgame tables, ICGA Journal 23 (2000) 151–162. **10.** R. de Man, Syzygy endgame tablebases (wdl/dtz), 2013. URL: https://syzygy-tables.info/, online resource. **11.** D. E. Knuth, R. W. Moore, An analysis of alpha-beta pruning, Artificial Intelligence 6 (1975) 293–326. doi:10.1016/0004-3702(75)90019-3. **12.** G. C. Stockman, A minimax algorithm better than alpha-beta?, Artificial Intelligence 12 (1979) 179–196. doi:10.1016/0004-3702(79)90011-0. **13.** A. Plaat, J. Schaeffer, W. Pijls, A. de Bruin, Best-first fixed-depth game-tree search in practice, Artificial Intelligence 87 (1996) 255–293. doi:10.1016/S0004-3702(96)00046-2. **14.** A. Plaat, MTD(f): A Minimax Algorithm Faster Than NegaScout, Technical Report IR-422, Leiden University, 1996. URL: https://pubs.liacs.nl/pdf/422.pdf. **15.** A. L. Zobrist, A New Hashing Method with Application for Game Playing, Technical Report Computer Sciences Technical Report 88, University of Wisconsin, 1970. **16**. J. W. Romein, H. E. Bal, Solving awari using parallel retrograde analysis, Computer 35 (2002) 26–33. doi:10.1109/MC.2002.1039515. **17.** S. Schiffel, Symmetry detection in general game playing, in: Proceedings of the AAAI Conference (Workshop on General Game Playing), 2010. **18**. M. Genesereth, N. Love, B. Pell, General game playing: Overview of the AAAI competition, AI Magazine 26 (2005) 62–72. doi:10.1609/aimag.v26i2.1810. **19.** T. E. Plambeck, G. Whitehead, The secrets of notakto: Winning at x-only tic-tac-toe, arXiv preprint arXiv:1301.1672 (2013). URL: https://arxiv.org/abs/1301.1672. **20.** Y. Elran, Wild tic-tac-toe, 2024. URL: https://www.futurelearn.com/courses/ introduction-to-recreational-math, course material, Davidson Institute / Weizmann Institute. **21.** D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, D. Hassabis, Mastering the game of go without human

 ISSN 2306-5478

knowledge, Nature 550 (2017) 354–359. doi:10.1038/ nature24270. **22.** D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, D. Hassabis, Mastering chess and shogi by self-play with a general reinforcement learning algorithm, arXiv preprint arXiv:1712.01815 (2017). URL: https: //arxiv.org/abs/1712.01815. **23.** L. Attia, L. Lichev, D. Mitsche, R. Saona, B. Ziliotto, Zero-sum random games on directed graphs, arXiv preprint arXiv:2401.16252 (2024).

REFERENCES

1. S. Garg, D. Songara, S. Maheshwari, The winning strategy of tic tac toe game model by using theoretical computer science, in: 2017 International Conference on Computer, Communications and Electronics (Comptelix), IEEE, Jaipur, India, 2017, pp. 522–526. doi:10.1109/COMPTELIX. 2017.8004027, accessed: February 19, 2025. **2.** S. W. Golomb, Tic-tac-toe, Mathematical Games and Recreations (1975). URL: https://www. researchgate.net/publication/343148543_Tic-Tac-Toe, available on ResearchGate; originally published in mathematical literature. Accessed: February 19, 2025. **3.** A. Bhatt, P. Varshney, K. Deb, In search of no-loss strategies for the game of tic-tac-toe using a customized genetic algorithm, in: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO '08), ACM, Atlanta, GA, USA, 2008, pp. 889–890. doi:10. 1145/1389095.1389270, accessed: February 19, 2025. **4.** J. Schaeffer, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu, S. Sutphen, Checkers is solved, Science 317 (2007) 1518–1522. doi:10.1126/science.1144079. **5.** R. Gasser, Solving Nine Men's Morris, Ph.D. thesis, ETH Zürich, 1996. **6.** G. E. Gévay, G. Danner, Calculating ultra-strong and extended solutions for nine men's morris, morabaraba, and lasker morris, arXiv preprint arXiv:1408.0032 (2014). URL: https://arxiv.org/abs/ 1408.0032. **7.** L. V. Allis, A Knowledge-Based Approach of Connect-Four, Master's thesis, Vrije Universiteit Amsterdam, 1988. **8.** L. V. Allis, M. van der Meulen, H. J. van den Herik, Proof-number search, Artificial Intelligence 66 (1994) 91–124. doi:10.1016/0004-3702(94)90008-6. **9.** E. V. Nalimov, G. M. Haworth, E. A. Heinz, Space-efficient indexing of chess endgame tables, ICGA Journal 23 (2000) 151–162. **10.** R. de Man, Syzygy endgame tablebases (wdl/dtz), 2013. URL: https://syzygy-tables.info/, online resource. **11.** D. E. Knuth, R. W. Moore, An analysis of alpha-beta pruning, Artificial Intelligence 6 (1975) 293–326. doi:10.1016/0004-3702(75)90019-3. **12.** G. C. Stockman, A minimax algorithm better than alpha-beta?, Artificial Intelligence 12 (1979) 179–196. doi:10.1016/0004-3702(79)90011-0. **13.** A. Plaat, J. Schaeffer, W. Pijls, A. de Bruin, Best-first fixed-depth game-tree search in practice, Artificial Intelligence 87 (1996) 255–

293. doi:10.1016/S0004-3702(96)00046-2. **14.** A. Plaat, MTD(f): A Minimax Algorithm Faster Than NegaScout, Technical Report IR-422, Leiden University, 1996. URL: https://pubs.liacs.nl/pdf/422.pdf. **15.** A. L. Zobrist, A New Hashing Method with Application for Game Playing, Technical Report Computer Sciences Technical Report 88, University of Wisconsin, 1970. **16**. J. W. Romein, H. E. Bal, Solving awari using parallel retrograde analysis, Computer 35 (2002) 26–33. doi:10.1109/MC.2002.1039515. **17.** S. Schiffel, Symmetry detection in general game playing, in: Proceedings of the AAAI Conference (Workshop on General Game Playing), 2010. **18**. M. Genesereth, N. Love, B. Pell, General game playing: Overview of the AAAI competition, AI Magazine 26 (2005) 62–72. doi:10.1609/aimag.v26i2.1810. **19.** T. E. Plambeck, G. Whitehead, The secrets of notakto: Winning at x-only tic-tac-toe, arXiv preprint arXiv:1301.1672 (2013). URL: https://arxiv.org/abs/1301.1672. **20.** Y. Elran, Wild tic-tac-toe, 2024. URL: https://www.futurelearn.com/courses/ introduction-to-recreational-math, course material, Davidson Institute / Weizmann Institute. **21.** D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, D. Hassabis, Mastering the game of go without human knowledge, Nature 550 (2017) 354–359. doi:10.1038/ nature24270. **22.** D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, D. Hassabis, Mastering chess and shogi by self-play with a general reinforcement learning algorithm, arXiv preprint arXiv:1712.01815 (2017). URL: https: //arxiv.org/abs/1712.01815. **23.** L. Attia, L. Lichev, D. Mitsche, R. Saona, B. Ziliotto, Zero-sum random games on directed graphs, arXiv preprint arXiv:2401.16252 (2024).

Порєв Г.В. [1; ORCID ID: 0000-0003-4825-0917],
Доктор технічних наук, Доцент
Зубик Я.Я. [2; ORCID ID: 0000-0002-0802-3552],
Старший викладач
[1] Київський національнийй університет ім. Т.Г. Шевченка
[2] Національний університет водного господарства та природокористування, Рівне

ВИЧЕРПНИЙ ЕВРИСТИЧНИЙ ПОШУК ДЛЯ РОЗВ'ЯЗАННЯ ДЕТЕРМІНОВАНОЇ СКІНЧЕНОЇ ГРИ З НУЛЬОВОЮ СУМОЮ

Хрестики-нулики (W3T) – це комбінаторне розширення класичної гри в хрестики-нулики, що відрізняється правилом, що гравці можуть розміщувати будь-які позначки на кожному ході. Ця додаткова гнучкість

ISSN 2306-5478

вносить значну стратегічну складність і досі залишала теоретичний результат гри невирішеним. У цьому дослідженні ми ретельно розглядаємо розв'язність W3T, побудувавши повне представлення її простору станів та всіх допустимих послідовностей ходів, використовуючи спеціально розроблену спрямовану ациклічну структуру графа, яка називається Атлас. Скінченна та детермінована природа гри дозволяє вичерпний прохід усіх допустимих позицій, що завершується повним маркуванням станів за допомогою ретроградного аналізу. Ми визначаємо та застосовуємо мітки результатів – виграш, програш або нічия – що поширюються від кінцевих позицій вгору по графу. Наша реалізація використовує потрійну схему кодування для унікального представлення станів дошки, з ефективним доступом до пам'яті та повторним використанням станів, що полегшується цілочисельними масивами. Результати демонструють, що W3T не лише розв'язується в класичному сенсі ігор з нульовою сумою з ідеальною інформацією, але й є вимушеною перемогою для першого гравця за оптимальної гри. Ця робота пропонує нову методологію для розв'язання подібних ігор зі скінченними задачами та підкреслює важливість структурної оптимізації в дослідженні простору станів. Запропонований фреймворк є прикладом форми символічного штучного інтелекту, застосованої до вичерпного міркування в просторі станів, пропонуючи прозору альтернативу статистичним підходам до навчання, які зазвичай використовуються в системах прийняття рішень для Інтернету речей.

***Ключові слова*:** евристика, штучний інтелект, теорія ігор, міркування в просторі станів, дикі хрестики-нулики, вичерпний пошук, мінімакс, альфа-бета обрізання

Отримано: 12 січня 2026 року
Прорецензовано: 21 лютого 2026 року
Прийнято до друку: 27 березня 2026 року