1. Introduction and Overview

- **Project Overview:**
  This project is Tic-Tac-Toe game where a human player competes against an AI. The AI uses advanced decision-making techniques to select optimal moves based on game states.
- **Applications of Similar Systems:** google Tic-Tac-Toe game and Tic-TacToe AI game in andriod

---

. Literature Review

1. Minimax algorithm and its applications in decision-making.
2. Alpha-Beta pruning for computational efficiency in tree searches.
3. Heuristic functions and their role in AI.
4. Game symmetry and its reduction techniques.
5. Applications of threading in real-time systems.

Links:

1. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=f7a444a0274b0957284f94a2732842eb9a73d3c7
2. https://core.ac.uk/reader/24065570
3. https://www.researchgate.net/profile/Plamenka-Borovska-2/publication/220795557_Efficiency_of_parallel_minimax_algorithm_for_game_tree_search/links/55c8809408aeca747d66c62b/Efficiency-ofparallelminimaxalgorithm-for-game-tree-search.pdf
4. https://dbcyelagiri.edu.in/dbcy-publication/support/pdf/6.pdf
5. https://www.researchgate.net/profile/Sumit-Shevtekar/publication/366169407_Analysis_of_Game_Tree_Search_Algorithms_Using_Minimax_Algorithm_and_Alpha-Beta_Pruning/links/66067082f5a5de0a9fe88bb2/Analysis-of-Game-Tree-Search-Algorithms-Using-Minimax-Algorithm-and-Alpha-Beta-Pruning.pdf

---

Development Platform:

- Windows.

- **Tools:**

  - Git for version control & vs code.

- **Programming Languages:**

  - Python.

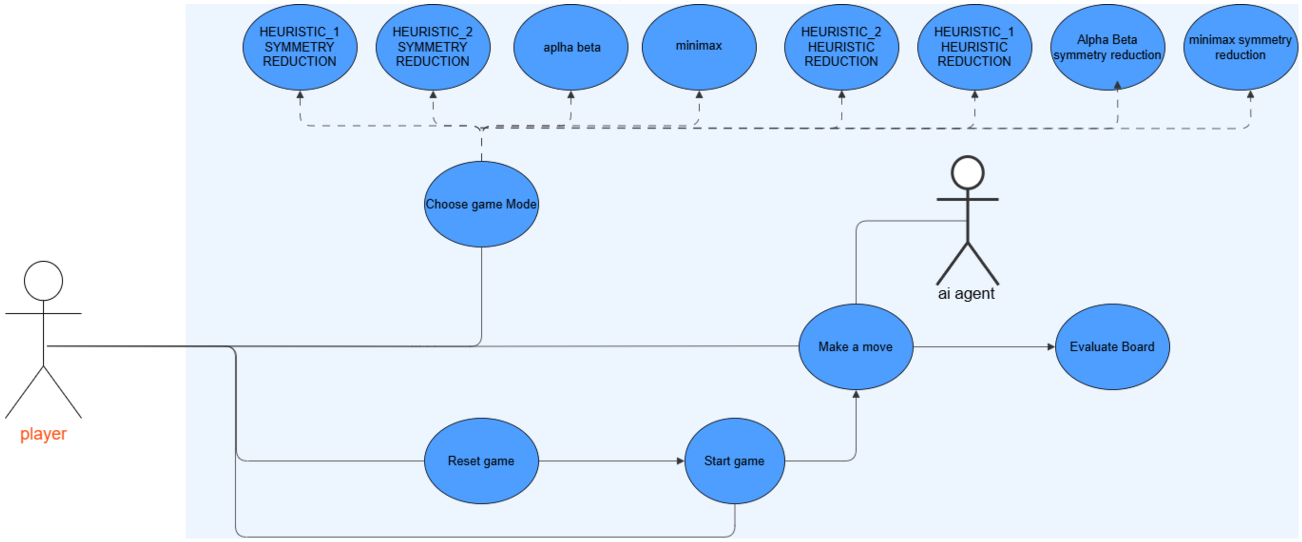
- **Libraries:**

  - Tinker for gui
  - Threading and ThreadPoolExecutor from concurrent.futures for multi threading
  - Math for import infinity
  - Time for plots
  - Enum for enumuration

2. Proposed Solution

Main Functionalities (Use-Case Diagram)

- **User Actions:**
    - Start a new game. ○ Select game difficulty/mode (Minimax, Alpha-Beta, heuristicbased). ○ Make a move on the board.
    - Reset the game.
- **System Actions:**
    - Determine the best AI move using the selected algorithm.

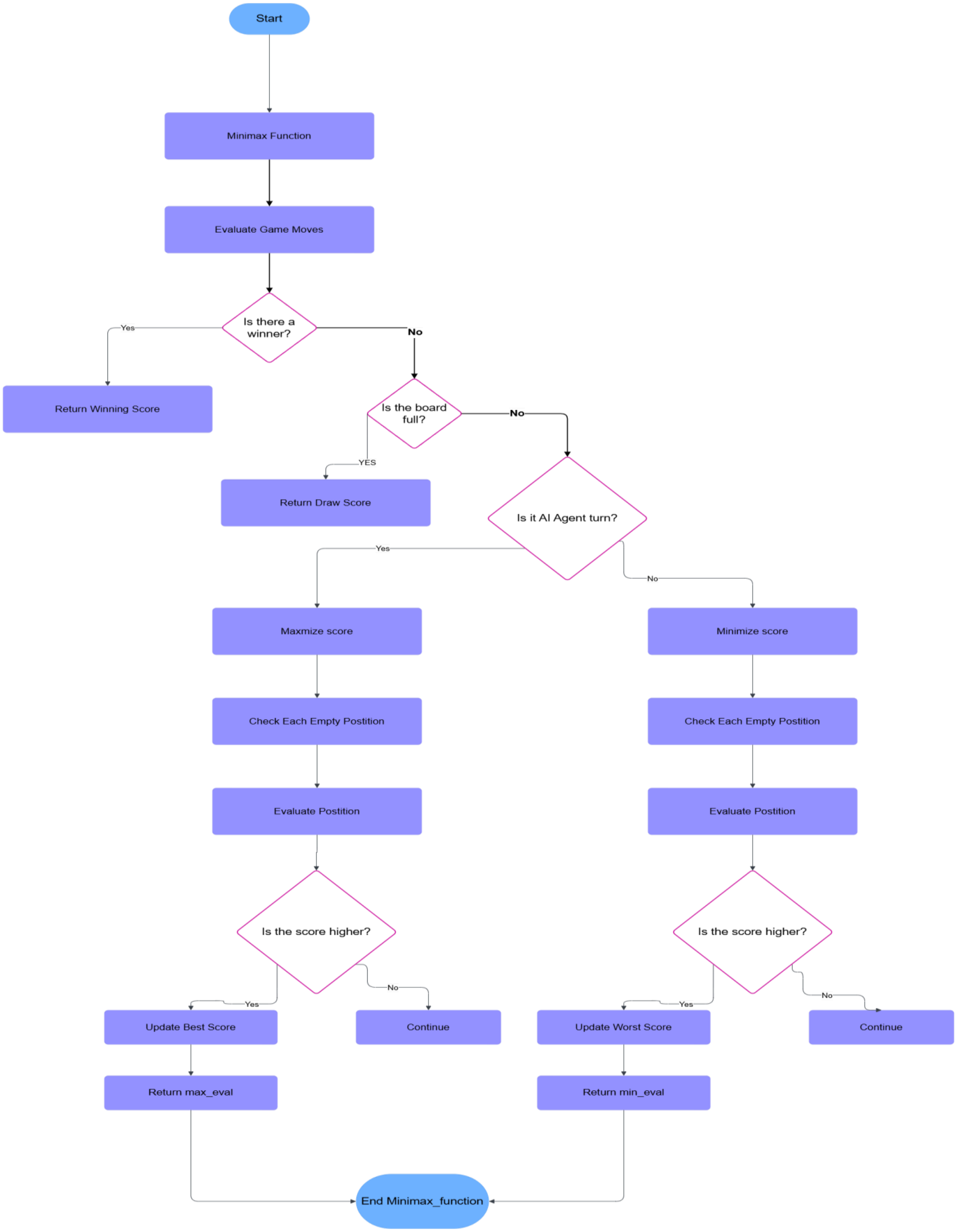

HEURISTIC_1 SYMMETRY REDUCTION
HEURISTIC_2 SYMMETRY REDUCTION
aplha beta
minimax
HEURISTIC_2 HEURISTIC REDUCTION
HEURISTIC_1 HEURISTIC REDUCTION
Alpha Beta symmetry reduction
minimax symmetry reduction
Choose game Mode
ai agent
player
Make a move
Evaluate Board
Reset game
Start game

3. Applied Algorithms:

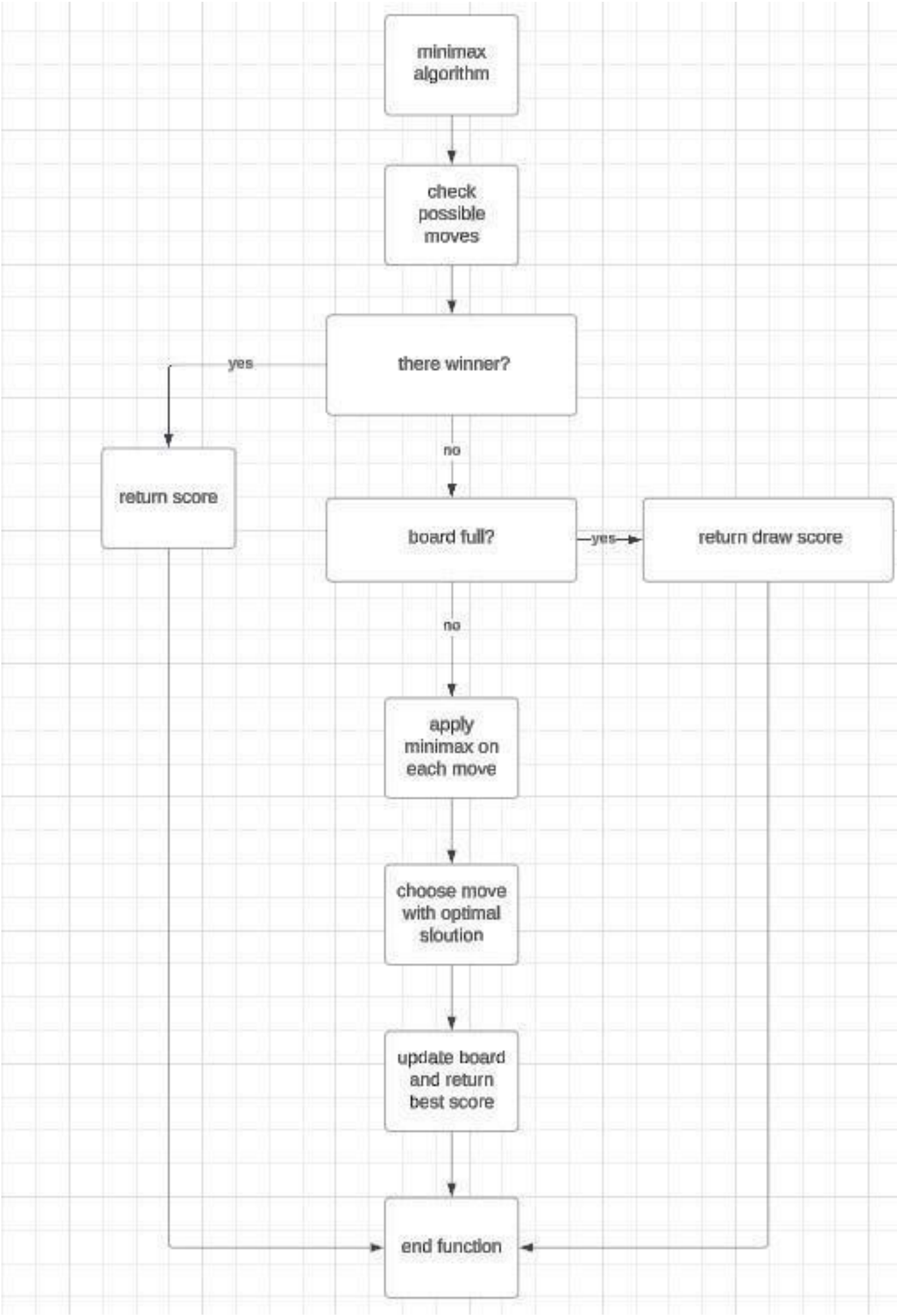
Minimax Algorithm

- **Overview:**
  Searches the game tree to determine the optimal move for the AI by minimizing the opponent's best outcome.

Flow chart:


Start
Minimax Function
Evaluate Game Moves
Is there a winner?
Yes
No
Return Winning Score
Is the board full?
No
YES
Return Draw Score
Is it AI Agent turn?
Yes
No
Maxmize score
Minimize score
Check Each Empty Postition
Check Each Empty Postition
Evaluate Postition
Evaluate Postition
Is the score higher?
Is the score higher?
Yes
No
Yes
No
Update Best Score
Continue
Update Worst Score
Continue
Return max_eval
Return min_eval
End Minimax_function

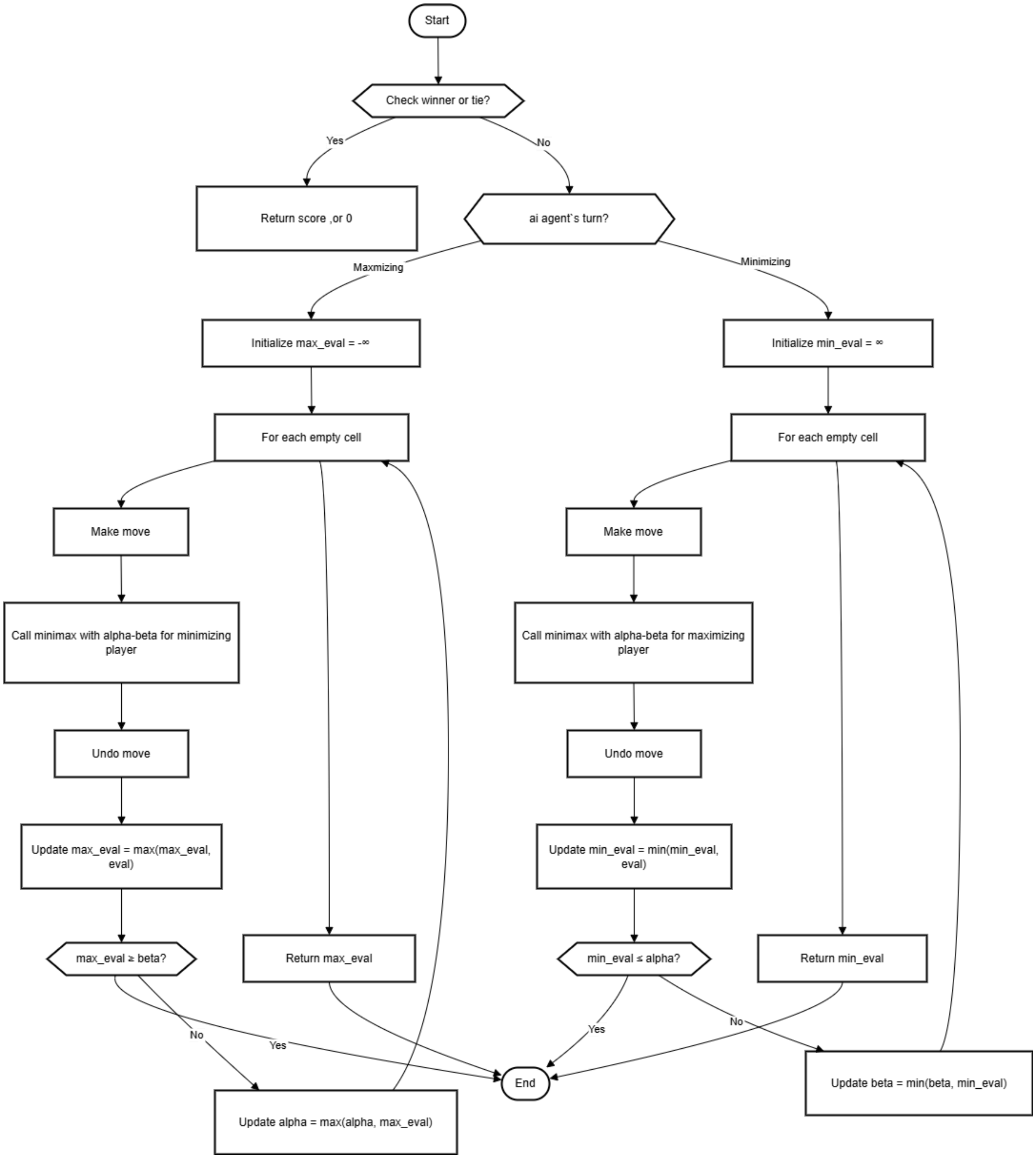
Block diagram:



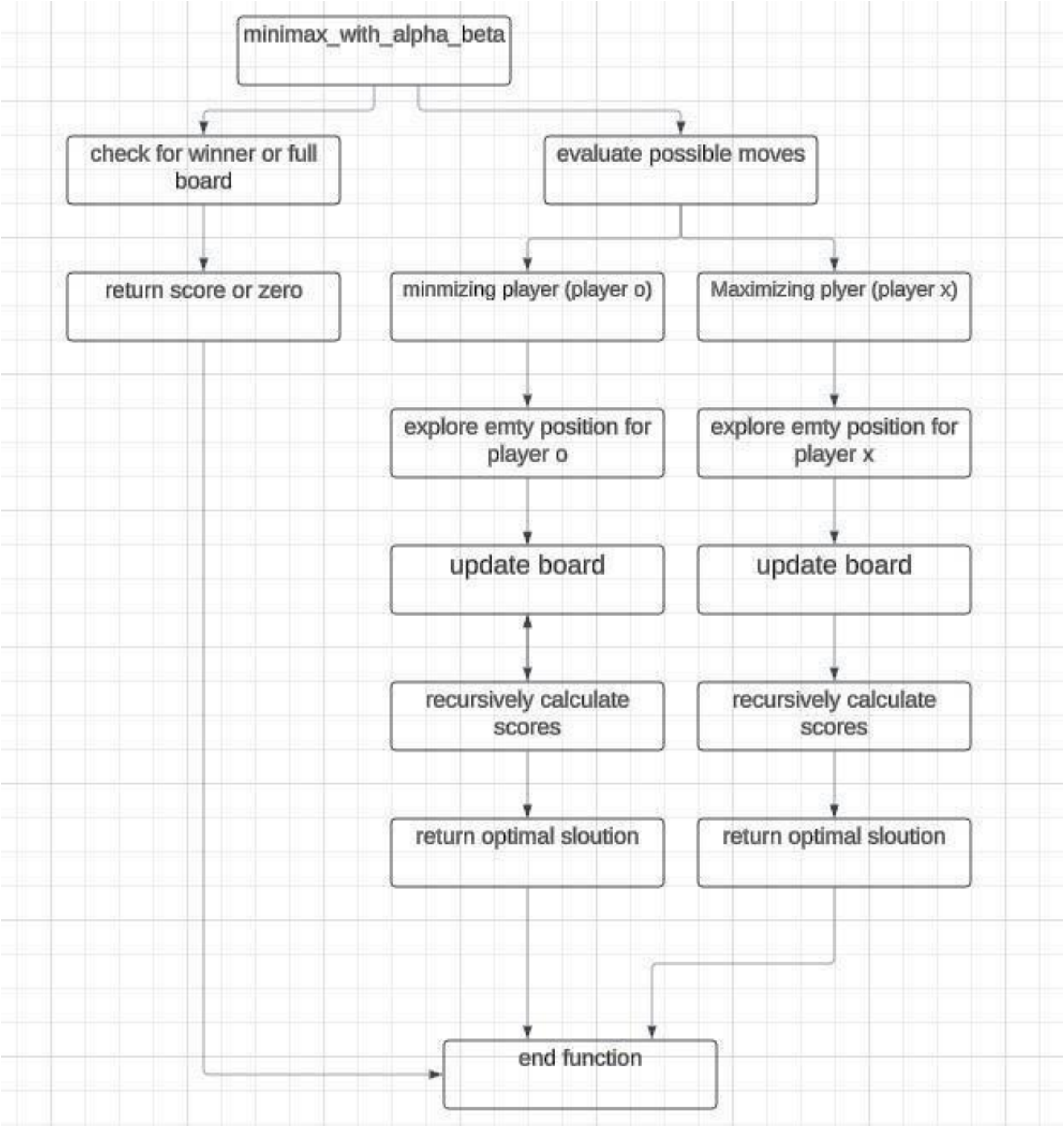
Alpha-Beta Pruning

- **Overview:**
  Optimized version of Minimax that eliminates branches of the tree that won't affect the outcome, reducing computation.
- **Design Rationale:**
  Improves performance by skipping unnecessary evaluations.

Flow chart:

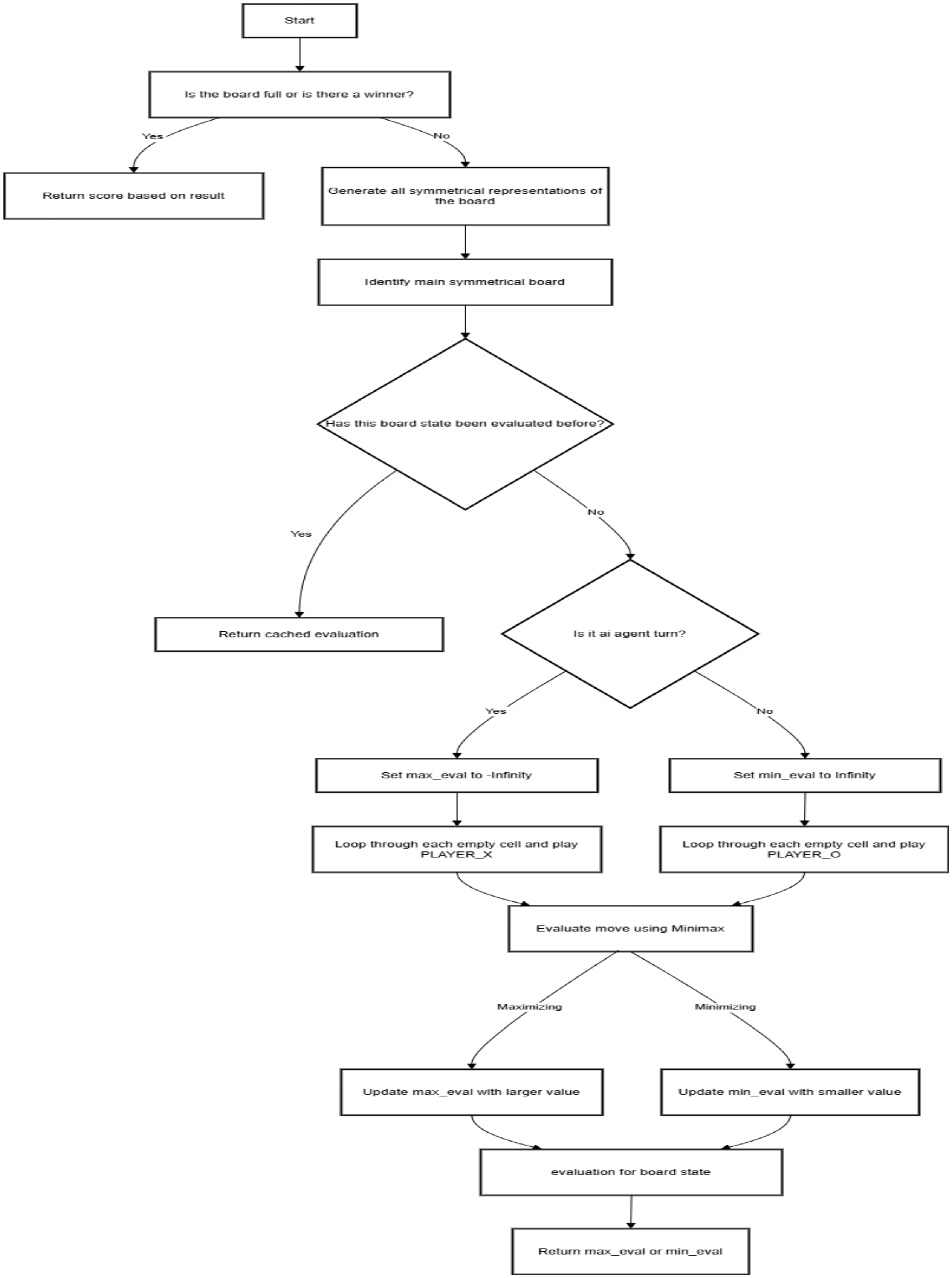

Start
Check winner or tie?
Yes
No
Return score ,or 0
ai agent`s turn?
Maxmizing
Minimizing
Initialize max_eval = -∞
Initialize min_eval = ∞
For each empty cell
For each empty cell
Make move
Make move
Call minimax with alpha-beta for minimizing player
Call minimax with alpha-beta for maximizing player
Undo move
Undo move
Update max_eval = max(max_eval, eval)
Update min_eval = min(min_eval, eval)
max_eval ≥ beta?
Return max_eval
min_eval ≤ alpha?
Return min_eval
No
Yes
Yes
No
End
Update alpha = max(alpha, max_eval)
Update beta = min(beta, min_eval)

Block diagram:

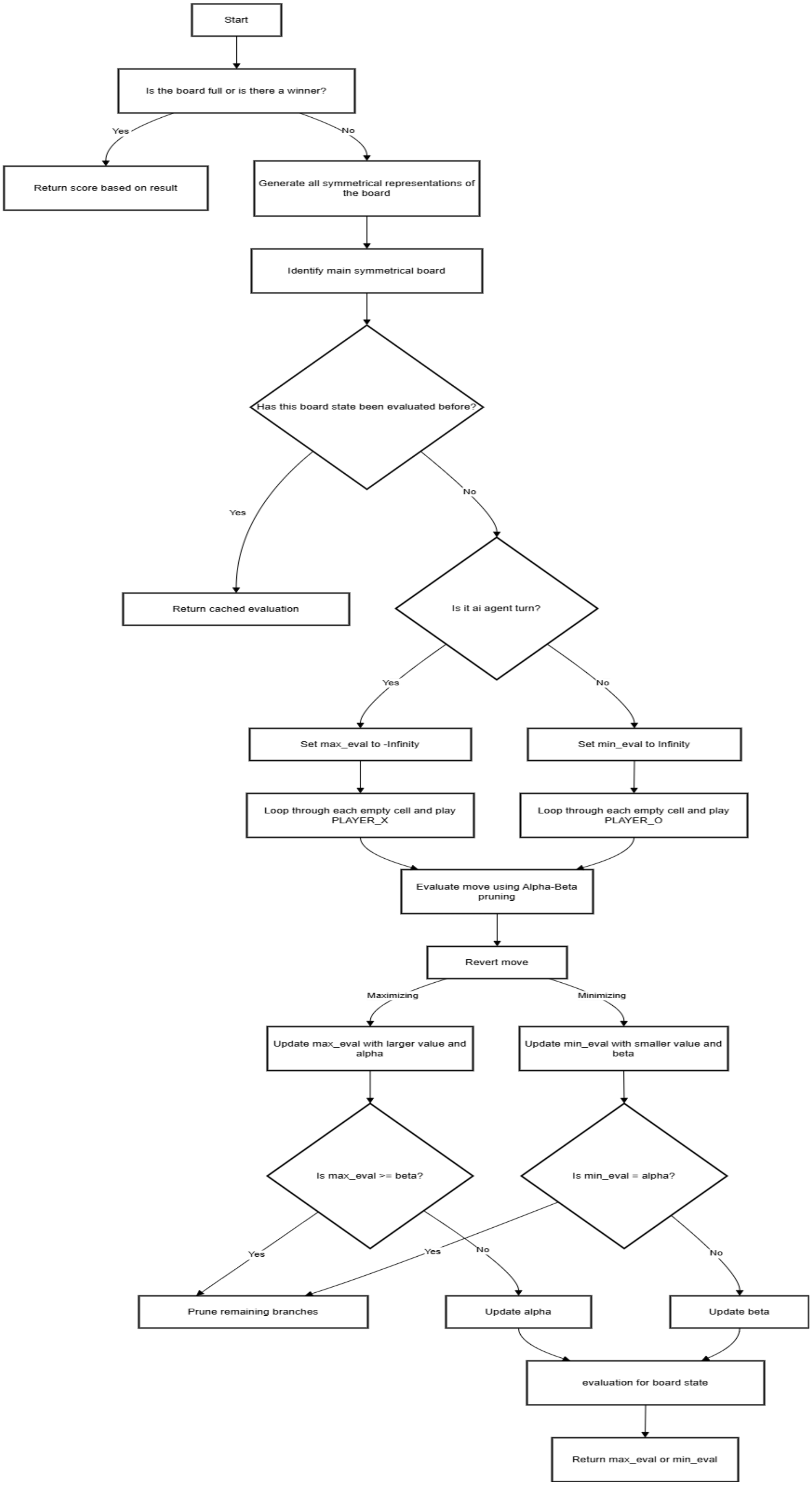

minimax_with_alpha_beta
check for winner or full board
evaluate possible moves
return score or zero
minmizing player (player o)
Maximizing plyer (player x)
explore emty position for player o
explore emty position for player x
update board
update board
recursively calculate scores
recursively calculate scores
return optimal sloution
return optimal sloution
end function

MINIMAX_SYMMETRY_REDUCTION

(Block diagram):


Start
Is the board full or is there a winner?
Yes
No
Return score based on result
Generate all symmetrical representations of the board
Identify main symmetrical board
Has this board state been evaluated before?
Yes
No
Return cached evaluation
Is it ai agent turn?
Yes
No
Set max_eval to -Infinity
Set min_eval to Infinity
Loop through each empty cell and play PLAYER_X
Loop through each empty cell and play PLAYER_O
Evaluate move using Minimax
Maximizing
Minimizing
Update max_eval with larger value
Update min_eval with smaller value
evaluation for board state
Return max_eval or min_eval

ALPHA_BETA_SYMMETRY_REDUCTION (Block diagram):

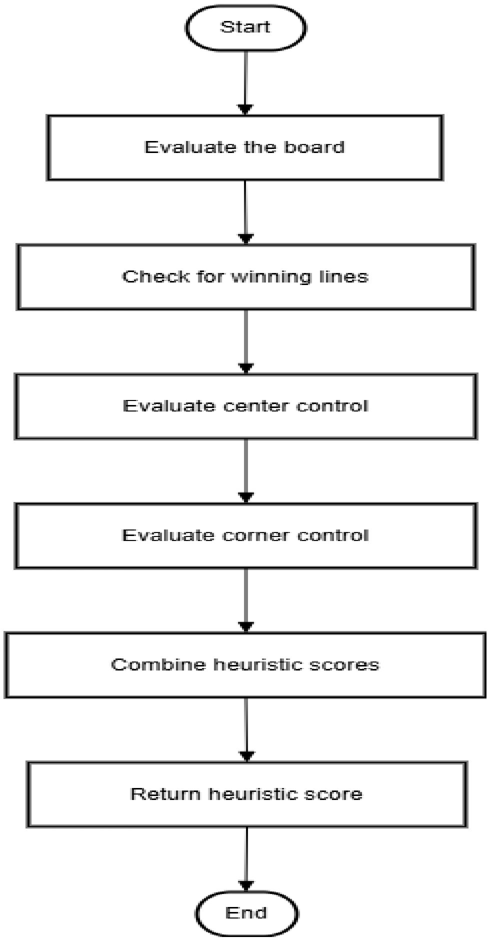

Start
Is the board full or is there a winner?
Yes
No
Return score based on result
Generate all symmetrical representations of the board
Identify main symmetrical board
Has this board state been evaluated before?
Yes
No
Return cached evaluation
Is it ai agent turn?
Yes
No
Set max_eval to -Infinity
Set min_eval to Infinity
Loop through each empty cell and play PLAYER_X
Loop through each empty cell and play PLAYER_O
Evaluate move using Alpha-Beta pruning
Revert move
Maximizing
Minimizing
Update max_eval with larger value and alpha
Update min_eval with smaller value and beta
Is max_eval >= beta?
Is min_eval = alpha?
Yes
Yes
No
No
Prune remaining branches
Update alpha
Update beta
evaluation for board state
Return max_eval or min_eval

Heuristic Functions

- **Design Choices:**
  - Winning line potential: Scores board configurations based on how close the AI or the opponent is to winning.
  - Center control: Rewards positions that dominate the center.
  - Corner control: Rewards positions that control the corners of the board.

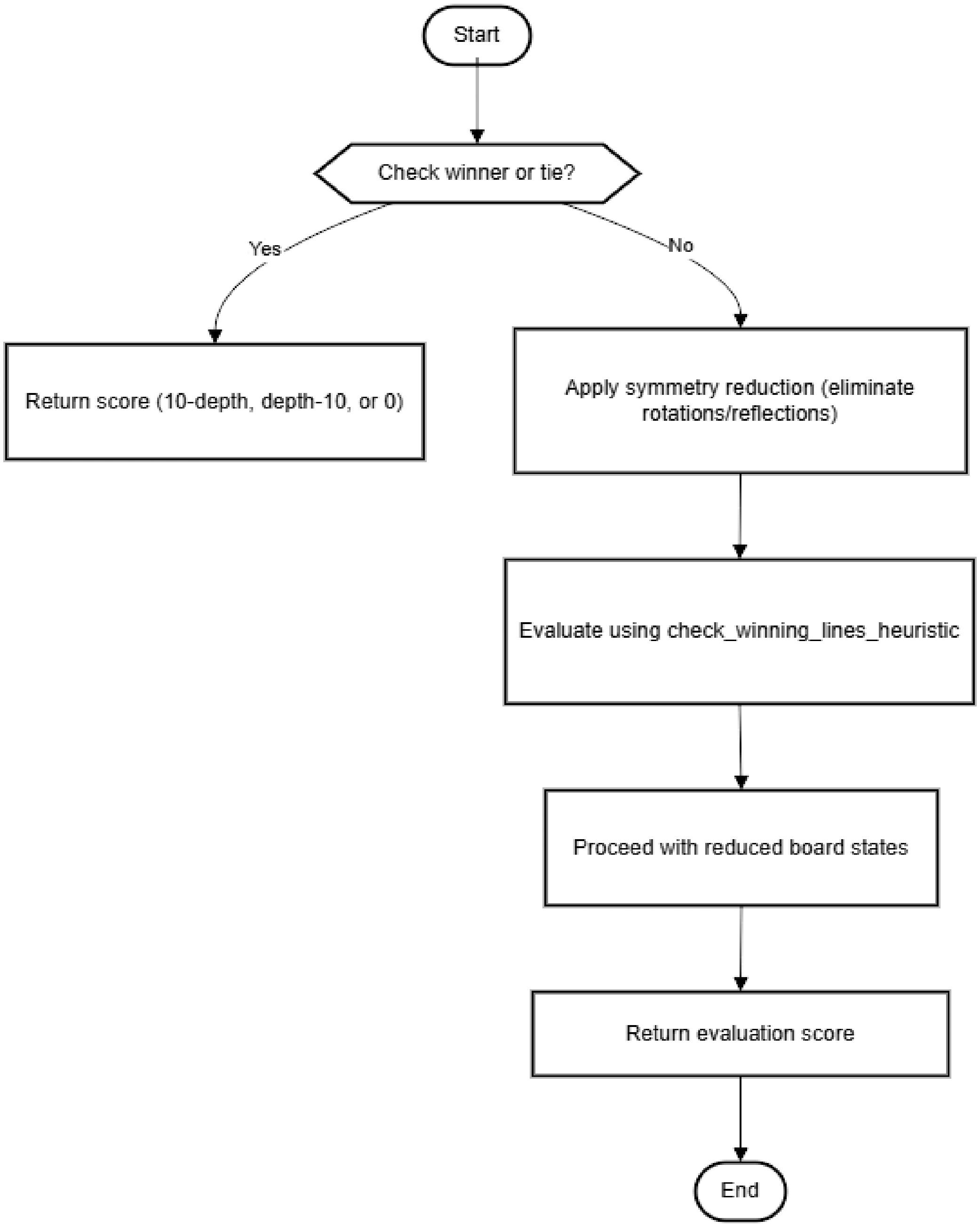
  - Combined heuristic: Weighted combination of the above.

Flow chart for **heurisitic evalution**:

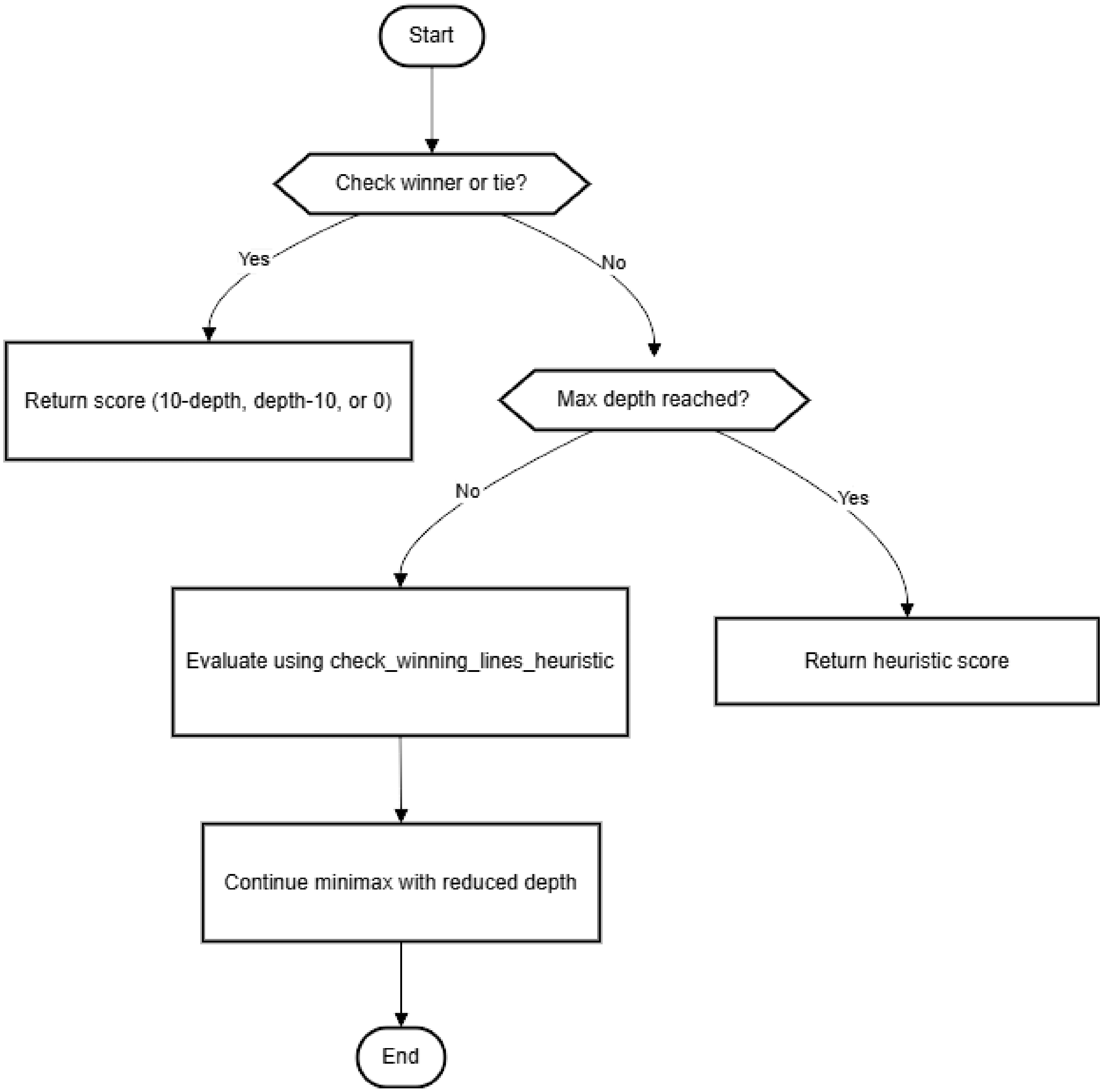


Heuristic_1 diagrams (Heuristic reduction & summetry reduction) :

HEURISTIC_1_SYMMETRY_REDUCTION

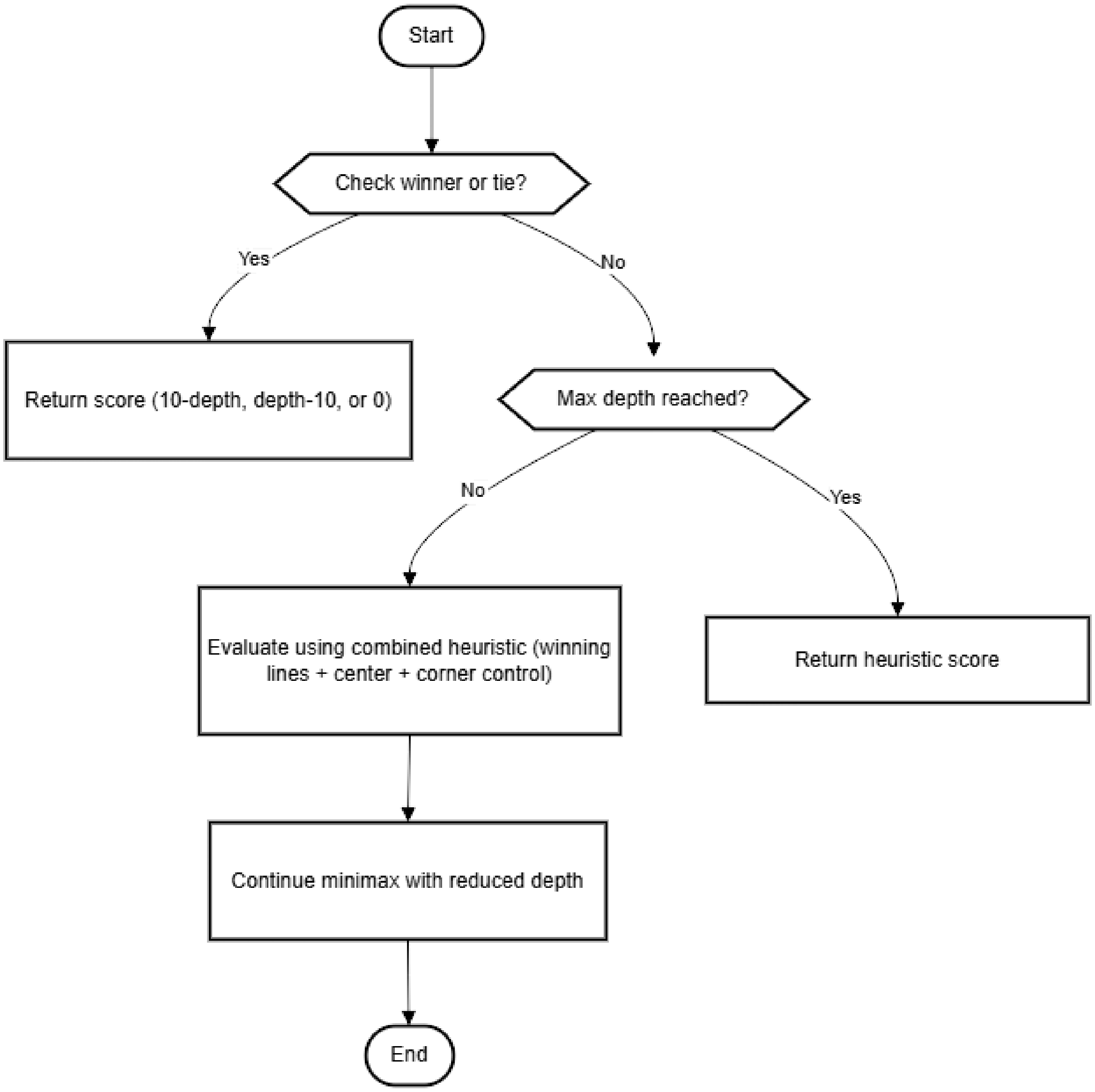

Start
Check winner or tie?
Yes
No
Return score (10-depth, depth-10, or 0)
Apply symmetry reduction (eliminate rotations/reflections)
Evaluate using check_winning_lines_heuristic
Proceed with reduced board states
Return evaluation score
End

HEURISTIC_1_HEURISTIC_REDUCTION

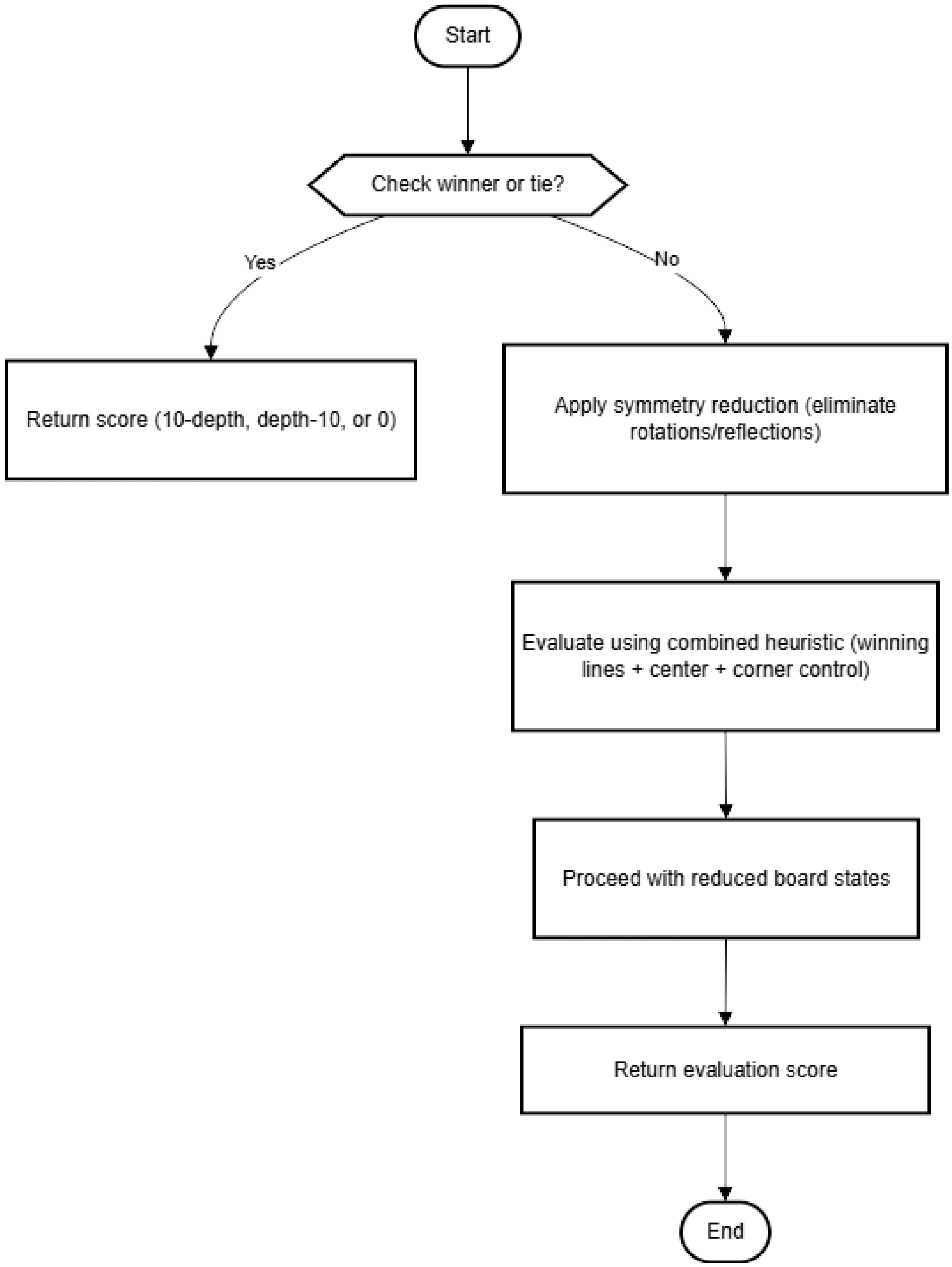

Start
Check winner or tie?
Yes
No
Return score (10-depth, depth-10, or 0)
Max depth reached?
No
Yes
Evaluate using check_winning_lines_heuristic
Return heuristic score
Continue minimax with reduced depth
End

Heuristic_2 diagrams (Heuristic reduction & summetry reduction) :

HEURISTIC_2_HEURISTIC_REDUCTION

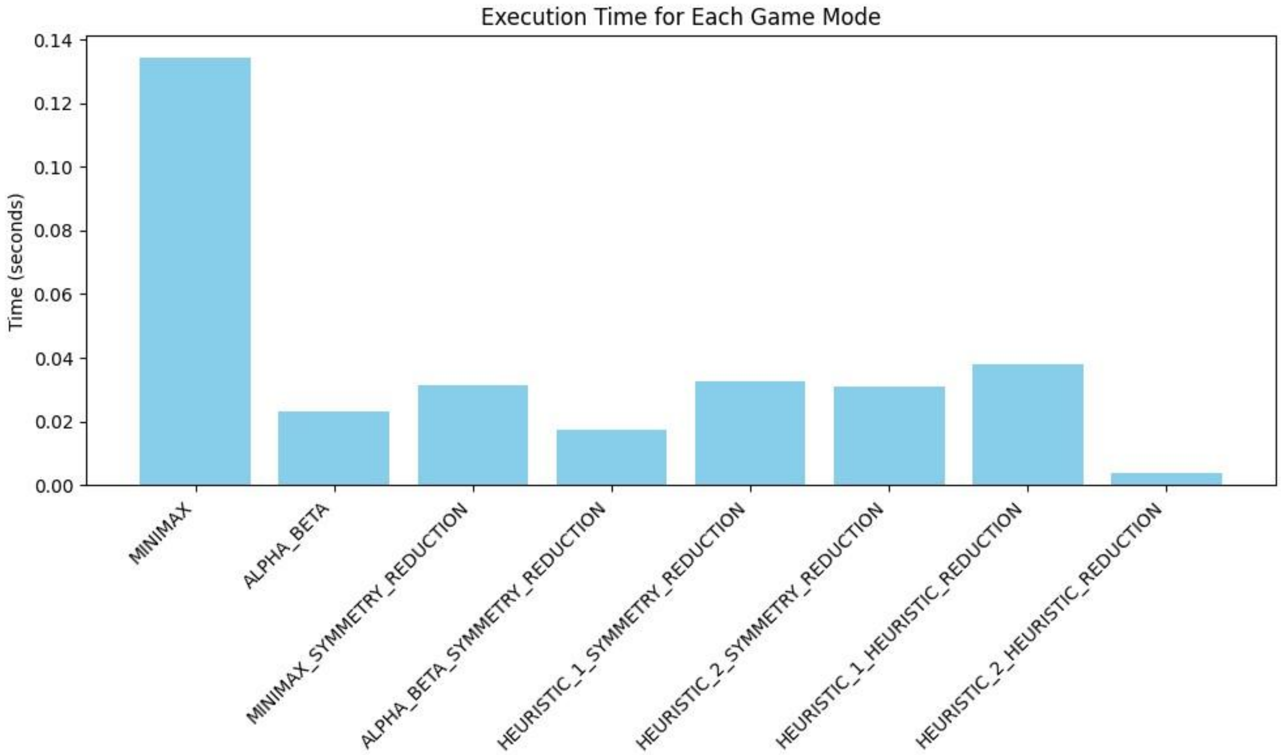

Start
Check winner or tie?
Yes
No
Return score (10-depth, depth-10, or 0)
Max depth reached?
No
Yes
Evaluate using combined heuristic (winning lines + center + corner control)
Return heuristic score
Continue minimax with reduced depth
End

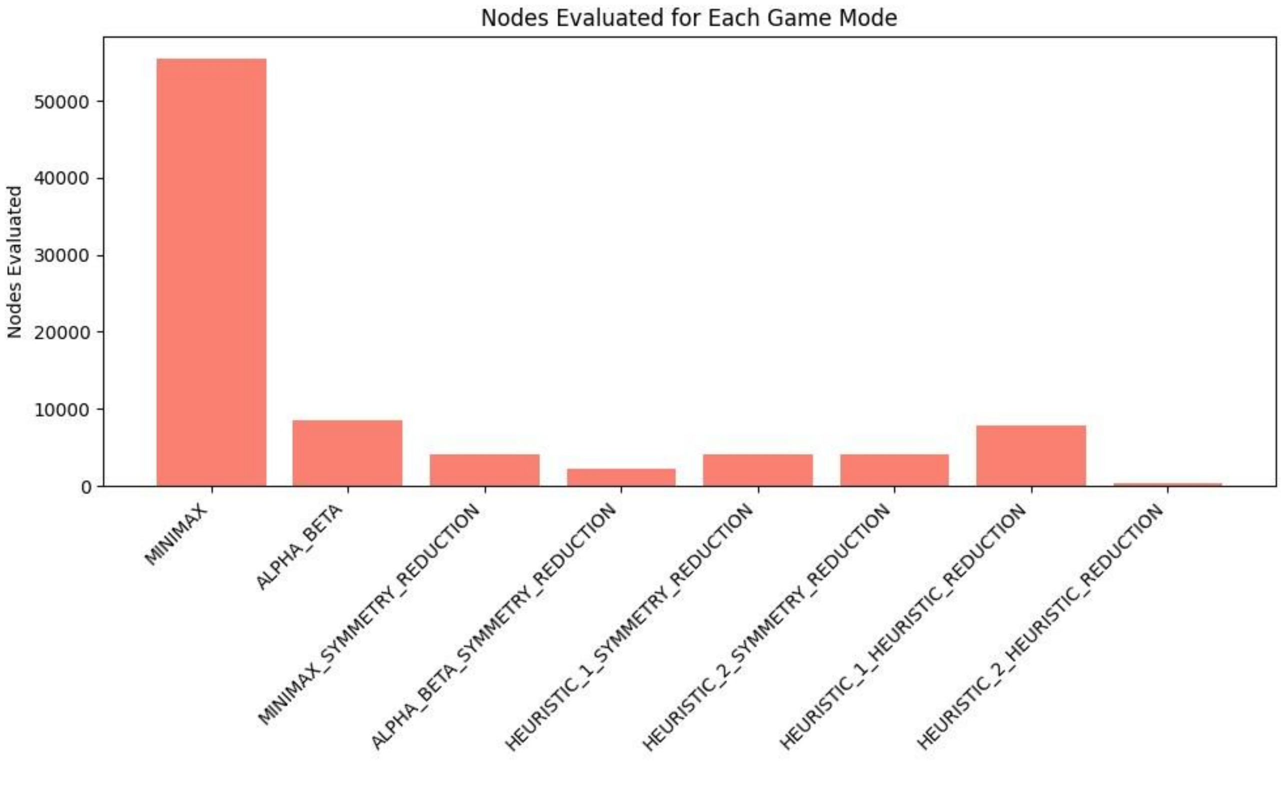
HEURISTIC_2_SYMMETRY_REDUCTION


Start
Check winner or tie?
Yes
No
Return score (10-depth, depth-10, or 0)
Apply symmetry reduction (eliminate rotations/reflections)
Evaluate using combined heuristic (winning lines + center + corner control)
Proceed with reduced board states
Return evaluation score
End

4.Experiments & Results

Plots displays Experiments & Results :





5. Analysis, Discussion, and Future Work

Analysis of the Results

The performance of different game-solving algorithms under various configurations. Here are the key insights from the data:

- **Performance Efficiency:**
  - **Alpha-Beta Pruning** consistently outperforms **Minimax** in execution time and node exploration, as expected due to its inherent pruning mechanism that reduces the search space.
  - The addition of **Symmetry Reduction** further enhances the efficiency across all algorithms by eliminating redundant computations caused by symmetrical states.
- **Heuristic-Based Approaches:**
  - Heuristic-based methods, especially with Symmetry Reduction, achieve optimal performance in specific configurations, such as significantly reduced computation time and fewer nodes explored.
  - **HEURISTIC_2_HEURISTIC_REDUCTION** stands out as the fastest algorithm, with the least number of nodes and states explored.

Advantages and Disadvantages

- **Advantages:**
  - **Minimax with Symmetry Reduction** offers a balanced trade-off between simplicity and efficiency by leveraging reduced state-space calculations.
  - **Alpha-Beta Pruning** demonstrates superior performance in deeper game trees by efficiently discarding irrelevant branches.
  - **Heuristic-Based Methods** exhibit robust speed improvements and adaptability for solving specific game states where domain knowledge is integrated into the decision-making process.
- **Disadvantages:**
  - **Minimax** is computationally expensive without enhancements and becomes infeasible for complex games.
  - While **Alpha-Beta Pruning** is efficient, its performance can still degrade if the tree is unbalanced or poorly ordered.
  - Heuristic methods, while fast, may trade off accuracy or optimality due to their reliance on approximations.

Behavior of Algorithms

The observed behaviors are attributed to the following reasons:

- **Symmetry Reduction:** Algorithms leveraging symmetry reduction excelled due to the elimination of redundant symmetric states, resulting in fewer computations.
- **Heuristic Functions:** The quality and specificity of heuristic functions directly influenced the performance. Accurate heuristics led to better pruning and faster decisions.
- **Algorithmic Design:** Alpha-Beta Pruning's ability to eliminate unnecessary evaluations (using upper and lower bounds) explains its improved performance compared to Minimax.

future modifications

- **Learning-Based Heuristics:** Incorporating machine learning techniques to develop adaptive heuristics based on past game data can improve decision accuracy and adaptability.
- **Parallel Processing:** Leveraging multi-core processors or GPUs for parallelizing state-space exploration could significantly speed up performance.