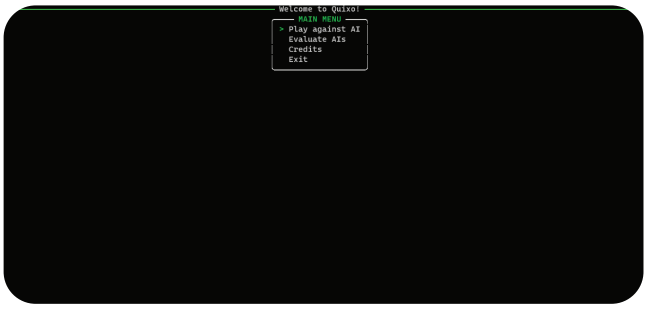
QUIXO

IF YOU'RE CURRENTLY USING A BRIGHT-THEMED TERMINAL, SWITCHING TO A DARK THEME MAY ENHANCE YOUR USER EXPERIENCE.

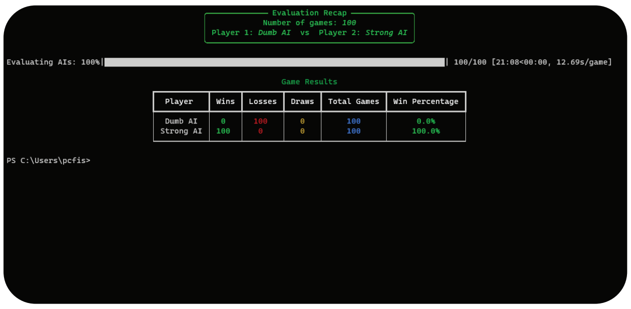
README
SUBMISSION: 19 FEBRUARY 2024 at 2.00 AM

Introduction & relevant aspects

Assuming we are all familiar with this turbocharged version of Tic-Tac-Toe, let's not spend time on explanations and dive straight into the discussion.
Let's start by clarifying what I mean for Draw because in the actual Quixo game, only two outcomes are available: Win or Lose. A Draw is determined when the total number of moves in the game exceeds 60 or a repeated pattern (the last sequence of 10 moves) is found in the game's move history. This has been done in an optic to avoid stales and consequently an infinite loop when making certain AIs playing against each other.
I have attempted to keep many of the original functions supplied intact to facilitate faster code comprehension: I only did some refactoring to the code provided and changed something in the play function mainly to address the Draw state checks and to make possible the interactive interface to work properly without making uneces-sary wrappers.
A custom menu was developed, granting users the ability to play against the AIs (also displaying the move history), and to evaluate any of them against a Random player or against each other as well.



```
Welcome to Quixo!
   MAIN MENU
 > Play against AI
   Evaluate AIs
   Credits
   Exit
```

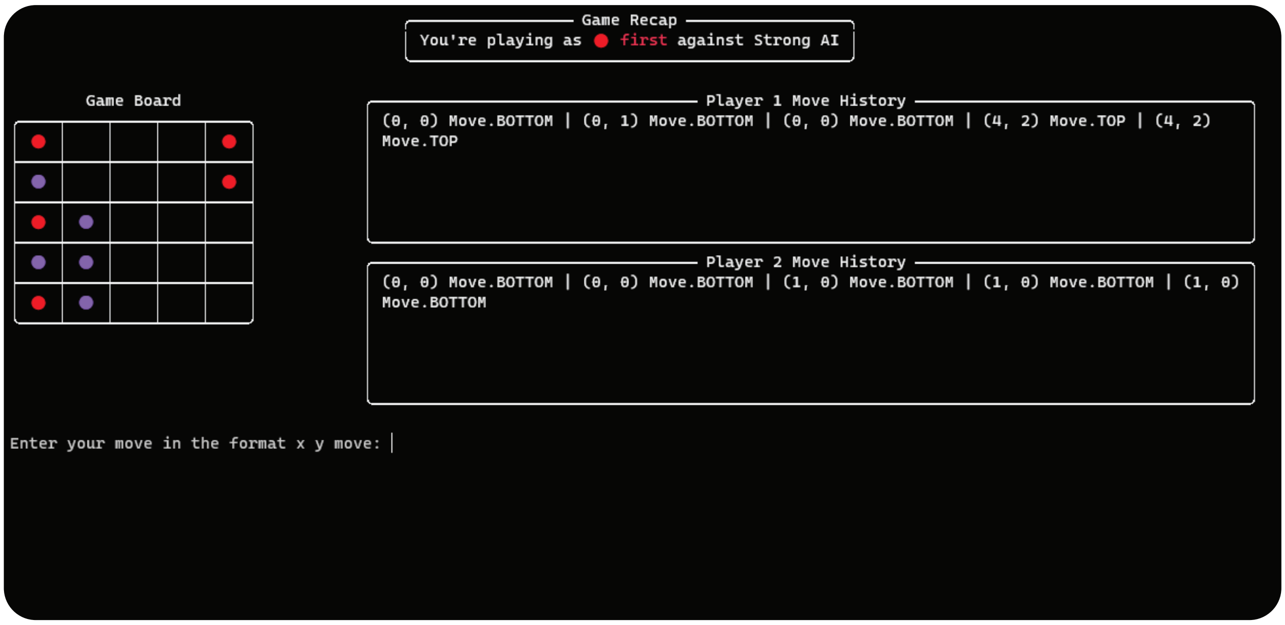


```
                    Evaluation Recap
                 Number of games: 100
       Player 1: Dumb AI  vs  Player 2: Strong AI

Evaluating AIs: 100%|██████████████████████████| 100/100 [21:08<00:00, 12.69s/game]

                      Game Results
  Player    | Wins | Losses | Draws | Total Games | Win Percentage
  Dumb AI   |  0   |  100   |   0   |    100      |     0.0%
  Strong AI | 100  |   0    |   0   |    100      |    100.0%

PS C:\Users\pcfis>
```



```
                         Game Recap
           You're playing as 🔴 first against Strong AI

    Game Board
                        Player 1 Move History
 🔴 |    |    |    | 🔴   (0, 0) Move.BOTTOM | (0, 1) Move.BOTTOM | (0, 0) Move.BOTTOM | (4, 2) Move.TOP | (4, 2)
 🟣 |    |    |    | 🔴   Move.TOP
 🔴 | 🟣 |    |    |
 🟣 | 🟣 |    |    |      Player 2 Move History
 🔴 | 🟣 |    |    |      (0, 0) Move.BOTTOM | (0, 0) Move.BOTTOM | (1, 0) Move.BOTTOM | (1, 0) Move.BOTTOM | (1, 0)
                        Move.BOTTOM

Enter your move in the format x y move: |
```

What strategy has been used?

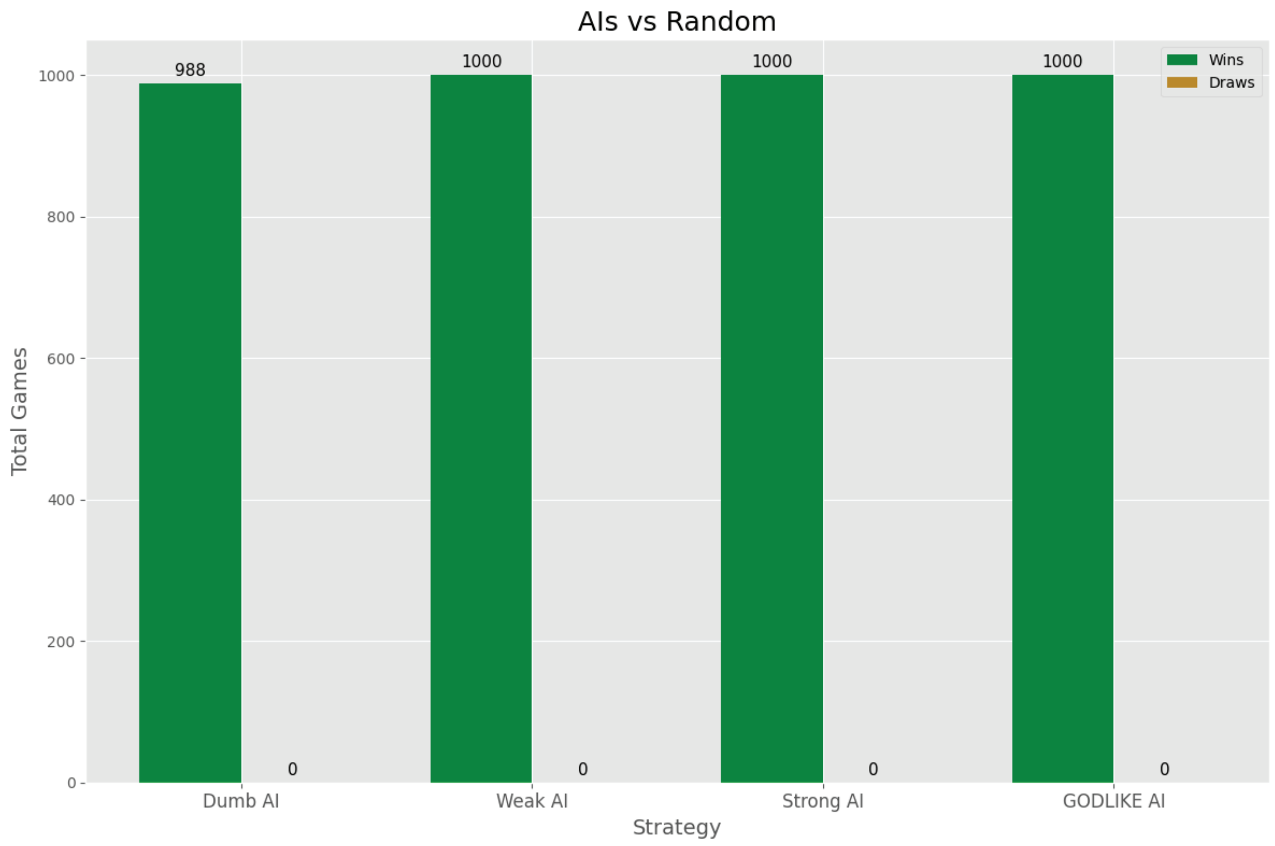
Let’s now talk about the strategy.
Quixo has a well-defined state space (all board configurations) and action space (all possible moves).
Since there it a consistent number of possible states the use of RL would be challenging, expecially with limited computational resources available for training.
For this reason, I thought the most suitable option was to implement a **recursive Minimax algorithm** in order to minimize the possible loss, always considering worst-case scenarios.
Since, as I said before, there a lot of states and therefore it seems like not possible to explore them all, Minimax comes along with a **Depth-Controlled Search** and a **Custom Evaluation Function**, making it able to evaluate every state assigning a value (more specifically for every state in which I find a win/lose, I assign ±infinite and for all the other states I return an evaluation based on the number of AIs’ pieces minus the number of the opponent’s pieces). The logic behind it is that the more pieces you have on the board, the more likely you are to win. Of course, this is under the constraint that the AI can foresee many moves ahead and determine if the opponent has the possibility to win.
I developed four different Agents, everyone able to explore the tree with a different **maximum depth limit**:

- **Dumb AI** (max_depth = 1)
- **Weak AI** (max_depth = 2)
- **Strong AI** (max_depth = 3)
- **GODLIKE AI** (max_depth = 4)

Minimax is also paired with **Alpha-Beta Pruning**, as it significantly reduces the execution time of the algorithm.
Dumb and Weak AI exhibit imperceptible response times, while Strong AI could take up to one second to perform a move, and GODLIKE AI might require approximately 5 to 10 seconds.

What are the results?

Testing AIs against a Random Player gave amazing results: (and also some joy...)



When the AIs were tested against each other, it became evident that the Dumb AI could be easily defeated. However, in tests involving the Weak AI or higher levels, the game was more likely to end in a draw.

THE END (or the beginning...)