Exercise 7:
Implementing the Rules for a Simple Board Game

January 2, 2025

Instructions

In this project, you will choose a simple board game and implement its rules and dynamics.

1. Select a game from the provided list, or propose your own game (subject to approval). Learn its rules thoroughly and play a few rounds to understand its dynamics.
2. Write a Python class to represent the game. This class should implement the following methods:
   (a) `__init__`: Start a new game.
   (b) `make_move`: Apply a move to the current game state.
   (c) `unmake_move`: Reverse a move. In some games, this is easier if `unmake_move` receives the move to unmake as an input.
   (d) `clone`: Create a deep copy of the current game state.
   (e) `encode`: Encode the game state as a binary vector. Ensure that all relevant information—such as whose turn it is, scores, or other game-specific features—is included.
   (f) `decode`: Translate an action index into a move in the game.
   (g) `status`: Return the game result if the game is over, or indicate that the game is ongoing.
   (h) `legal_moves`: Return a list of the legal moves in the current position.
3. Debug your game thoroughly! Simulate complete games to verify that all rules are implemented correctly. Pay close attention to edge cases and confirm that the game behaves as expected in every scenario.

Remarks

- You may choose whether each node in the game tree holds a copy of the game state or uses `make_move` and `unmake_move` to traverse the tree. For example, in Othello, moves involve flipping pieces. To support `unmake_move`, the game must remember which pieces were flipped. In such cases, storing a copy of the game state at each node might be simpler and faster.
- For efficiency it is sometimes better to check in the `make_move` function if the move ends the game and if so to update the status, rather than checking the whole board for a winning configuration in the `status` function.
- Think about what the neural network will need to make decisions. Include all relevant information, such as the current scores and the player turn. You might also include simple precomputed features (such as material count in chess) if they are relevant to your game.
- In the decode function, ensure that every possible move corresponds to a unique index. Think carefully about how to enumerate moves in your game.