Color Wars (AI)

> Whoever said the pen is mightier than the sword obviously never encountered automatic weapons.
>
> *Douglas MacArthur*

Problem Statement

Each game consists of P ($2 \leq P \leq 4$) players on an $N \times N$ board of cells ($64 \leq N \leq 256$, chosen uniformly at random). Each player is assigned a unique color (numbered 0 to $P - 1$). Each board grid is either empty, colored with a single color, or colored with multiple colors.

At the beginning of each round, if a cell is colored with a single color, then for each of the four cardinally-adjacent cells, if the cell is empty then it is colored with the same color as the original. All expansions happen simultaneously. It is possible that two or more single-colored cells with different colors expand into the same cell at the same time; this creates a colored cell with multiple colors. Note that multi-colored cells do not expand in future rounds.

Then, each player selects an empty cell and colors it with their respective color. If two players select the same cell, then the cell selected is shared. If an invalid cell is selected (e.g. colored or outside of the grid), the engine will skip the player's turn. This process continues until all cells are colored.

A player's score is calculated by the proportion of the grid they occupy. Specifically, if a player occupies A cells, their score is $A \ / \ (N \cdot N)$. Multi-color cells are split evenly between all players that occupy the cells (e.g. if a cell contains colors 1, 2, and 4, then that cell contributes $1l3$ to each of the respective players' scores). The winner is decided by who occupies the most area by the end.

Implementation

You are to implement a class that inherits the provided class `Player`.

The player class must implement two functions, `__init__`, which is called once at the beginning of the game, and `play` which is called in every turn.

`__init__` takes as input:

- `player_index (int)`: The unique ID (an integer from 0 to $P - 1$) that is assigned to the player.

- `grid_size (int)`: The side length of the map in the current match.
- `num_players (int)`: The number of players in the current match.

`__init__` does not provide anything as output.

`play` takes as input:

- `board (List[List[int]])`: An $N \times N$ grid representing the state of the board immediately before the player's turn. Each cell contains an integer from 0 to $2^P - 1$.
    - The color(s) of a cell is determined by the binary representation of the integer it contains. Specifically, if the i-th binary digit is set, then the cell contains color i. For instance, a cell containing 11 has colors 0, 1, and 3 because its binary expansion is 1101.
- `history (List[List[Tuple[int, int]]])`: A $T \times P$ matrix that stores the positions of previous moves. The j-th entry in the i-th row is a tuple that stores the x and y positions of the move player j played in turn i.

`play` must provide as output a tuple `(x, y)`, the position to play the next move. This must satisfy $0 \leq x, y \leq N - 1$.

Scoring

In every match, you will be matched against one to three other players. A total of 10 independent games will be played for variance reduction. Each player's score for that match will be calculated as their average score over the games. Leaderboard ratings are calculated according to the Glicko rating system. Matches will occur every 30 minutes. You will be matched against others with similar ratings.

Constraints

Your submission must adhere to the following constraints:

- **Output Query Time Limit:** 1s.
- **CPU Time Limit:** 10s.
- **Wall Time Limit:** 60s.
- **Memory Limit:** 2^{28} bytes.

If your submission provides an error or violates one of the above constraints, your turn will be skipped for the remainder of the game.

Submission

Implement your final submission in `submission.py` and upload said file to the submission server. **Do not** modify the name of the class (`SubmissionPlayer`), and **do not** use print

statements. You may instead use the `logging` library (see example) to debug. Debug outputs will be returned as feedback (see below).

Feedback

After each match, you will receive detailed feedback in JSON format containing:

Metadata (`meta`):

- The size of the map (`gridSize`)
- The number of players in the match (`numPlayers`)
- The number of games played in the match (`numGames`)
- Your player index (`playerIndex`)

For the overall match (`summary`):

- Your total average score across all games (`totalScore`)
- Your rank in the match (1-4) (`rankInMatch`)
- A list of your scores for each individual game (`gameScores`)

For each game (`games`):

- Turn-by-turn information including (`turns`):
    - The turn number (`time`)
    - The move your bot submitted (`yourMove`)
    - The move submitted by each player that turn (`allMoves`)
- Your final score this game (`finalScore`)
- The number of turns the game lasted (`turnCount`)

Any debug output from your code (using the logging library) will also be included in the feedback (`debugLog`).

You may visualize the returned feedback to view each game in a match. See the `README.md` file in the starter code for details.

Clarifications