Bid (AI)

The meek shall inherit the earth, but not the mineral rights.

J. Paul Getty

Flavor Text

There is no wealth. Only leverage.

In the not-so-distant future, value is no longer created. It's extracted. As a bidding executive at the Global Arbitrage Conglomerate, you fight over the economy's remains with three rivals. Each round reveals a new target: a company, a contract, or a toxic liability.

Your fifteen influence tokens are your arsenal. Some prizes still hold value. Others are poison. Your job isn't to build value. It's to avoid the fallout.

Welcome to pure arbitrage, where fortune favors the precise.

Problem Statement

Each game consists of 4 players. At the start of the game, each player receives an identical hand of fifteen *bidding cards*, numbered 1 through 15. These cards are used to place bids during the game.

Distinct from the bidding cards are fifteen *score cards*: ten positive cards numbered 1 through 10, and five negative cards numbered -5 through -1. All fifteen score cards are shuffled face down into a single deck.

The game consists of fifteen rounds. At the start of each round, the top score card from the deck is revealed. All players simultaneously select and reveal one bidding card from their hand, representing their bid for that score card. For positive score cards (1 to 10), the player with the *highest unique bid* wins the score card. For negative score cards (-5 to -1), the player with the *lowest unique bid* wins the score card. If no unique bid exists, no one wins the score card. After each round, all used bidding cards are discarded and cannot be used for the remainder of the game.

After all fifteen rounds, each player's score is computed as the sum of all score cards they won over the game. Players aim to maximize their score.

You are tasked with writing an automated strategy to play the above game.

Implementation

You are to implement a class (which inherits the provided `Player` class) to play the above game.

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

`__init__` takes as input:

- `player_index (int)`: The index that is assigned to the player.

`__init__` does not provide anything as output.

`play` takes as input:

- `score_card (int)`: The number of the score card revealed this round.
- `player_history (List[List[int]])`: A list whose i-th element contains the bid history of player with index i.

`play` must provide as output an integer `x`, the bid card to reveal this round.

Scoring

In every match, you will be matched against three other players. A total of 100 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:** 3s.
- **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, random moves will be used as fallback 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 random seed (`seed`)
- 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`):

- The order of score cards that were revealed (`scoreCardOrder`)
- Turn-by-turn information including (`turns`):
    - The score card revealed that turn (`scoreCard`)
    - Your bid (`yourBid`)
    - All players' bids (`gameScores`)
    - Which bids were unique (`uniqueBids`)
    - Whether you won the card (`won`)

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

Clarifications