Preface

We have received a thumbs up from Professor Weinberg, who said to just make sure there is "no copy pasting" on this assignment. It is encouraged to interpret what we are doing into your own words, and even your own code. Strictly following these instructions is also completely ok. Welcome to the coalition :)

Introduction

This document explicitly describes how to implement each of the four required functions in this week's strategy design using the coalition strategy. For details about why the following works, see the explanation document.

Please download the starter code, complete coding the strategy below, and submit to the leaderboard by the end of Friday, April 30th so that we can be sure that the coalition strategy is working as expected and your implementation is correct.

Name your file **`Miner_coalition_[netId].java`**. The naming allows us to see how many people are participating in the coalition. This is important to determine when we do not yet have enough people in the coalition to give large scores.

This is the recommended approach:

1. duplicate or copy the file **`Miner_bribable.java`**
2. change the file name and class name appropriately to **`Miner_coalition_[netId]`**
3. change the name of `getDesiredChain()` to `getBlockToMine()` and
    a. change the function scope of the new `getBlockToMine()` from `private` to `public`
    b. delete the other function named `getBlockToMine()`
        i. (lines 38-40)

Required Methods

In your final code, you must have the following four methods, each of is explained below:
`refreshNetwork, getBlockToMine, getAmountToSpend, getAmountToBribe`

`getBlockToMine`

You must select the `BlockChain` object where you have the most currency (stake) not including the most recent bribe, tiebreaking in favor of the longest chain. The recommended method is as follows (in order):[1]

[1] Optional: to maintain same variable naming, rename the four instances `maxBribe` to `maxStake`

1. [**inside the `for` loop**] Create a local variable `stake` of type `double` inside the `for` loop and set it equal to `block.getStakeForMiner(miningIndex)`
2. If `block.getMiner()` equals your `miningIndex`, subtract `block.getPrev().getBribeAmount()` from `stake`
3. Replace each of the <u>three</u> instances of `block.getBribeAmount()` with `stake`

`refreshNetwork`

You must save the previous blocks and your miner's index. If you copied **`Miner_bribable.java`**, leave this as is.

`getAmountToSpend`

You must spend your entire stake whenever there is a `BlockChain` object that is a longest chain and where you have the most stake.[2] If there is not such a block chain, spend nothing. The recommended method is as follows:

1. Create a local variable `expected` of type `BlockChain` and set it to equal the result of a call to your `getBlockToMine()` method
2. Iterate over each of the `BlockChain` objects in the `blocks` array
    a. This can be a `for` loop
    b. On each iteration, if `getRank()` of the current block is strictly greater than `expected.getRank()`, then return `0`
        i. This can be executed as an if statement
3. Otherwise, if no such block exists after all iterations, return `expected.getStakeForMiner(miningIndex)`

`getAmountToBribe`

There is never a need to bribe, so simply return `0`.

Local Testing

To ensure that your implementation is functioning as expected, you can edit **`miners.txt`** to contain the following strategies and then run *`make test`*.

- 2x longest
- 2x aggressive
- 2x bribable
- 8x coalition

=> most coalition miners should score 6 - 7 million

- 4x longest

[2] If you would like to test spending less than your entire budget, it is totally fine to play around with the amount you risk. Please spend at least 13 though, otherwise the strategy may not work as expected.

- 4x aggressive
- 4x bribable
- 8x coalition

=> most coalition miners should score 2.4 - 2.8 million

- 4x longest
- 4x aggressive
- 4x bribable
- 32x coalition

=> most coalition miners should score 2.7 - 3.3 million