The Lemonade Stand game is a three-player game played on a circular island. On each day (round), the three players simultaneously choose one of twelve equally separated positions on the perimeter of the island on which to set up their lemonade stand. We will number these positions $1, 2, \ldots, 12$, like on a clock face. Customers are uniformly distributed around the island and will patronize the stand that's nearest to them (and will evenly divide their business among stands that are equidistant from them). Thus, the profits for the players on a given day can be modeled as follows: each player's payoff is the sum of the distance to the nearest player in the clockwise and counter-clockwise directions (as measured by the number of spots separating them). This process is repeated for some finite number of rounds, and each player's total payoff is the sum of their profits from each round. Figure 1 illustrates some possible outcomes from a single day's play. In the left-most case, where all players have chosen a different location for the day, the red player's payoff is $5 + 4 = 9$, the green player's payoff is $3 + 5 = 8$, and the blue player's payoff is $4 + 3 = 7$. In the middle case, where the red and blue players have chosen the same location, they both receive a payoff of 6, with the green player receiving 12. In the right-most outcome, all three players have coincidentally chosen the same location on the island and they all receive 8. This is an example of a *constant-sum game*: no matter the outcome, the sum of the payoffs to the players is always 24.

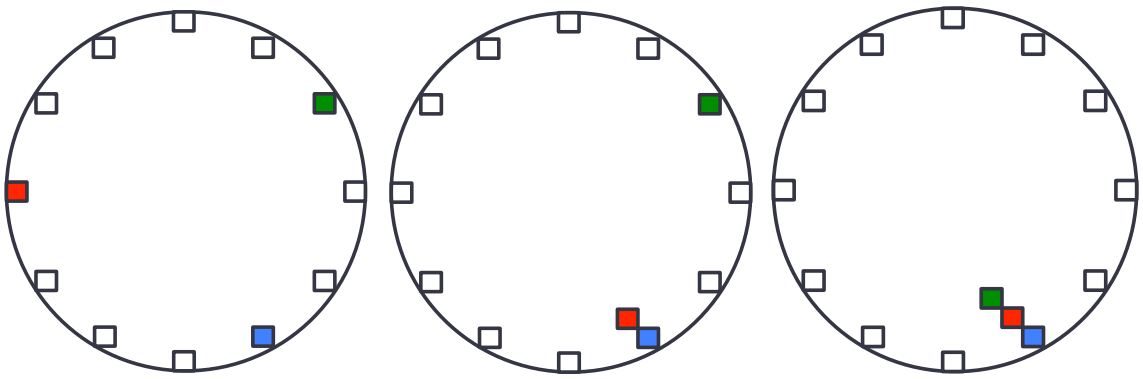

Figure 1: Sample outcomes from a single round of the Lemonade Stand game.

A Nash equilibrium in this game is a tuple of strategies (π_R, π_G, π_B) for the three players such that no player can improve their payoff by unilaterally changing their own strategy. The Lemonade Stand game has *many* Nash equilibria — for example, consider the case where the three players play the pure strategies $(9, 2, 5)$, i.e., the red player always chooses 9 o'clock, the green player always picks 2 o'clock, and the blue player always sets up at 5 o'clock (this will always result in the outcome shown in the left-most panel of figure 1). None of the

players in this scenario can improve their payoff by moving their stand, if their competitors do not move.

In a game where a glut of Nash equilibria exist, selecting the right strategy to play can get tricky — after all, the theory only predicts which outcomes are stable, but is agnostic about which equilibrium (if any) will actually be achieved in practice. The prospect of *coordination* raises even more interesting possibilities in a three-player game: two of the players could collude to always minimize the third player's score, while increasing their own payoffs. For example, suppose the red and green players enter into a pact to always pick spots that are diametrically opposite to each other on the island — then, the blue player's payoff is always limited to 6 on each round, with red and green splitting the remaining 18. The challenge is to determine how to achieve such cooperation with other agents without cheating, i.e., without explicit communication or pre-game arrangements. This is the question you will be exploring on this assignment.

In particular, your task is to create agents that attempt to maximize their payoff over a multi-round staging of the Lemonade Stand game, against a variety of opponents. You will enter one agent of your choice into a round-robin tournament against bots designed by your classmates — a portion of your grade on this project (5%) will be based on the performance of your agent in this contest. Bots that employ novel and well-motivated strategies, but that do not fare well in the tournament, will also receive full-credit on this portion to encourage intellectual risk-taking.

All of your bot entries must be written in the Java programming language.

Getting Started

Start by downloading the code archive on Moodle. The `Tournament` class runs a Lemonade Stand tournament between three named bots that lasts a specified number of rounds. All the tournament parameters can be specified on the command line. Every bot that you design must implement the `Bot` interface. For example bot implementations, refer to `RandomBot` and `FiveOClockBot`. To run a tournament between three bots, say two `RandomBot`s and a `FiveOClockBot`, first compile the two corresponding classes. Then, run the `Tournament` class from the terminal as follows:

```
java Tournament RandomBot FiveOClockBot RandomBot 1000
```

You should feel free to modify the `Tournament` class in any way that eases the experimentation and data collection process. There are two main requirements for the bots that you submit as your tournament entries:

- They *must* implement the `Bot` interface.
- They must operate within the stipulated time and space constraints — a 100000-round match between your entry and `RandomBot` should complete within 60 seconds using no more than 512MB of heap space on a standard campus Mac computer.

The Write-Up

Remember the overarching writing rule for this course: *you need to be sufficiently precise with your writing and include enough details that a competent reader could reproduce your results.* Here are some specific things to address in your write-up, in no particular order. This is *not* meant to be an exhaustive list.

- Describe the algorithm behind the entry to the class tournament. What approach did you use to model your opponents (if any)?
- Were there other promising strategies that you developed as well? How did you choose which algorithm would be your final entry to the class tournament? You should justify your choices with data.
- As always, cite your sources if you sought inspiration from the published literature.

Deadlines

- **Tuesday, Feb. 26, 11:55pm:** Optionally enter up to two of your bots for a "warm-up" class-wide tournament. Upload your bot entries via Moodle.
- **Friday, Mar. 1, 11:55pm:** Final version of papers due on Moodle. This version should contain your names, emails and the *Contributions* subsection. Also upload your final bot submission to the class tournament by this deadline. Due to constraints imposed by Spring Break, there will not be an opportunity for a peer review cycle on this assignment.

Recommended Timetable

Here's a recommendation for how to budget your time over the next couple of weeks as you work on this assignment.

- **Feb. 20–23:** Read the problem description, look over the provided code base, implement some simple bots and run some preliminary experiments. Try designing your own approaches for modeling opponents, or signaling to other contestants. How do these bots do? Write your *Introduction* section.
- **Feb. 24–26** Do background research and see what strategies have been studied for this game in the past. Also consider algorithmic approaches that have been used in other iterated games (for example, iterated Rock-Paper-Scissors or iterated Prisoner's Dilemma). Implement one or more of these and run some more experiments. Finalize your entry (or entries) to the warm-up tournament. Write your *Background* section.
- **Feb. 27–28:** Continue running experiments as necessary. Write your *Experiments*, *Results* and *Conclusions* sections.
- **Mar 1:** Wrap-up any pending experiments, write the the abstract, revise and proofread the entire paper and submit your final draft, as well as your entry to the final tournament.