Exercises 6

Task 6.1. Collective Optimization

Most collective operations that involve data transfer can be simulated with `Send()`/`Recv()` calls. For example, instead of calling `Bcast()`, you can send a separate message to each process via the `Send()` function (so if there are P processes in total, you will need to send $P - 1$ messages). Which solution would be more efficient? Why?

Hint: Suppose you are the author of `Bcast()`. How do you think to implement this function? How can you make it faster than $P - 1$ calls of `Send()`?

Task 6.2. Skunk Debugger

The code [ex06_skunkWithBugs.java]() implements an MPI-based simulator of a party game called [Skunk](). You can compile it with

```
javac -cp .;%MPJ_HOME%/lib/mpj.jar ex06_skunkWithBugs.java
```

and run for any arbitrary number of players:

```
# 3 players and a host
mpjrun -np 4 ex06_skunkWithBugs
```

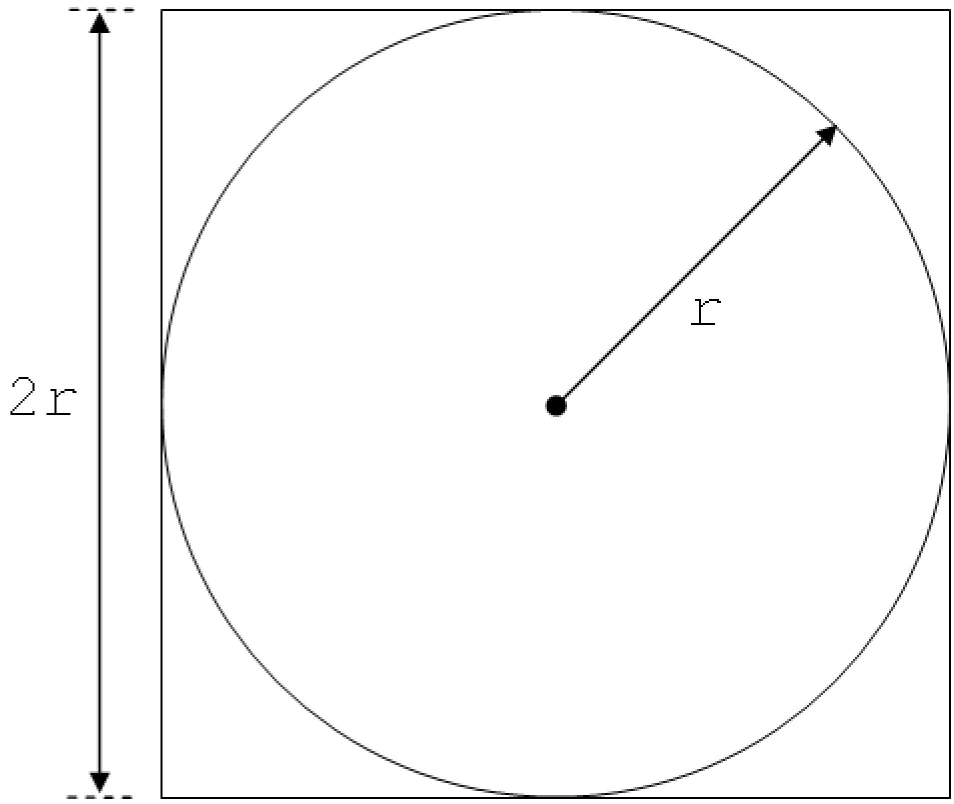
At the end, the host must announce the winner and the winner's score:

```
Winner: 3, Score: 17
```

Unfortunately, there are at least **three** bugs in the code that prevent the code from working properly. Please fix the bugs! You can presume that all intermediate printouts are for debugging; you only need to care about the final printed line.

Task 6.3. Finding Pi

The approximate value of π (3.14159⋯) can be found experimentally with a so-called "dartboard method" as follows. Let's draw a circle with a radius r and place it inside a square board:



–

Next, let's throw N darts into random points of our drawing and count the number of darts N_0 that happened to be inside the circle. The probability theory states that

N / N0 ≈ area-of-square / area-of-circle

(More precise results can be obtained with higher values of N). Since the area of our square is $(2r)^2$, and the area of our circle is πr^2, we can conclude that

$\pi \approx 4N_0 / N$

Write an MPI program that calculates the value of π using this method. The code should run P parallel processes generating random numbers. Each process makes N / P "throws", then the results are collected and the value of $4N_0 / N$ is reported to the user. You can specify the values of N and r via command line arguments.

Example:

```
>mpjrun -np 4 w06_findingPi 10000000 10000
Pi = 3.1415376
```

Task 6.4. Three-Player Tic-Tac-Toe

Create a simulation of a 3-player Tic-Tac-Toe game. The game is a modification of the classic [Tic-Tac-Toe]() with the following rules:

1. The 4 × 4 board is used instead of 3 × 3.
2. There are three players: “X”, “O”, and “#”.
3. It is enough to have three symbols in a row to win (four symbols in a row is a victory as well).

The program can be organized as follows:

The process with rank (ID) 0 is the “game host”. In a loop, it sends to each player the current state of the game field and a request to make the next move. It also displays the state of the game field after each turn. When the game is over, the host sends to each player a special shutdown command and declares the winner.

All other processes are “players”. The task of a player is to make a move in the situation, provided by the game host, and to shutdown upon request. You can implement any playing strategy, even random moves are acceptable.

Example of a game session:

```
>mpjrun -np 4 w06_ticTacToe  // run the game for 4 processes (host + 3 players)

X turn:
 . . . .
 . . X .
 . . . .
 . . . .

O turn:
 . . . .
 . . X .
 . . O .
 . . . .

# turn:
 . . . .
 . . X .
 . . O .
 # . . .

...    // more output...

X turn:
 . . . .
 . X X X
 . . O O
 # . # .

The winner is: X
```