CSCE 315
Project 2: Changes and Clarifications

(Keep in mind that changes are an expected part of a software development process. In this case, we have some different ideas and more details about exactly what we'd like. The changes in the program should be reflected in your second sprint.)

For the Kalah game, there are a few things that we'd like to have added in:

1) First, we now realize that there are lots of possible variations of Kalah. We'd like to allow the following variations to the game.
    a. Games should be able to be run where the number of holes on each side (not counting the Kalah, or store) vary. You should allow games with at least a range from 4 to 9 in terms of number of holes on each side.
    b. Games should be able to be run where the number of seeds starting in each hole varies. You should allow games with at least a range from 1 to 10 in terms of number of seeds per hole, starting out.
    c. You should allow a random distribution of seeds on each side to begin with. The total number should stay the same (i.e. the product of the number of holes and the number of seeds per hold), but the seeds should be assigned to one of the holes at random. Both players should have the same arrangement of starting seeds on each side (e.g. if Player 1 looks at their board and has seeds numbering 6, 4, 7, 6, 5, 8 from left to right, then looking across the board, player 1 would see 8, 5, 6, 7, 4, 6 from left to right (i.e. when going counterclockwise, the order would be 6,4,7,6,5,8,6,4,7,6,5,8).
2) Second, we'd like to have a "Pie rule" option for the first move. The Pie rule is a way of evening out games where one player has an advantage. It comes from the way that you ensure a fair split if a piece of pie is cut in two: let one person cut the slice, then the other gets to choose which of the two to keep. The way the pie rule works is this:
    a. Player 1 makes a move, as normal.
    b. Player 2 makes a choice:
        i. Option 1: Player 2 lets Player 1's move stand, and then goes ahead and plays as normal
        ii. Option 2: Player 2 chooses to switch positions with Player 1 (i.e. Player 2 essentially becomes Player 1). So, the person who was Player 1 makes another move, this time as Player 2.

    After that first move, the game proceeds as normal – there are no further switches later on.
3) We also want to implement a time limit. That is, you should allow people, if desired, to set a time limit for how soon a move must be made. If a player does not make the move in that time, then that player loses. Time can be specified as a number of milliseconds.
4) We want to allow players to play a client-server system. Details of this will be provided below.

Going forward, you should plan on implementing these changes, including adding items to the backlog as appropriate in order to incorporate differences from the original plan.

Regarding the client-server system, here are some of the details:
You should allow game play between two computers, a computer and a person, or two people.

Communication will be through ASCII text strings passed through sockets. An unused port should be used for the server. You have to be extra cautious with your code since your server can easily become a security hole. Before you print anything that came from the client or the server, you have to first check if these are printable characters, and make sure the length of the input is within your input buffer size. The list of commands are simple:

command	::==	*info*| *move* | *ack*
info	::==	INFO *game_config*
move	::==	*hole* (*hole*)* | P
ack	::==	WELCOME | READY |OK | ILLEGAL | TIME | LOSER | WINNER | TIE
game_config	::==	<int for holes per side> <int for seeds per side> <long int for time> <F | S> <S | R *hole_config*>
hole_config	::==	<int for seeds in hole 1> … <int for seeds in hole h>

Note the following:

- A game configuration consists of a number of holes per side and a number of seeds per hole, followed by a time limit per move, followed by either an "S" or an "R", with "S" indicating a standard starting configuration with equal numbers of seeds per hole, and an "R" indicating a random starting configuration with random numbers of seeds per hole.
- A time limit of 0 should be interpreted as no time limit. You wait indefinitely for a response.
- The hole numbers start with 1 at the player's left and h at the right, where h is the number of holes per side. Holes are ordered from each player's perspective (so the hole opposite player 1's hole 1 is player 2's hole h).
- Report ILLEGAL if an illegal move is made or TIME if the time limit was exceeded. This will end the game, with the person making the illegal move or going overtime being the loser. The server should report TIME or ILLEGAL, after passing the move along. If the client took too long to make a move or made an illegal move, the server should send it a TIME or ILLEGAL response instead of an OK. If the server took too long to make a move, it should send a TIME statement after sending an OK, before sending the move. If the server made an illegal move, it should send the move, then an ILLEGAL acknowledgement (followed by a WINNER acknowledgement).
- The server should be the one to report TIME, ILLEGAL, WELCOME, LOSER, WINNER, or TIE
- The client should be the one to report READY
- A move of "P" is allowed only on Player 2's first move of the game, and indicates a Pie move. That means that player 1 and 2 swap positions, so the next turn is still taken by Player 1, but with Player 2's board.
- The time is a long integer giving the maximum amount of time your routine should take to make a move, in milliseconds. You should use the function `System.currentTimeMillis()` to get timing information in milliseconds. Note that performance will vary depending on the computer used, so your routine may need to adjust to what the current time is.
- Each move by the server or client should be acknowledged by an OK, sent as soon as a move is received and verified. The timer should start from when the OK is received. *The client should not delay in sending the OK in order to have more time!*
- The server should go ahead and send a final move, even if it is illegal, or results in a win/loss/tie before sending the game result.

- The game config is discussed in more detail below.

Your program should choose to act as either server or as client. (Note: it is OK to have your program also allow local games with no client/server connection. You can also split the functionality, so that there is a single server, and separate clients that can run an AI)

- If acting as a server, your program will determine the setup for the game. Your server will allow the client to be one of the two players. For the other player, the server can:
    - Connect to a second client
    - Run its own AI
    - Get input from a human.

  It will need to accept moves from the client and return information to the client.
- If acting as a client, it will need to connect to the server (you will need to let your program read in a host name and port to connect to), get a game configuration, pass to the server, and get back a responding move from the server. Moves should be able to be generated either from the user or from the AI
    - You may create separate clients – one for user input, and a separate one for AI, if you prefer.
    - The client must return a move within the designated time. The server will give a time limit exceeded response (and the other player will win) if this is not done.

When the client connects to the server, it should receive a WELCOME acknowledgement, followed by an INFO statement that gives the game configuration. The client should return READY to acknowledge the game configuration. After this, the server should send BEGIN to the client, and then each side should send moves back and forth. When the game is over, either due to an illegal move, time limit exceeded, a tie, or a win/loss, the server should send a notice that the client is either a WINNER, LOSER, or there was a TIE. Before doing so, an ILLEGAL or TIME acknowledgement should be sent, if applicable.

The info statement will give the game configuration from the perspective of the client:

- The first two numbers are integers giving the number of holes and seeds per hole, respectively.
- Next is the time the client has to move (note: this might be different from the server's time).
- Next is either an F or an S, to denote whether the client will be the First Player (F) or Second Player (S).
- Next is either an S (for a standard configuration with equal seeds in each hole) or R (for a random configuration with differing seeds in each hole. If random, there will be a list of the number of seeds per hole, with one integer per hole (on each side).

Below are a couple of examples.
First, here is a game where the client loses because it takes too long to make its first move.

WELCOME	Acknowledge that the client connected
INFO 4 1 5000 F S	A standard game with 4 holes per side, each with 1 seed, 5 seconds per move and the client goes first
READY	Client acknowledges getting the game info
4 3	Client moves first, from hole 3, but takes too long
TIME	Server tells client it took too long
LOSER	Server notifies the client that the client lost.

In this example, we set up a very small standard game and play a few moves:

WELCOME	Acknowledge that the client connected
INFO 4 1 5000 F S	A standard game with 4 holes per side, each with 1 seed, 5 seconds per move and the client goes first
READY	Client acknowledges getting the game info
4 3	Client moves first, chooses to move from hole 4 (puts one in store), then from hole 3 to 4 (captures opposite piece, so now 3 in store)
OK	Server acknowledges getting client's move
P	The server has chosen a Pie move, meaning they've swapped positions
OK	Client acknowledges getting the move from the server
4 3	Client moves from hole 4 then from hole 3, like before
OK	Server acknowledges getting client's move
2	Sever moves from hole 2
LOSER	Server notifies the client that the client lost.

Starting configuration:

0	1	1	1	1	0
	1	1	1	1	

After Player 1's move. The move from 4 put 1 seed in the store, meaning Player 1 moved again. Moving from 3 put a seed in hole 4, which was empty, thereby capturing the seed opposite:

0	1	1	1	0	3
	1	1	0	0	

Player 2 chose a Pie move, meaning that the board is flipped around and it's Player 1's turn again:

3	0	0	1	1	0
	0	1	1	1	

Player 1 makes the same move as before: 4 then 3. Now it's Player 2's turn:

3	0	0	1	0	3
	0	1	0	0	

Player 2 moves from hole 2. This puts a seed in hole 3, capturing the seed opposite it. Player 2 has won:

5	0	0	0	0	3
	0	0	0	0	

Here is another example:

WELCOME	Acknowledge that the client connected
INFO 6 6 1000 S R 6 4 7 6 5 8	A game with 6 holes per side, each with 6 seeds, 1 second per move and the client goes second. Random configuration with distribution of seeds as shown (see example below)
READY	Client acknowledges getting the game info
3	Server moves first, chooses to move from hole 3 (puts one in store), then from hole 3 to 4 (captures opposite piece, so now 3 in store)
OK	Client acknowledges getting server's move
2 1	Client does not choose a Pie move, instead chooses to move from hole 2, then hole 1
OK	Sever acknowledges getting the move from the client
3	Sever tries to move from hole 3
OK	Client still acknowledges getting server's move (might be sent later, though)
ILLEGAL	Server reports the move as illegal.
WINNER	Server notifies the client that the client won the game.

Starting configuration:

0	8	5	6	7	4	6	0
	6	4	7	6	5	8	

After server's move (server is on North side in this case, playing first). Hole 3 had 7 seeds, and the end distribution is as follows:

1	9	6	7	0	4	6	0
	7	5	8	6	5	8	

Client moves from hole 2. The 5 seeds result in the last falling into the store, and the second move is from hole 1. This yields the following configuration:

1	9	6	7	0	4	7	2
	0	1	10	8	7	10	

The server then tries to move from hole 3. Since hole 3 is empty, this is an illegal move, so the client is declared the winner.