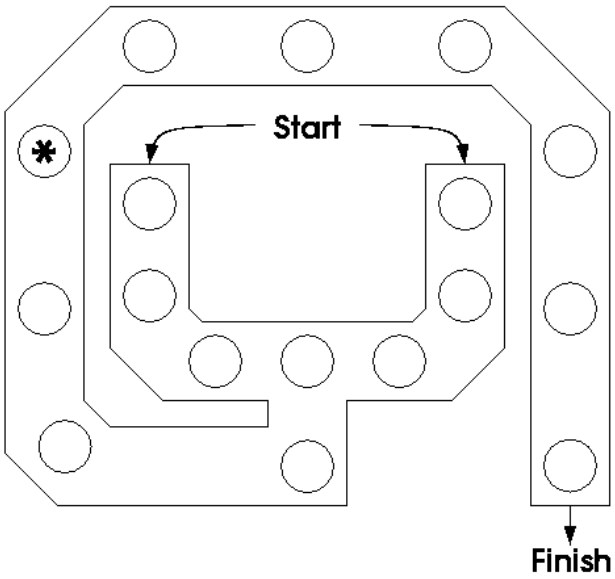
Introduction
You will work as part of a small software development team to develop a game. The game is inspired by Lotus, developed by Dominique Tellier and published by Ravensburger Spieleverlag.

The game
We will implement this as a two-player game. Players alternate their turns. The object of the game is to move your game pieces off the board before your opponent does. The rules of the game are deceptively simple, but developing a good strategy can be hard.

The board
The Lotus game board has two start positions, and one finish position. It also has a special position, marked with a "*", called the trampoline position.



Our gameboard will have this same basic design. It will, however, not have the trampoline position. It will instead have three positions with special properties. Which positions will be special will be determined randomly each time the game is played. The only position on the board which will not have any special properties is the very last position before the finish. The special properties will be described once movement of game pieces has been described.

Game pieces
Player 1 begins with 10 white game tokens. Player 2 begins with 10 black game tokens. Each player's game tokens are arranged in the center of the game board in four stacks. One stack has 4 game tokens, one has 3, one has two and the last one has only 1 token.

Moving game pieces
At the start of the game each stack has game pieces of the same color. Later in the game stacks can have game pieces of different colors. Only the top game piece of a stack can be moved. A piece at the top of a stack of height *n* can be moved *n* spaces on the board.

A player may move their pieces onto the board using either of the start spaces.

Pieces can only be moved forward on the board, never backward, unless the piece is on the position with the "move backwards" special property (to be labeled with a minus sign on the board). A piece moved from this position moves the same number of spaces as for a regular position (i.e. determined by the height of the stack) but it moves backwards rather than forwards.

Each position on the board has a height, which is added to the number of pieces on the position to determine the height of the stack. For all positions except two the height is zero. Two positions have the special property that their height is non-zero. One position has the special property that its height 1, and another has the special property that its height is 2.

If a player can move one of their own game pieces they must do so. If a player cannot move one of their own game pieces they have the option of moving one of the other player's game pieces, but they can elect to not move any piece.

In order to take a piece off the board a player must move the piece past the last space on the board.

Winning
The first player to move all their pieces off the board wins.

Your task
Your team must design and implement this game with a simple user interface. The interface must consist of a graphical view and a Finch-based control. The view only needs to be functional, not pretty - it must present a player with the current configuration of the game, including the board and the pieces on it, an indication of whose turn it is, and, if the game is over, who won. The Finch-based control must indicate whose turn it is by the color of the Finch's beak (so for the purposes of your implementation player 1 will play RED and player 2 will play BLUE). Come up with some creative way of using the Finch sensors (e.g. the accelerometer) to control which game token is going to be moved. The Finch API is here:

http://www.finchrobot.com/javadoc/index.html?edu/cmu/ri/createlab/terk/robot/finch/Finch.html

Submission
You will submit via Web-CAT. Submissions are due by Friday, April 5, on or before 9:00 PM for everyone. Each team will make ONE submission. Decide early on who will be responsible for making your team's submission.

Grading
Your team's project submission will be graded on its merits as follows:

Functionality: 60%
30% is awarded for stage 1 game-logic functionality, and 30% for the user-interface.

Documentation: 10%
You must have a brief user's manual which described how to start and how to play your game (including the commands/rules AS THEY WORK IN YOUR GAME). In other words, if your game does not enforce turn-taking, say so!) Name your user's manual User.txt (it must be a plain text file).

You must also have javadoc comments for each class and each public method which says WHAT the class or method does, not HOW it does it. For methods, the role and expected value for each parameter must be explained, and the return value must be documented too (describe the significance of possible return values).

Testing: 30%
You are expected to have JUnit tests for the functionality of each NON-UI public method you write. Do not write tests for your UI code. We are still not aiming for perfection at this point, but we do expect to see a good suite of sensible tests. (E.g. if you move the top token from a stack of height 3, that token should be removed from its original stack and placed onto the stack three positions further down the board.)

Your overall grade is based on the overall team grade for the submission, and your average peer grade. Basic point: if you do not contribute as much as others on your team you will not receive as much credit as your teammates. More details about how peer evaluation works will be made available soon.

Team meetings
To help you manage your time, here are some do's and don'ts for recitation:
• Do discuss accomplishments (goals met) since last meeting problems (goals not met) that have come up since last meeting goals for next meeting schedule for the next week for pairs to meet to pair program.
• **Do NOT even consider writing code during your recitation time unless you have done the above.** Remember, your recitation time is the only time during the week when you are guaranteed to be able to get together as a team. This time is precious - don't waste it.
• Do have someone take notes each meeting, and keep notes in your code repository. Each team **MUST** submit one set of meeting notes per week. For the first week of stage 1 your repository may not be set up yet. Once repositories are set up, put a copy in your repository, in a folder called "Minutes".
You must also e-mail me a copy of your team meeting notes. The subject line of the e-mail must be

[CSE116-T<xxx>] Minutes from meeting on <DATE>

where <xxx> is the team number, and <DATE> is the date in mm-dd-yyyy format.
• Do compromise.
• Do NOT exclude team members - use all your team's resources. Each team has people with different skill levels and talents. Coding prowess is just one of many skills needed to complete this project.
When pairing, strong and weak coders should work together, and the weaker coder should drive more often than the stronger coder (to help them build their skills with the help of the stronger coder, who navigates for them). Doing this will pay off big-time in stage two of the project! Likewise, not doing this will cost you big-time in stage two!