Introduction
You will continue your work on the game, but from a partial stage 1 solution provided to you in the repository. Your team MUST base its stage 2 solution on this stage 1 implementation. Your team must also add additional functionality as described below.

The game
The game must now support between 2 and 6 players. The number of players is determined by the number of names given as command-line arguments when the game is started. Command-line arguments are supplied to the program through the 'args' parameter of the main method.

The board
We will modify the game board in the following ways. There will be three starting tracks (left, center and right). The game board will three positions with special properties. Which positions will be special will be determined randomly each time the game is played. The special positions can appear anywhere on the board, including the starting tracks and the finish position. The special properties will be described once movement of game pieces has been described.

Game pieces
For a 2-3 player game: Each player begins with 10 game tokens, all of the same color. 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. Player 1 plays red (R), player 2 blue (B), player 3 green (G).

For a 4-6 player game: Each player begins with 6 game tokens, all of the same color. Each player's game tokens are arranged in the center of the game board in three stacks. One stack has 3 game tokens, one has two and the last one has only 1 token. Player 1 plays red (R), player 2 blue (B), player 3 green (G), player 4 white (W), player 5 black (K), and player 6 yellow (Y).

Moving game pieces
Movement rules are the same as for stage 1.

A player may move their pieces onto the board using any 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 on the backwards space) but it moves backwards rather than forwards, except if the height of the stack is 1, in which case the piece moves 1 space forwards. When moving backwards off the common track a piece must move onto the same track it used to enter the board (left, center or right). If it moves off the board into the start stacks it must end up on the same start stack from which it originated. If the piece is N positions from the start of the board and it has to move N+K positions backwards, it is simply pushed onto the correct start stack. The extra "movement potential" does not affect gameplay.

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

is 2, and another has the special property that its offset is 5.

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 two possible user interfaces. Both will use the same view. The view only needs to be functional, not pretty – use the one supplied as a starting point. The view 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.

One of the two possible controls is the Finch-based control of stage 1. The other is a mouse-based control along the lines of what is in the supplied base code (where interaction is via mouse clicks on JButtons).

All players will use the same control, either the finch-based or the mouse-based control. The control to use is indicated by entering either "finch" or "mouse" as the first command line argument. For example, to play a three-player game with Amy, Bob and Chloe using the Finch the command line arguments would be:

finch Amy Bob Chloe

wheras to play a three-player game with Amy, Bob and Chloe using the mouse the command line arguments would be:

mouse Amy Bob Chloe

If there are an inappropriate number of command line arguments, or if some String other than "finch" or "mouse" is used to indicate the control, and error message must be displayed on the console and the program must terminate.

NOTE ABOUT SUPPLIED PARTIAL SOLUTION
The supplied partial solution to stage 1 is just that: partial. Not all required stage 1 functionality is present. You are responsible for adding this functionality. There are no tests written. There are no javadoc comments. And the code is buggy. You must take care of these issues and add the required functionality to the project. You MUST build your stage 2 solution on the supplied partial stage 1 solution. This is, in part, an exercise in code reading, code debugging, and learning to work with unfamiliar code. Keep in mind you know what problem the code is trying to solve – this gives you a significant advantage when looking at the code.

Submission

You will submit via Web-CAT. Submissions are due by Monday, April 29, 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%
15% is awarded for stage 1 game-logic functionality, 15% for stage 2 functionality, 15% for the finch control, 15% for the mouse control.

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!