Assignment 1 part 2: Othello Logic

Value: 8% of your overall grade.

Due date: November 15th, 2020 (Sunday, midnight)

Purpose:

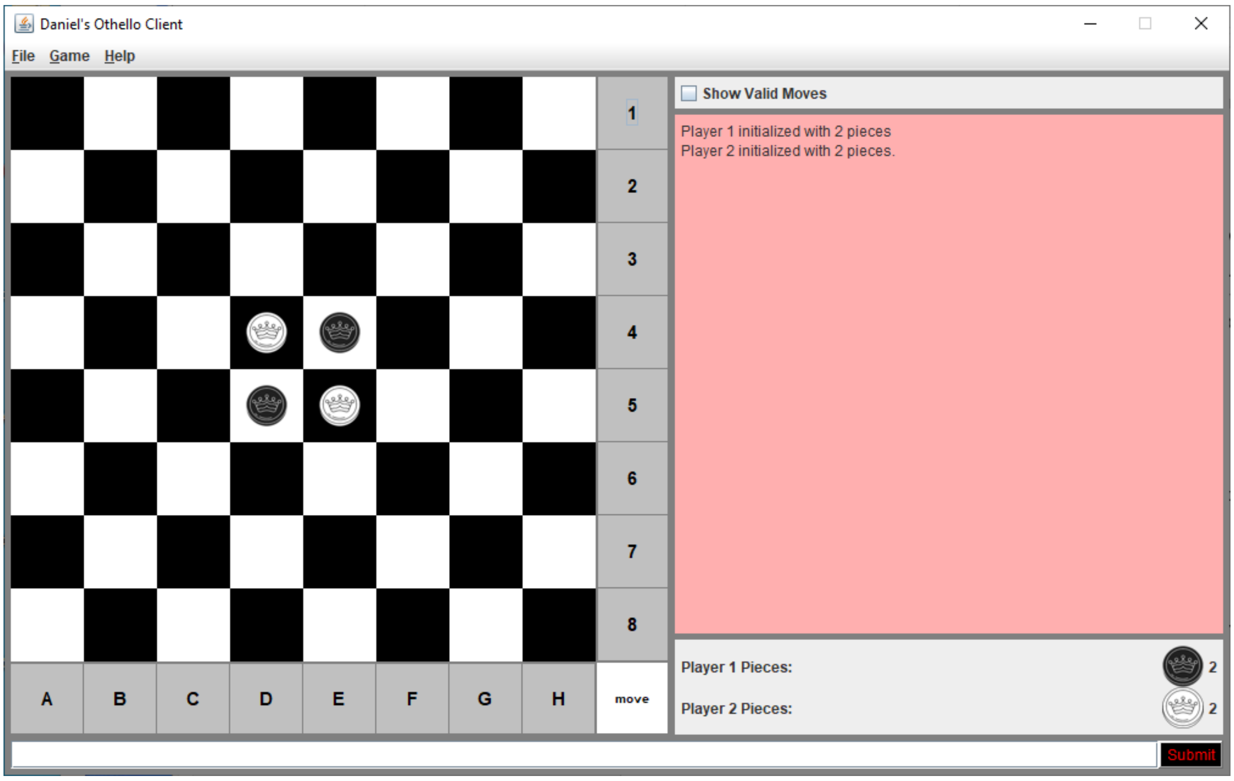
Focus group testing suggests we need to add some extra functionality to the UI before we can finish the implementation. We also need to get a working product out to market.

But seriously:

We're going to implement the game itself. But first, we need to make a few changes to the UI. Fortunately, they're fairly small:

- You will need to add a menu system. Further details on this below.
- You will need to change the "pink zone" label to a text area.
- And you will need to swap the bottom right "score" zone so that black is player 1.

You will also need to implement the game logic itself.



Player 1 initialized with 2 pieces
Player 2 initialized with 2 pieces.

Rules:

Othello (also known as Reversi) is a deceptively simple game of logic and tactics. Players take turns placing a chip of their colour on an empty square. If the piece they place sandwiches an opposing piece between itself and another piece of their own, all sandwiched pieces flip to their colour.

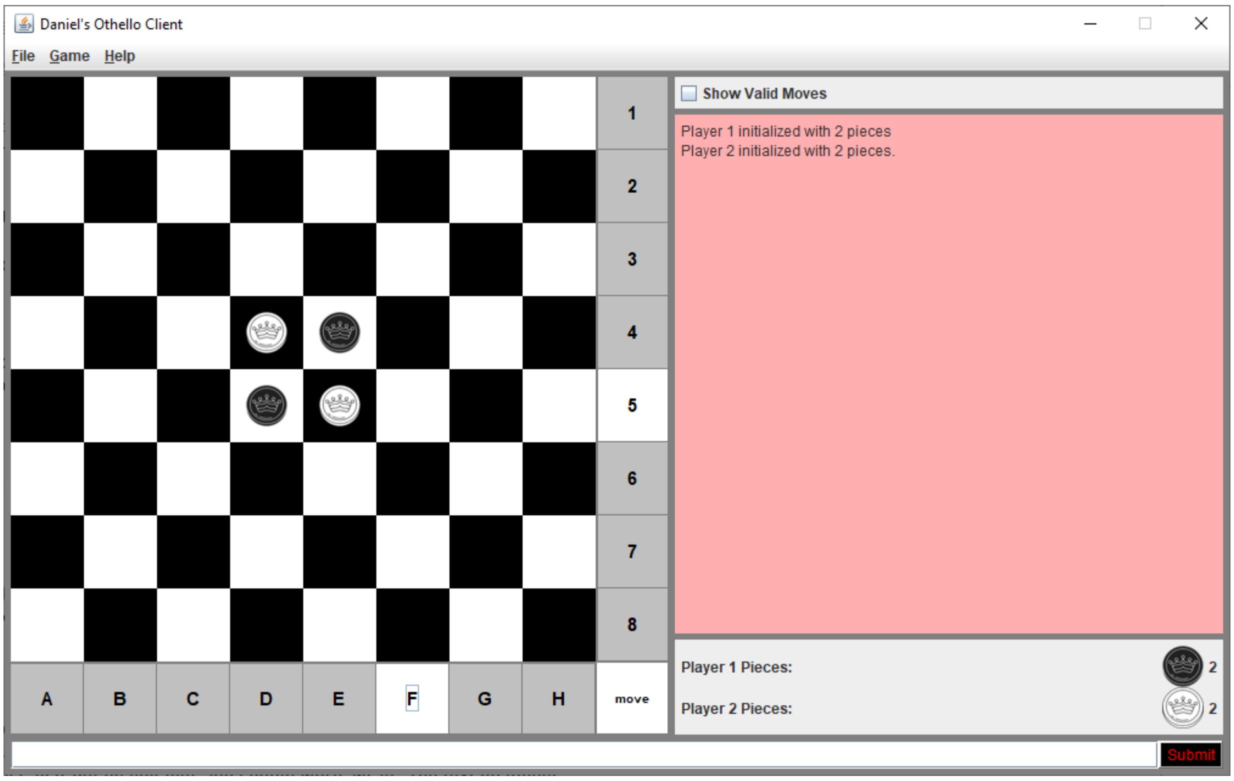
For instance, in the above screenshot on the previous page, black may move to F5. This sandwiches the white piece at E5 between itself and the piece at D5. Captures may be made in all orthogonal (left/right, up/down) and diagonal directions.

A move is valid *only* if it results in one or more capture. If a capture can not be made, then the player must skip their turn. If neither player can make a capture, or the board is full, the game ends immediately. The winner will be the player with the most pieces of their colour.

Black will always be player 1 and go first, in this implementation.

Behaviour

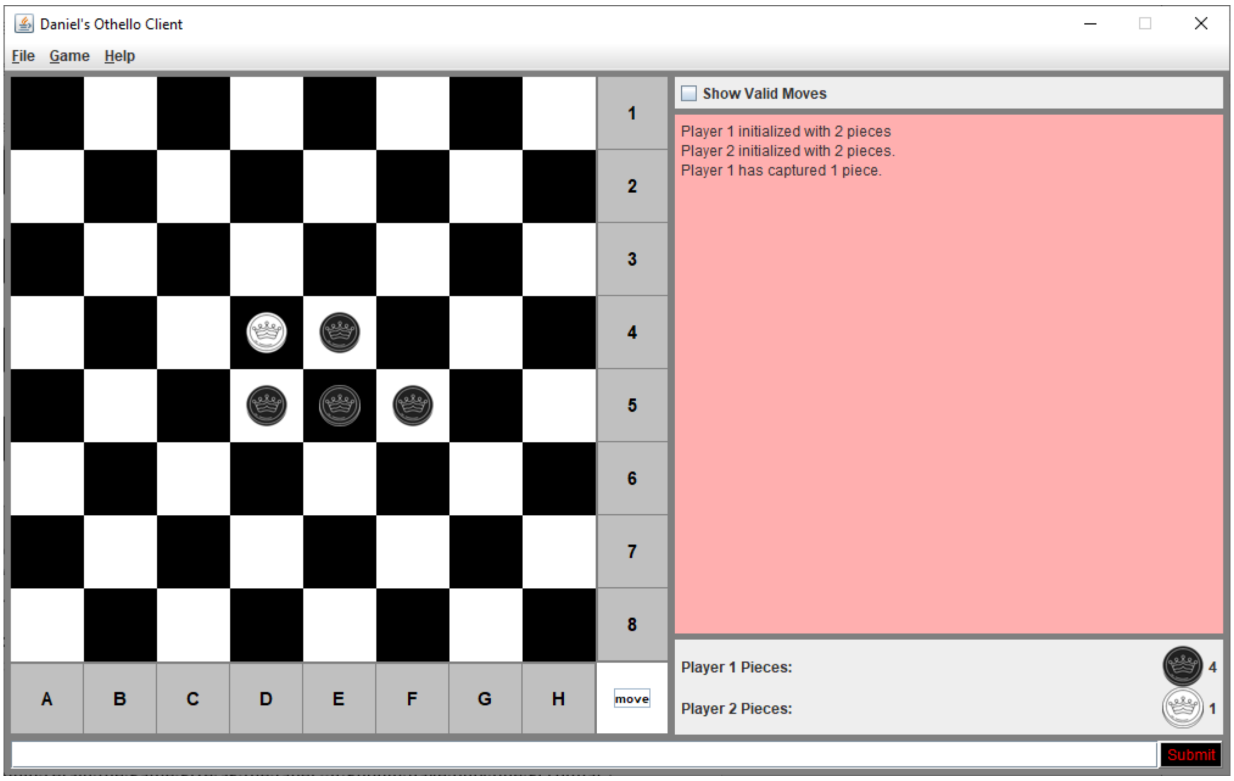
A player may select their move by clicking on a row button and a column button. Buttons that have been pressed must now have a white background. If a player should click a second button in the row or column, the first should revert to light gray, and the new button should light up. (ie: after I click 5, if I click 4, 4 should be white, and 5 should be gray.)



Player 1 initialized with 2 pieces
Player 2 initialized with 2 pieces.

(F5 is selected in this screenshot.)

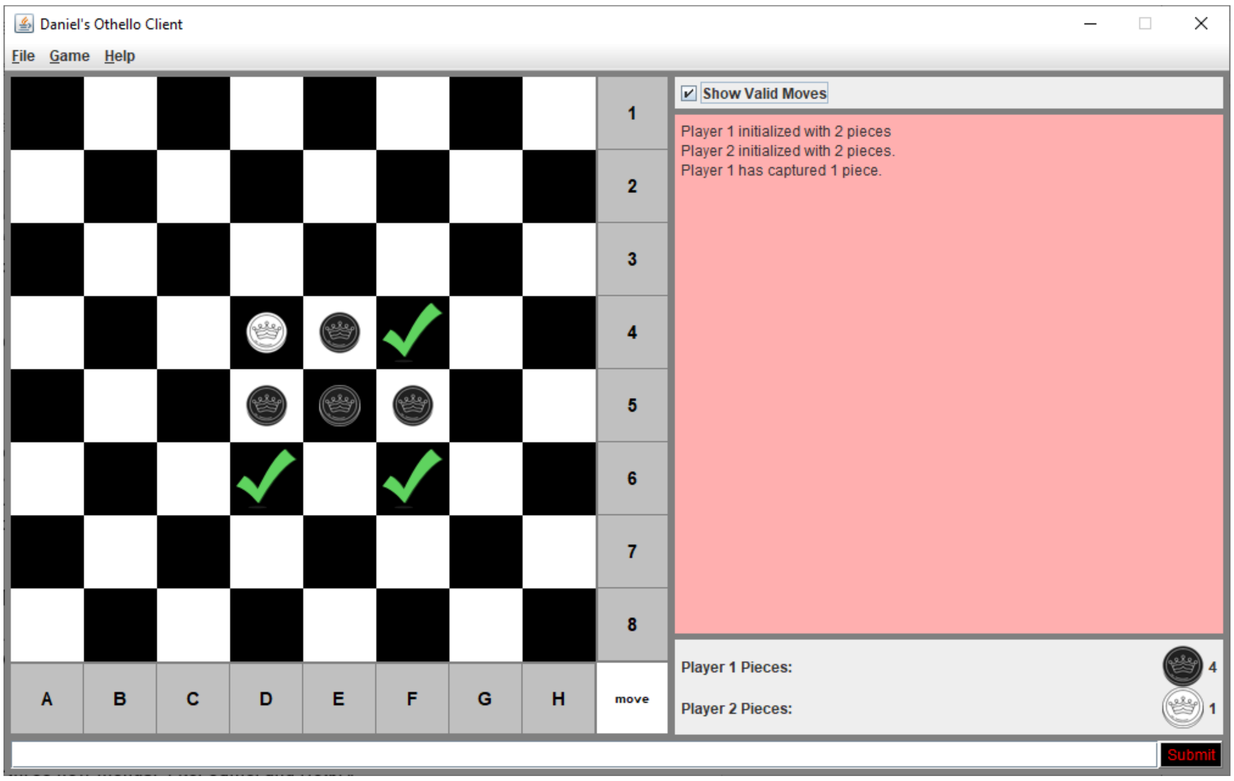
The “move” button will only accept the input when both a row button and a column button are selected. If the move is not valid, nothing happens. If the move is valid, a text message should appear in the output detailing how many captures were made, the board should update, and the scores should update.



```
Player 1 initialized with 2 pieces
Player 2 initialized with 2 pieces.
Player 1 has captured 1 piece.
```

(the end result of having moved to F5. Now it’s white’s turn.)

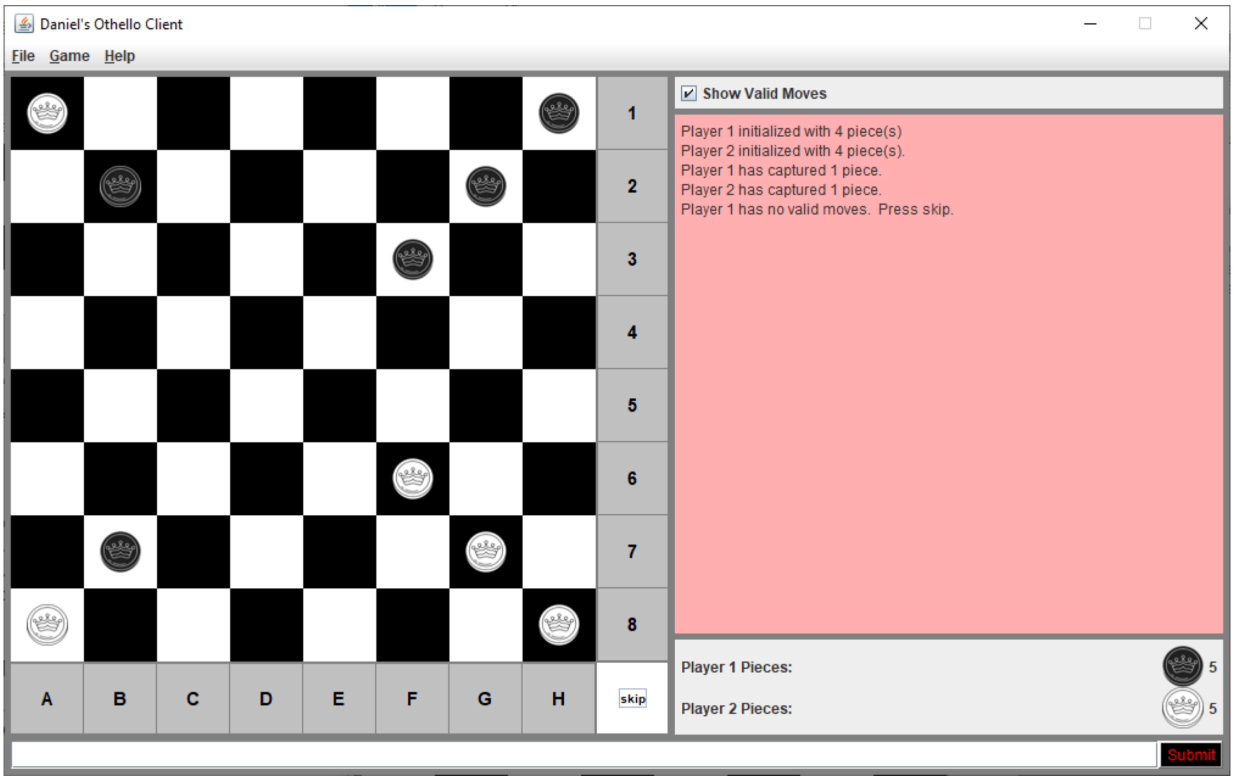
If the “show Valid moves” checkbox is selected, all valid areas to move will be shown with a green checkmark (which is provided). Deselecting this option should immediately make the checkmarks go away.



Player 1 initialized with 2 pieces
Player 2 initialized with 2 pieces.
Player 1 has captured 1 piece.

(these are the valid moves that white can make—note "Show Valid Moves" is selected.)

If a player can not make a move, they should be informed in the output area, and the move button should be renamed to "skip":

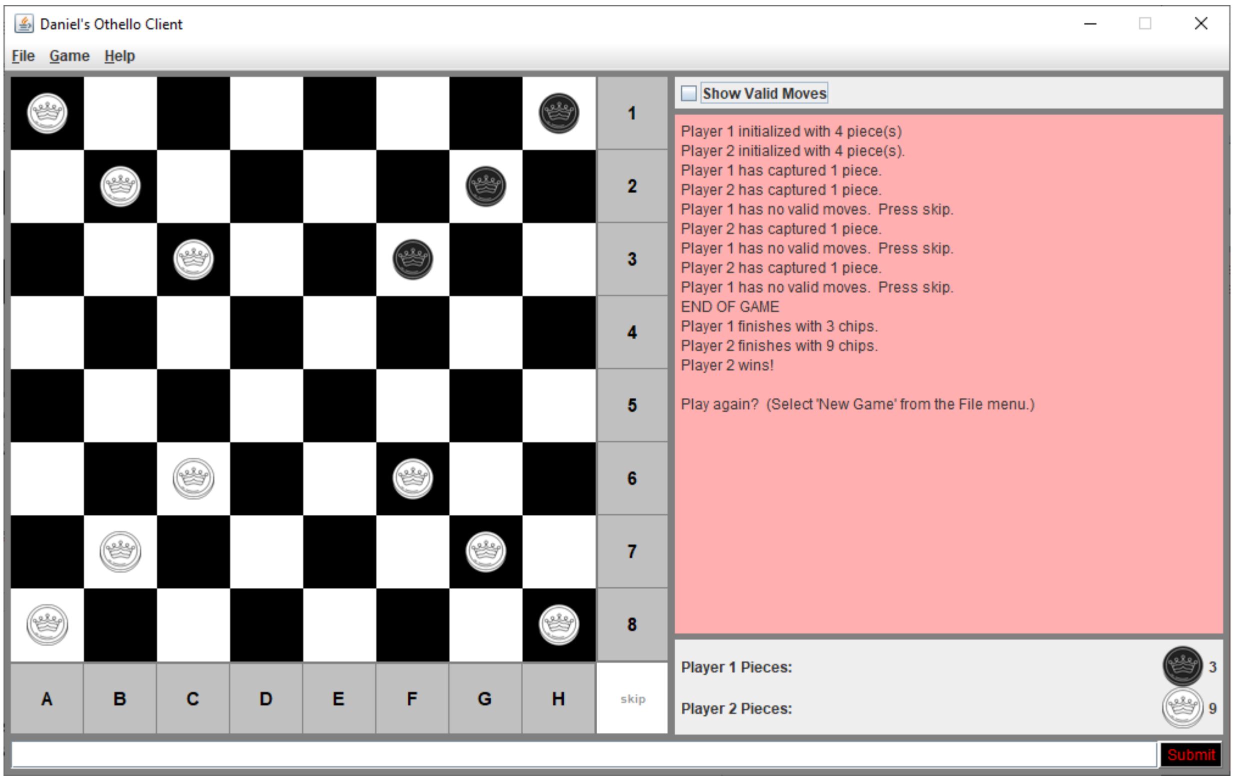


Player 1 initialized with 4 piece(s)
Player 2 initialized with 4 piece(s).
Player 1 has captured 1 piece.
Player 2 has captured 1 piece.
Player 1 has no valid moves. Press skip.

(in this scenario, black has no valid moves.)

Once skip is pressed, if the other player DOES have a valid move, the skip button should become "move" again.

If neither player has any valid moves, the game should end. Tally up the chips, declare a winner, and disable the "move/skip" button until a new game is selected.



```
Player 1 initialized with 4 piece(s)
Player 2 initialized with 4 piece(s).
Player 1 has captured 1 piece.
Player 2 has captured 1 piece.
Player 1 has no valid moves.  Press skip.
Player 2 has captured 1 piece.
Player 1 has no valid moves.  Press skip.
Player 2 has captured 1 piece.
Player 1 has no valid moves.  Press skip.
END OF GAME
Player 1 finishes with 3 chips.
Player 2 finishes with 9 chips.
Player 2 wins!

Play again?  (Select 'New Game' from the File menu.)
```

Player 1 Pieces: 3

Player 2 Pieces: 9

(It's game over here. White won this situation totally engineered situation.)

UI CHANGES:

The Text Area:

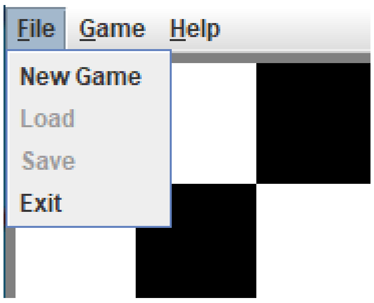
The text area should retain the same size as the label it's replacing. It should have only one scrollbar, a vertical one that appears *only when needed*. There should be no horizontal scrollbar. The text area must also *not be editable*, and should word-wrap. The text no longer needs to be centered.

The score zone

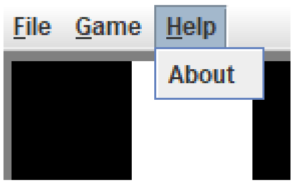
In Othello, the black player goes first. The bottom right corner should reflect this. Black will be player 1, white will be player 2.

The menu.

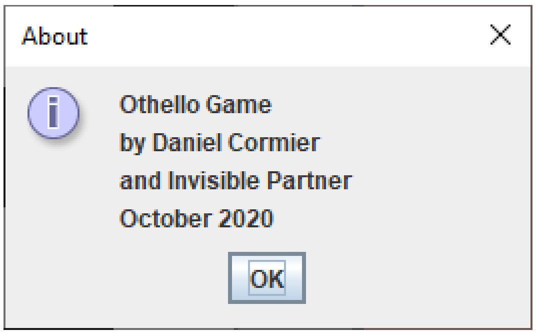
There should be three new menus: File, Game, and Help.



The file menu contains “New game”, “load,” “save,” and “exit” options. Only New Game and Exit should be active. New Game will reset the board entirely based on the “debug scenario” (see below), and Exit will immediately terminate the program.



The Help menu contains only one option: “About”. Selecting this should make a new dialog appear (see Hybrid 5):

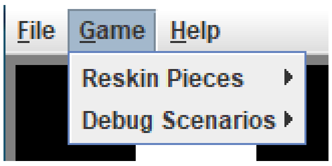


About

Othello Game
by Daniel Cormier
and Invisible Partner
October 2020

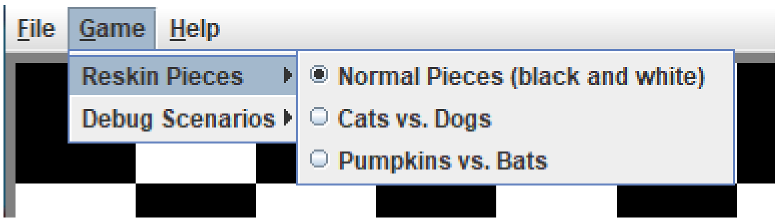
OK

It should appear much as this does, except that it should contain your name and that of your partner if applicable. It should have “About” in the title.

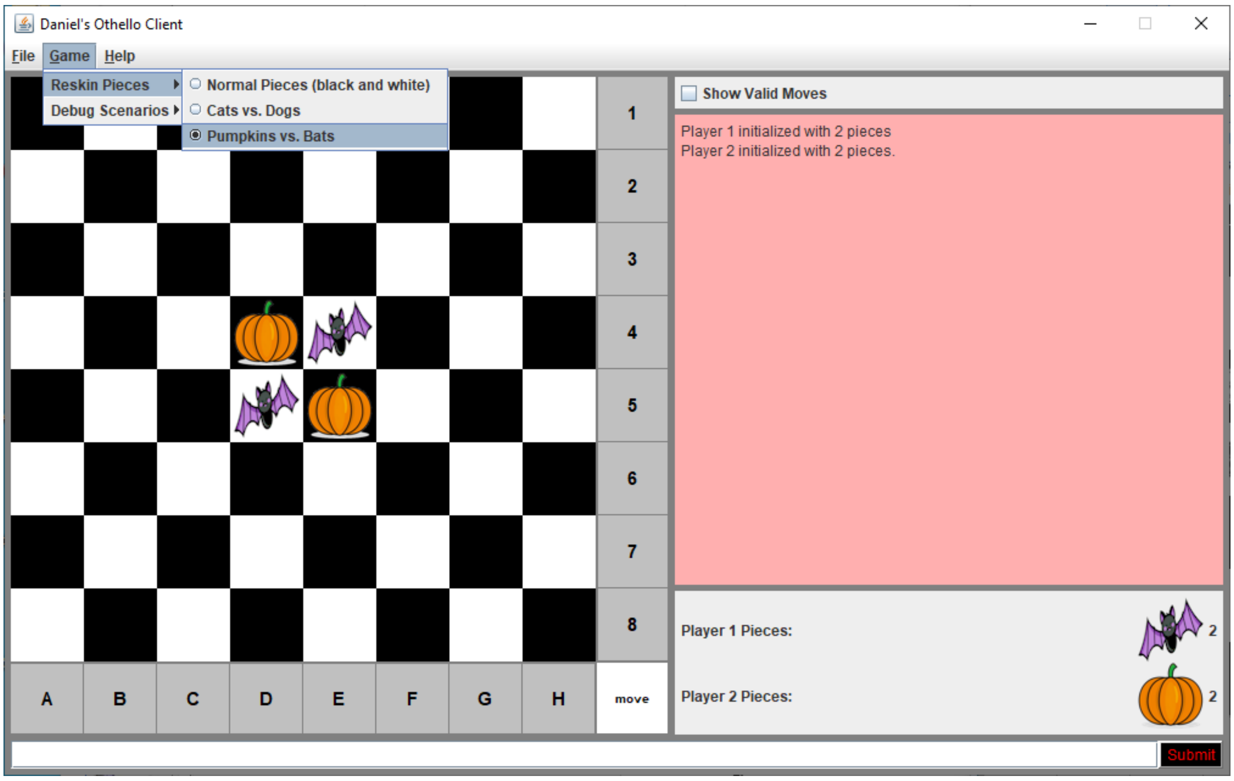


The game menu should contain two submenus, “reskin pieces” and “debug scenarios”. See the end of this document for debug scenarios.

Reskin pieces will allow the user to swap pieces (graphics will be provided):



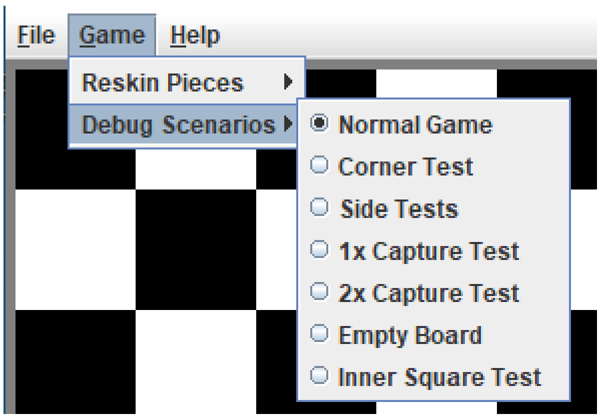
When a new set of pieces is selected, this should take effect immediately upon clicking:



Player 1 initialized with 2 pieces
Player 2 initialized with 2 pieces.

Player 1 Pieces: 2
Player 2 Pieces: 2

(Player 1 is now batty and player 2 is good to gourd.)



The other menu works in tandem with New Game. You may select a board layout (these will be detailed at the end of this document). These changes will only take effect when New Game is selected.

Code:

In this assignment, you will be adding a few methods to OthelloViewController, to the inner class Controller, and creating a new class, OthelloModel.

OthelloViewController:
You will need a method that updates the board when it's called. It will rename the "move" button to "skip" if need be. It will read an instance of OthelloModel and update the pieces on the board after every move, as well as updating the score board in the bottom right. It will also display checkmarks if "Show Valid Moves" is selected.

You may add extra methods to OthelloViewController to make UI changes, as you see fit—bearing in mind that this is MVC-style programming, and so only "view"-related code should go here.

OthelloModel
OthelloModel should contain one 8x8 two-dimensional array that represents the board. In that array, 0 represents an empty square, 1 is for black, and 2 is for white. It must have the following methods:

public int getBoard(int x, int y)
returns the contents of a given square (0 for empty, 1 for black, 2 for white)

public void initialize(int mode)
This contains several different board layouts you can use to test portions of your code. See the end of this document for the layouts I will be using to test yours.

public boolean isValid(int x, int y, int player)
This logic will check to see if a player (1 for black, 2 for white) may make a valid move at that particular square. True if yes, false if no.

public int move (int x, int y, int player)

The player will attempt to move here. Returns the number of chips captured; 0 if the move is illegal. It will also update its internal board model if the move is legal, flipping all appropriate chips to the new colour.

public boolean canMove(int player)

Returns true if the given player has a valid move they can do at all, anywhere on the board.

public int getChips(int player)

Returns the total number of pieces the specified player has on the board at the time of calling.

Controller

The Controller will be reading the user input and calling appropriate methods in ViewController to make adjustments to the UI. It will, if it determines that a player has made a move, call appropriate methods in the OthelloModel, and return the results to the View. It may make small changes to the UI and send output to the text area.

What to submit:

As per assignment submission guidelines, submit one zip file containing all the .class files, all the .java files, and all the image files. (The JavaFX support files are also acceptable).

All code must have the required headers, Javadoc comments, and be adequately commented as well. It should be neat and free of commented-out or still-active debug code.

Submit this zip file on Brightspace on or before the duedate.

Your code will be run from the command line, and I highly suggest you do the same. It should rely on no external libraries.

Final advice:

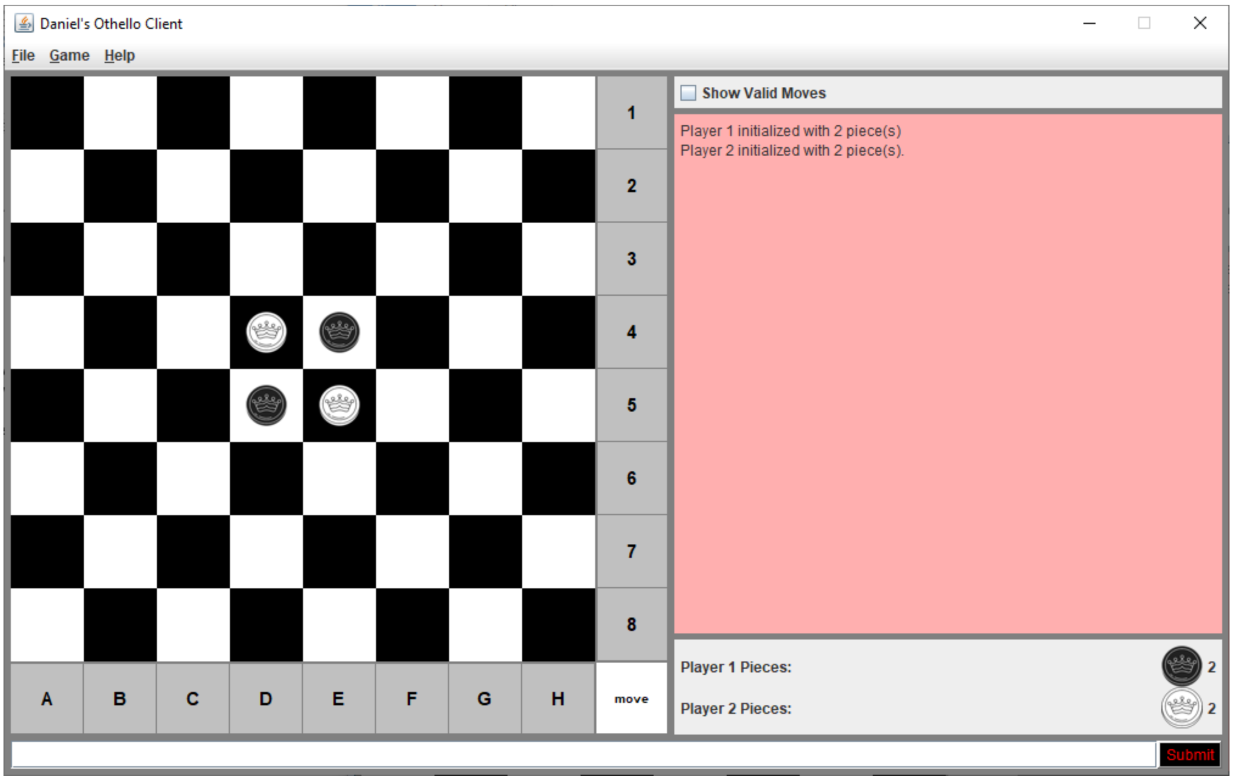
The code is well within your abilities. There is, however, a fair bit of it. Get started on this early and leave ample time for debugging, and you will do well. Try to avoid the Crunch where possible, because that way lies madness and stress.

I will be using the screenshots in the appendix to test your code. You should use them too to ensure your code doesn't throw exceptions or errors.

"Adding manpower to a late software project, makes it later." Frederick P. Brooks Jr., *The Mythical Man-Month: Essays on Software Engineering*

Appendix: My testing scenarios:

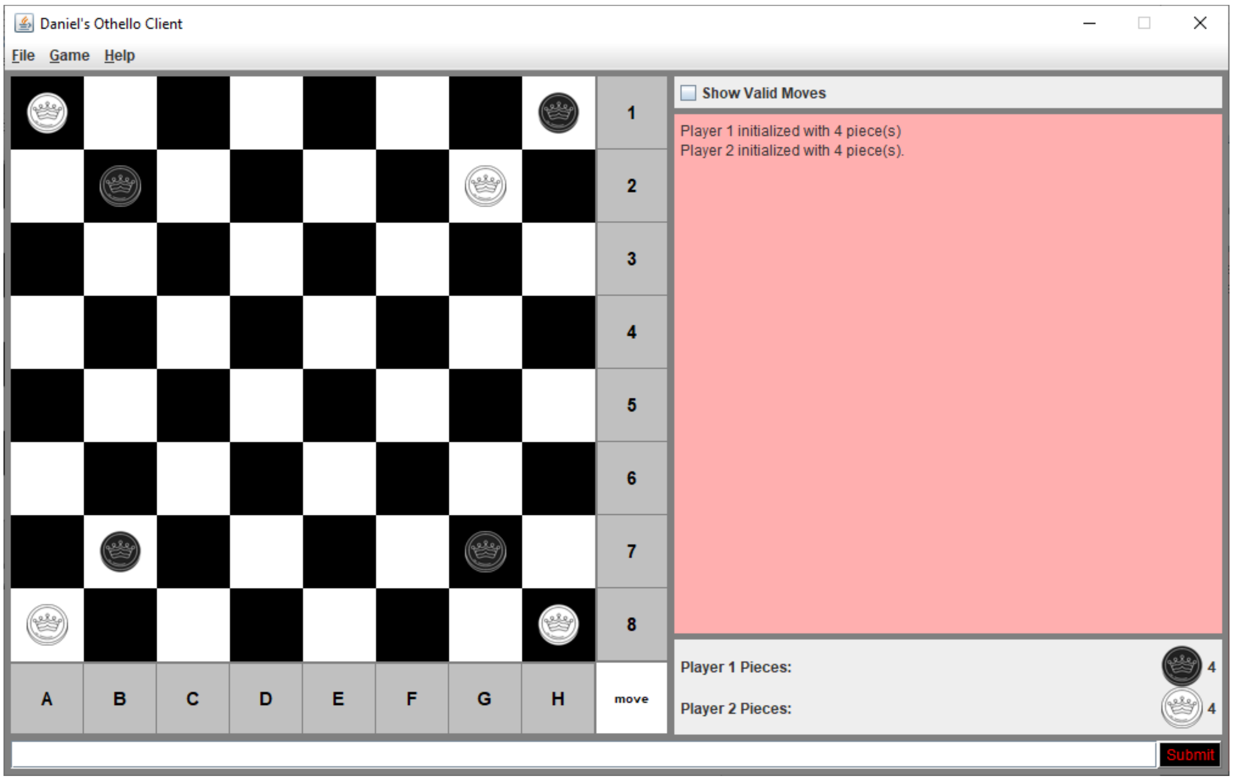
Scenario 0: The Normal Game:



Player 1 initialized with 2 piece(s)
Player 2 initialized with 2 piece(s).

Player 1 Pieces: 2
Player 2 Pieces: 2

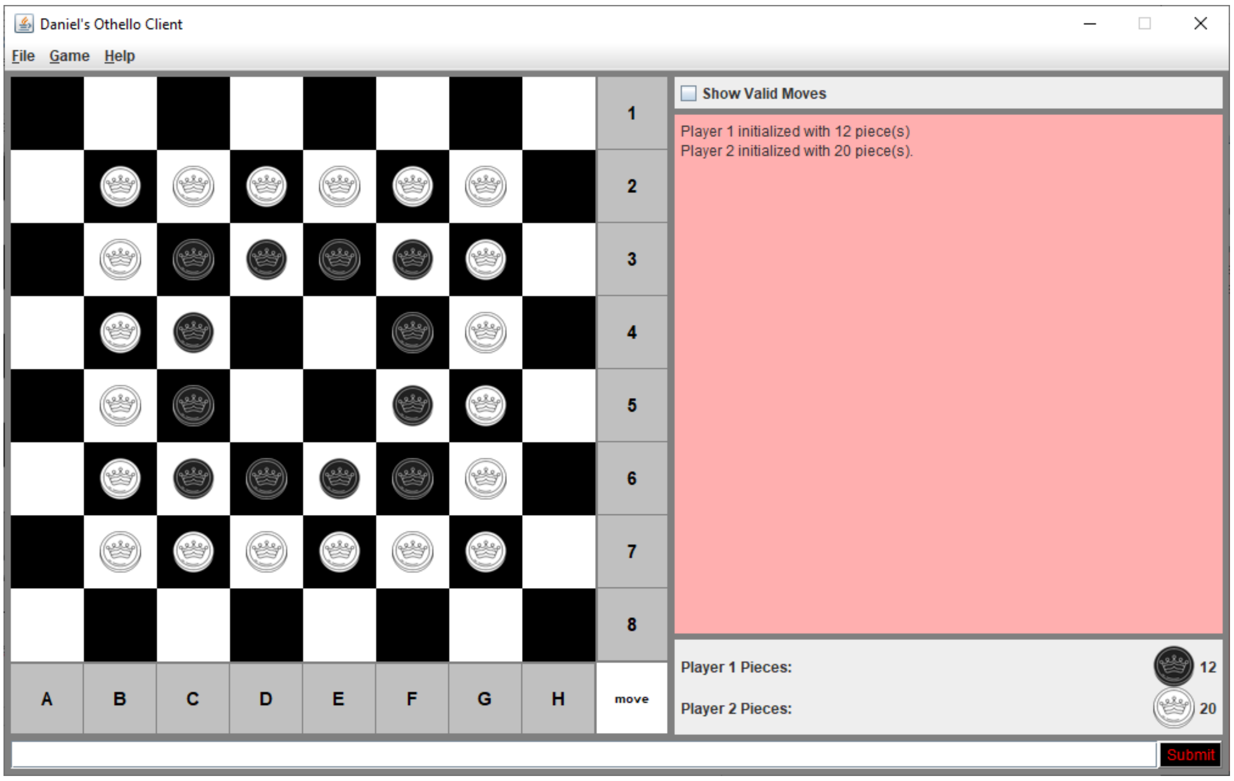
Scenario 1: Corner Test (or a quick game to test systems)



Player 1 initialized with 4 piece(s)
Player 2 initialized with 4 piece(s).

Player 1 Pieces: 4
Player 2 Pieces: 4

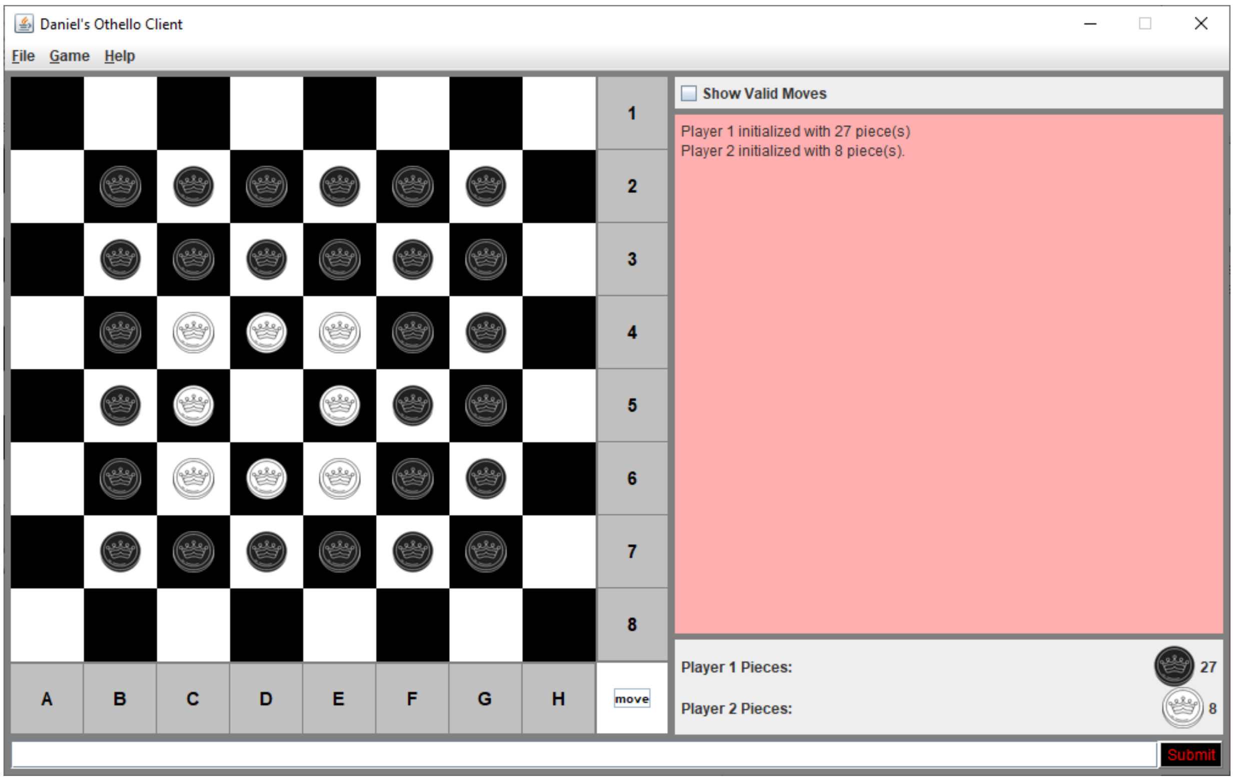
Scenario 2: Side Tests (for testing captures on the corners and edges)



Player 1 initialized with 12 piece(s)
Player 2 initialized with 20 piece(s).

Player 1 Pieces: 12
Player 2 Pieces: 20

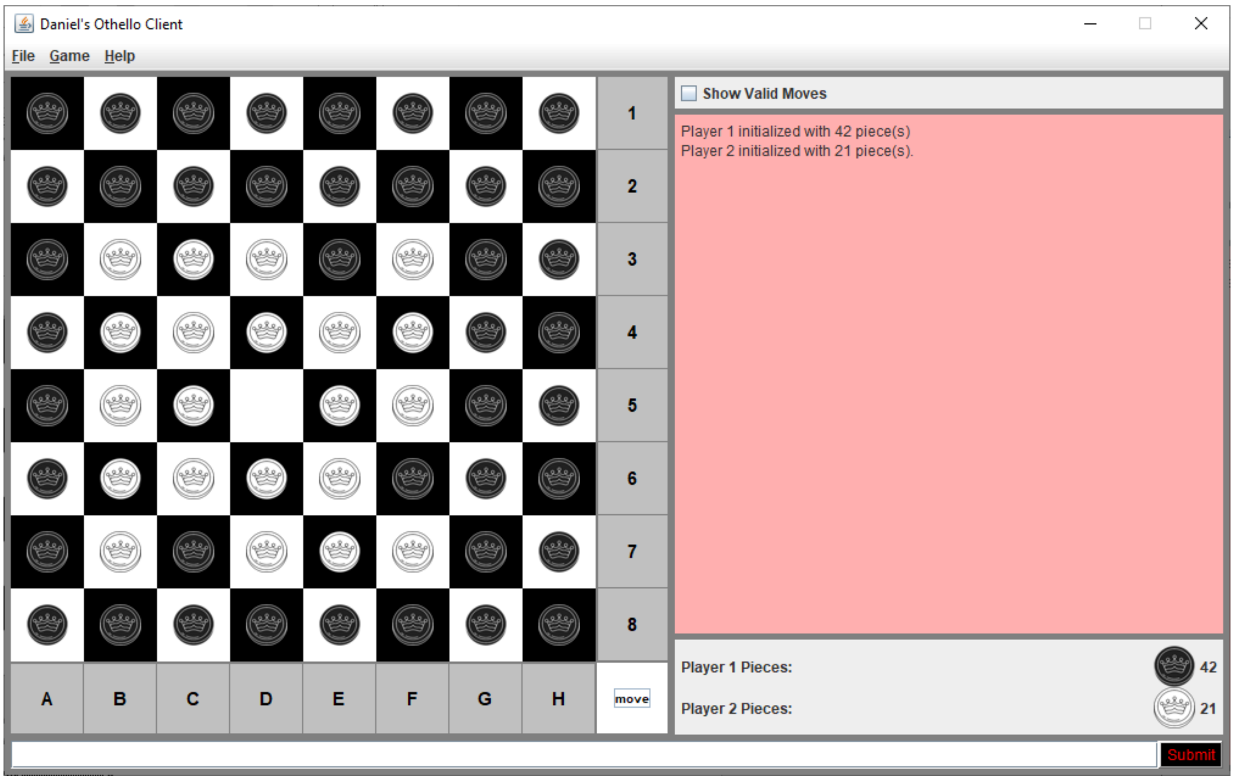
Scenario 3: 1x Capture Test (Black has one move, at D5, to capture 1 piece in all eight directions.)



Player 1 initialized with 27 piece(s)
Player 2 initialized with 8 piece(s).

Player 1 Pieces: 27
Player 2 Pieces: 8

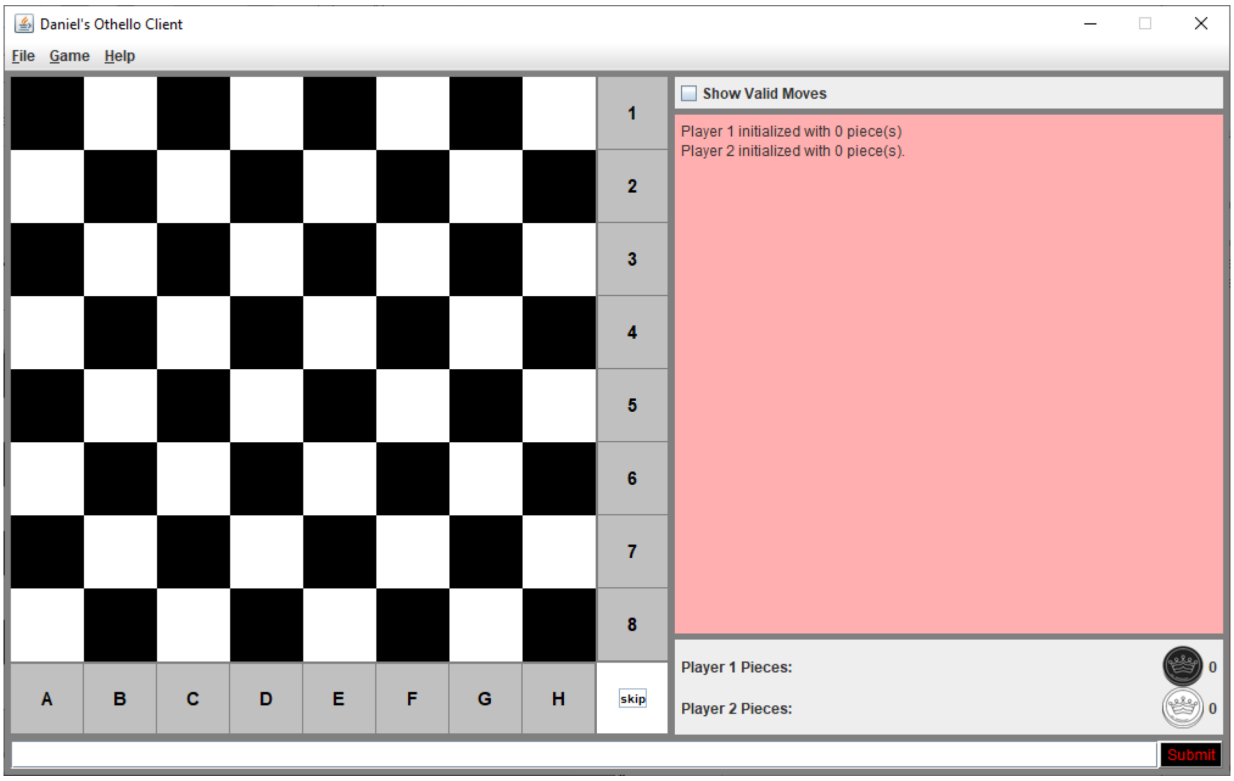
Scenario 4: 2x Capture Test (Black has one move that should result in a total of 16 captures).



Player 1 initialized with 42 piece(s)
Player 2 initialized with 21 piece(s).

Player 1 Pieces: 42
Player 2 Pieces: 21

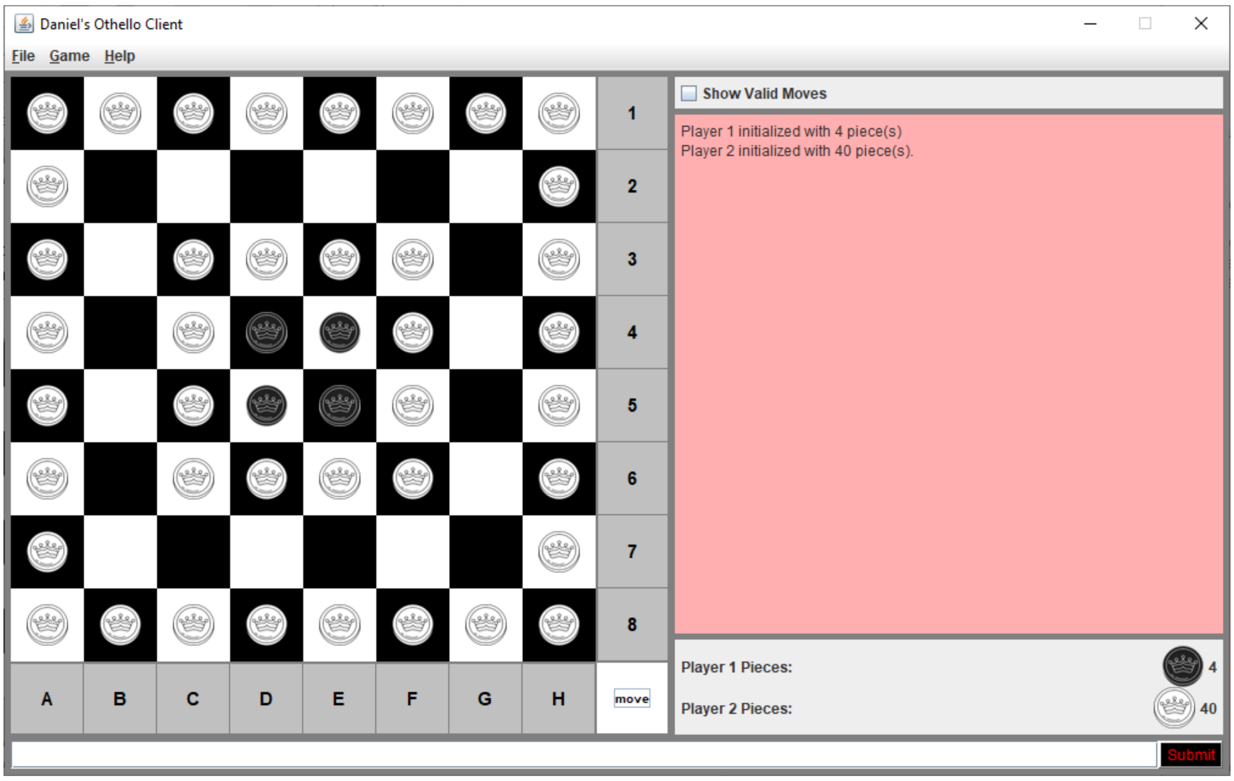
Scenario 5: Empty Board. (The board is empty! "move" should become "skip" right away and the game should end as soon as it's pressed.)



Player 1 initialized with 0 piece(s)
Player 2 initialized with 0 piece(s).

Player 1 Pieces: 0
Player 2 Pieces: 0

Finally, Scenario 6: Inner Square Test. Black can make a valid move in any of those empty squares. Only the inner white pieces should be capturable.



Player 1 initialized with 4 piece(s)
Player 2 initialized with 40 piece(s).

Player 1 Pieces: 4
Player 2 Pieces: 40