Project Part 3

Who: Ryan Baten, Andrew Huang, Patrick Severy, Peter Delevoryas
Title: Double Jump
Vision: An easy to use, no installation web based game that allows people to play checkers with their friends.

Automated Tests:
We made our own testing framework and module for automated tests!
They are designed to test the game logic in the game, including move validation, move execution, board state, game simulation, turn-taking, etc. This means we do not have automated tests for mouse clicks, or animations. We believe User Acceptance Tests are better suited to determining the correctness of these aspects of our game.

STEPS TO RUN OUR TESTS:

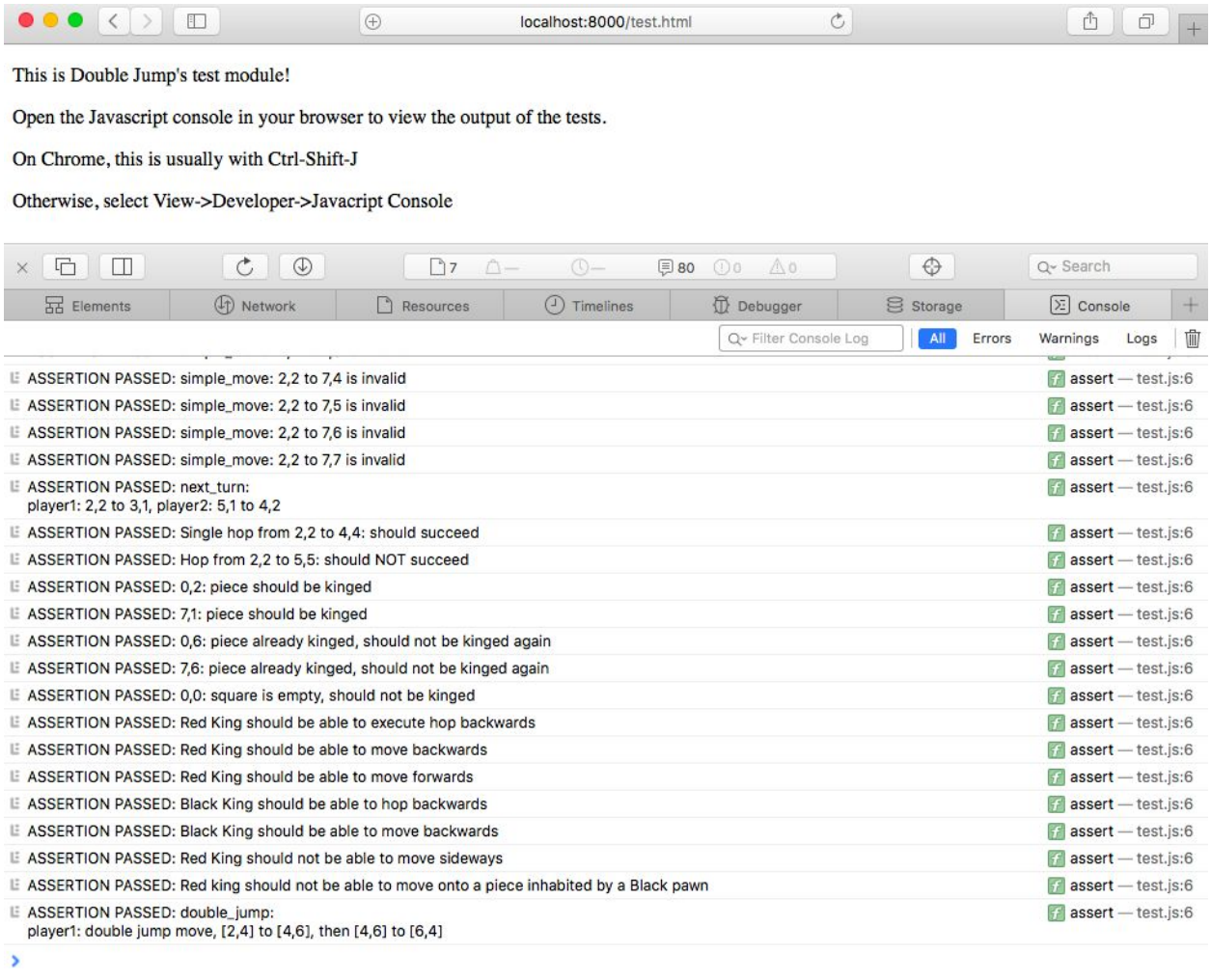
1. Clone the **master branch with this link:** https://github.com/psevery/csci3308.git
2. On the command line, **change directories** to "**csci3308/DoubleJump/public**"
3. Start a python file server with "**python -m SimpleHTTPServer 8000**" (assuming python version 2).
4. In either Chrome, Firefox, or Safari, visit "**localhost:8000/test.html**"
5. Open the Javascript Console (in **Chrome**, this is in **View->Developer->Javascript Console**)
6. You should see the output from running the test suite in the console!

Currently, all of our automated tests are passing (As indicated by ASSERTION PASSED).
If a test is failing, the test will print ASSERTION FAILED in red-colored text.
In both cases, a test prints a description string associated with the test (passed as the second parameter to assert()), describing what was tested. Some tests assert something should be invalid: for example, a pawn moving backwards should be invalid, so we would (and do) test this as:

"assert(**!** game.valid_move(backwards_move_input), "Pawn moving backwards is invalid")"

The code for our test framework and test suite is in "csci3308/DoubleJump/public/**tests/test.js**". To test our game, we made a function that simulates the game with a predetermined set of inputs. So, you give the function the initial game state, a set of inputs (corresponding to piece moves), and then the game is run with those inputs, advancing the game state. The final game state is then checked, to make sure the game is producing the correct output, either by moving pieces on the board, removing them after a hop, or simply ignoring a move if it's invalid.

You can see a screenshot of part of our test output below!


localhost:8000/test.html
This is Double Jump's test module!
Open the Javascript console in your browser to view the output of the tests.
On Chrome, this is usually with Ctrl-Shift-J
Otherwise, select View->Developer->Javacript Console
Search
Elements
Network
Resources
Timelines
Debugger
Storage
Console
Filter Console Log
All
Errors
Warnings
Logs
ASSERTION PASSED: simple_move: 2,2 to 7,4 is invalid
ASSERTION PASSED: simple_move: 2,2 to 7,5 is invalid
ASSERTION PASSED: simple_move: 2,2 to 7,6 is invalid
ASSERTION PASSED: simple_move: 2,2 to 7,7 is invalid
ASSERTION PASSED: next_turn:
player1: 2,2 to 3,1, player2: 5,1 to 4,2
ASSERTION PASSED: Single hop from 2,2 to 4,4: should succeed
ASSERTION PASSED: Hop from 2,2 to 5,5: should NOT succeed
ASSERTION PASSED: 0,2: piece should be kinged
ASSERTION PASSED: 7,1: piece should be kinged
ASSERTION PASSED: 0,6: piece already kinged, should not be kinged again
ASSERTION PASSED: 7,6: piece already kinged, should not be kinged again
ASSERTION PASSED: 0,0: square is empty, should not be kinged
ASSERTION PASSED: Red King should be able to execute hop backwards
ASSERTION PASSED: Red King should be able to move backwards
ASSERTION PASSED: Red King should be able to move forwards
ASSERTION PASSED: Black King should be able to hop backwards
ASSERTION PASSED: Black King should be able to move backwards
ASSERTION PASSED: Red King should not be able to move sideways
ASSERTION PASSED: Red king should not be able to move onto a piece inhabited by a Black pawn
ASSERTION PASSED: double_jump:
player1: double jump move, [2,4] to [4,6], then [4,6] to [6,4]
assert — test.js:6

User Acceptance Tests:
TO COMPLETE THESE TESTS, RUN A FILE SERVER (AS DESCRIBED ABOVE, IN THE AUTOMATED TESTS SECTION), AND VISIT "localhost:8000", THEN SELECT "Local Play". From there, you can execute these tests yourself!

Basic Move						
Test Case ID: UT-01			**Test Designed By:** Andrew Huang			
Test Priority: High			**Test Designed Date:** 11 November 2015			
Test Module: Game			**Test Executed By:**			
Test Title: Basic Moves			**Test Executed Date:**			
Description: Test the functionality of a basic diagonal move on the board.						
Pre-Conditions: Game must be loaded and on the board.						
Dependencies: Game must display on webpage and user must be able to click on the piece.						
Step:	**Test Steps:**	**Test Data:**	**Expected Result:**	**Actual Result:**	**Pass/Fail:**	**Notes:**
1	Click on any desired black piece.	None	Nothing visible should occur.	Actual Result is Expected Result	Pass	
2	Click on a valid destination location.	None	Piece that was selected should move to the new location.	Actual Result is Expected Result	Pass	
3	Click on again on a black piece and try to move it.	None	Nothing should happen.	Actual Result is Expected Result	Pass	After black makes a move, they should not be allowed to make a move until red finishes making a move.
4	Click on a red piece and try to move to any invalid area.	None	Nothing should happen.	Actual Result is Expected Result	Pass	
5	Click on a red piece and make a valid move.	None	The red piece should move.	Actual Result is Expected Result	Pass	
Post-Conditions: Pieces moved by the player should be in new positions.						

Double Jump						
Test Case ID: UT-02			**Test Designed By:** Patrick Severy			
Test Priority: High			**Test Designed Date:** 8 November 2015			
Test Module: Game			**Test Executed By:** Patrick Severy			
Test Title: Double Jump Pieces			**Test Executed Date:** 8 November 2015			
Description: This is a user test to see if “double jumping” pieces works correctly.						
Pre-Conditions: The game must be loaded in a browser. This is a test for same-screen play.						
Dependencies: Hopping a single piece must work.						
Step:	**Test Steps:**	**Test Data:**	**Expected Result:**	**Actual Result:**	**Pass/Fail :**	**Notes:**
1	Move pieces so that one player is in a position to “double jump” 2 pieces of the opposite color.	none	The pieces move across the board correctly.	I was able to move pieces to set up red team for a double jump on black team.	Pass	
2	Hop the first piece	none	The piece that was hopped is removed from the board.	The piece that I hopped is gone and the score for black decremented.	Pass	
3	Select the same piece you just moved to hop the next opponent piece.	none	The piece is selected.	I clicked on the piece and believe it to be selected. The gui does not convey this information.	Unsure	The gui should convey which piece is selected.
4	Move the piece over the next opponent.	none	The piece hops the opponent and removes them.	The piece did not move.	Fail.	
Post-Conditions: Both hopped pieces are removed from the board. The scoreboard reflects the removal of the pieces.						

Pieces Get Kinged						
Test Case ID: UT-03			**Test Designed By: Peter Delevoryas**			
Test Priority: High			**Test Designed Date:** 11/11/2015			
Test Module: Game			**Test Executed By:** Peter Delevoryas			
Test Title: Pieces Get Kinged			**Test Executed Date:** 11/11/2015			
Description: If a pawn reaches the opposite side, it should be Kinged: the piece's appearance should change, and be capable of moving backwards.						
Pre-Conditions: The game should be loaded in the browser, and a player should advance at least one piece to the opposite side of the board, removing other pieces that are in the way.						
Dependencies: Simple and Hop Moves: Otherwise, we can't be sure a king piece can move backwards/hop backwards.						
Step:	**Test Steps:**	**Test Data:**	**Expected Result:**	**Actual Result:**	**Pass/Fail :**	**Notes:**
1	Move a black (or red) piece to a square in the row before the opposing side's home row.	None	Clicking the piece across the board should work (of course, executing moves for the opposite side as well)	Actual Result is the Expected Result	Pass	
2	Move black (or red) piece to a square in the opposing side's home row.	None	Piece should successfully move, and it's appearance should change (a yellow plus sign should be on it).	Actual Result is the Expected Result	Pass	
3	Move black (or red) piece that was kinged backwards left and right (diagonally)	None	Unlike regular pawn pieces, the Kinged piece should be capable of moving backwards (and forwards) diagonally.	Actual Result is the Expected Result	Pass	
4	Use kinged piece to hop other pieces, backwards or forwards.	None	The king piece should also be able to hop pawn pieces, in forwards or backwards directions.	Actual Result is the Expected Result	Pass	
Post-Conditions: One or more pieces should be King pieces, and pieces hopped by Kings have been removed.						

VCS: GitHub https://github.com/psevery/csci3308