Test Case Document

BubbleChess v1.0

Steve Calabro, Mark Koh, Alex Mann, and Eric Most

1/20/15

Revision History

Name	Date	Change purpose	Version
Steve Calabro, Mark Koh, Alex Mann, Eric Most	2/26/2015	Initial version	1.0

Table of Contents

1. Test Cases

ID	Req.	Desc.	Test Steps	Expected Results	Actual Results	Comments
0	1.1.1	The client application will pass the requested username and password to the server to authenticate it and add it to the database	1. Receive requested login info from GUI 2. Pass data to server	Server response “success” for valid info or “failure” for invalid info		Object returned will be a JSON object. Invalid info for account creation would be an existing username
1	1.1.3	The client application will pass login information to the server to authenticate it	1. Receive user login info from GUI 2. Pass data to server	Server reponds with “success” for valid login or “failure” for invlid login		Object returned will be in JSON format.
2	1.2.1.1	The client application will alert the server to create a new game in the DB	1. The GUI will alert the client backend that the user would like to create a new game 2. The client application will send a request to the server with the UserID	Server responds with “success” and a gameID, or “failure”		Object returned will be in JSON format.
3	1.2.1.2	The client application will alert the server to add a user to a supplied gameID	1. The GUI will alert the client backend that the user would like to create a new game 2. The client application will send a request to the server with the UserID	Server responds with “success” and the opponent userID and playernumber, or “failure” if the game does not exist, has ended, or is full		Object returned will be in JSON format.
4	1.4.1	The chess board will be displayed to each user with their respective pieces oriented at the bottom of the board.	1. Create two users 2. Start a game	Both users will see the gameboard with their pieces oriented at the bottom		

5	1.4.2.1	The user will be able to offer a draw.	1. Create two users 2. Start a game 3. User 1 clicks the “Offer Draw” button	User 2 is notified that user 1 has offered a draw		
6	1.4.2.2	The user will be able to resign.	1. Create two users 2. Start a game 3. User 1 clicks the “Resign” button	User 2 is notified that user 1 has resigned. The game ends automatically		
7	1.4.2.3	The user will be able to request to abort the game.	1. Create two users 2. Start a game 3. User 1 clicks the “Request Abort” button	User 2 is notified that user 1 has request to abort the game		
8	1.4.2.4	The user will be able to view the game notation.	1. Create two users 2. Start a game 3. Play a few moves	Both users can see the game notation to the right of the board		
9	1.4.3.1	The game will end automatically in the case of the players agree to a draw.	1. Create two users 2. Start a game 3. User 1 clicks the “Offer Draw” button 4. User two accepts the draw offer by clicking the “Offer Draw” button	The game ends automatically in a draw		
10	1.4.3.2	The game will end automatically in the case of checkmate.	1. Create two users 2. Start a game 3. Play a sample game out to a checkmate position	The game ends automatically		
11	1.4.3.3	The game will end automatically in the case of stalemate.	1. Create two users 2. Start a game 3. Play a sample game out to a stalemate position	The game ends automatically		

12	1.4.3.4	The game will end automatically in the case of insufficient mating material.	1. Create two users 2. Start a game 3. Play sample games to test the following piece configurations: a. KN vs. K b. KNN vs. K c. KB vs. K	The game ends automatically in a draw when a specified configuration is reached		
13	1.4.3.5	The game will end automatically in the case of three-fold repetition.	1. Create two users 2. Start a game 3. Repeat the same position three times 4. User 1 (or user 2) claims a draw by clicking the “Offer Draw” button	The game ends automatically in a draw by three-fold repetition		
14	1.4.3.6	The game will end automatically in the case of a user runs out of time.	1. Create two users 2. Start a game 3. Allow user 1 to run out of time	The game ends automatically in a win for user 2		
15	1.4.4	The game will implement a clock accurate to 0.1 seconds.	1. Create two users 2. Start a game	Both users will see a clock for their time accurate to 0.1 seconds		
16	1.5.1	When a player clicks a piece, we must assert that none of the allowed moves allow the king to be left under attack	1. User clicks piece 2. Generate all possible moves for piece 3. Assert in validMove() that the move does not leave the king under attack. 4. If it does not, add it to the validMoves list	Unprotected move is not displayed to user		
17	1.5.2	When a player clicks a piece, we must assert that none of the allowed moves places the piece in the same location as a friendly	1. User clicks piece 2. Generate all possible moves for piece 3. Check if a friendly piece	Invalid move is not added to validMove list, thus not displayed to the user		

		piece	occupies the destination square			
18	1.5.3	When a piece has the ability to move multiple squares, it must not move past another piece	1. User clicks piece 2. Generate all possible moves for piece 3. Assert in validMove() that there are no pieces in direct line between the from and to coordinates.	If the move leads past another piece on the board, it is not shown to the user		
19	1.5.4.1	The king moves one space in any of eight directions	1. User clicks king 2. Generate all possible moves for the king 3. Assert in validMove() that the king cannot be moved into attack	If a king move is invalid it is not shown to the user		
20	1.5.4.2	Castling	1. User clicks king 2. Check in getMoves() if the king is on starting square and has not moved (then add to possible moves) 3. Assert in validMove() that the rook hasn't moved, there are no pieces between the king and rook,and the king isn't moving out of, through, or into attack	If legal, castling is displayed as a possible move to the user		
21	1.5.5	The rook moves horizontally or vertically in four directions	1. User clicks rook 2. Generate all possible moves for the rook 3. Assert in validMove() that each move is legal	Legal moves for the rook are displayed to the user		
22	1.5.6	The knight moves in a "L-shape",	1. User clicks knight	Legal moves for the knight are		

		one space horizontally or vertically, and then the two spaces in the other direction, and vice versa	2. Generate all possible moves for the knight 3. Assert in validMove() that each move is legal	displayed to the user		
23	1.5.7	The bishop moves diagonally in four directions	1. User clicks bishop 2. Generate all possible moves for the bishop 3. Assert in validMove() that each move is legal	Legal moves for the bishop are displayed to the user		
24	1.5.8	The queen moves horizontally, vertically, or diagonally in any of eight directions	1. User clicks queen 2. Generate all possible moves for the queen 3. Assert in validMove() that each move is legal	Legal moves for the queen are displayed to the user		
25	1.5.9.1	The pawn moves one or two spaces forward on its first move, and one space forward otherwise.	1. User clicks pawn 2. Check in getMoves() if the pawn has not moved (then add square two spaces forward) 3. Assert in validMove() that no other piece occupies destination square and that the move is legal	User sees possibility of moving pawn forward two spaces on first move, in addition to one space forward at any other time (if legal)		
26	1.5.9.2	The pawn moves diagonally one space forward when	1. User clicks pawn 2. Assert in validMove() that there must be an enemy piece	User sees possibility of capturing an enemy piece by moving the pawn one space forward		

		capturing and it cannot capture a piece directly in front of it	located one space forward diagonally 3. Assert in validMove() that the move is legal	diagonally (if legal)		
27	1.5.9.3	En passant	1. User clicks pawn 2. Assert in validMove() that there must be an enemy pawn that has moved forward two spaces through a square the pawn attacks on the preceding move 3. Assert in validMove() that the move is legal	User sees possibility of en passant (if legal)		
28	1.5.9.4	Promotion	1. User clicks pawn 2. User moves pawn to last rank 3. User is prompted to promote the pawn to a queen, rook, knight, or bishop	User is able to promote the pawn to a queen, rook, knight, or bishop		
29	1.3.5	The server will receive request to create a new user from the client	1. Receive request to create a user from the client 2. Parse the request 3. Create a user with the information	Respond “Success” and user id if new user can be created or “Failure” if user can not be created		Object is sent to client as JSON object. with a valid response string and user ID
30	1.3.5	The server will authenticate a login upon request from the client	1. Receive request to authenticate a user from the client 2. Parse the request 3. Authenticate user	Respond “Success” for valid info or “Failure” for invalid info		Object is sent to client as JSON object. With a valid response string
31	1.3.5	The server will receive a request to get a userid from a username	1. Receive request to obtain userid from username 2. Parse request	Respond “Success” with a user id if user was found or “Failure” if the information was invalid		Object is sent to client as a JSON object with a

			3. Return user id			response string and a user ID
32	1.3.6.1	The server will take in a request to create a game	1. Receive request from client 2. Parse request 3. Create game with first user	Respond “Success” with valid gameID or “Failure” if the game could not be created		Object is sent to client as JSON object with response status and gameID
33	1.3.6.2	The server receives a request to join a game	1. Receive request from client 2. Parse request 3. Find if the game id is a valid game 4. See if the game has only 1 player 5. Create new 2nd player in game 6. Return game id to client	Respond “Success” with a valid game ID of the joined game or “Failure” if the game could not be found or was full		Object is sent to client as JSON object with response status and gameID
34	1.3.7	The server receives a request to get the opponent information	1. Receive request from client 2. Parse request 3. Return opponent information	Respond “Success” with valid opponent information or “Failure” if there is no opponent or the information could not be obtained		Object is sent to the client as a JSON object with valid opponent userid, username, and player number
35	1.3.8	The server receives a request to check if a gameid is a valid game	1. Receive request from client 2. Parse requests 3. Return status of query	Respond “Success” if the game ID is a valid game ID and “Failure” if the game does not exist		Object is sent to the client as a JSON object with a response string
36	1.3.4	The server receives a request to send a move to the game and the opponent	1. Receive request from client 2. Parse request 3. Insert move into game and DB 4. Send move to client 5. Return status to client	Respond “Success” if the move was able to be saved to the database and send to the opponent and “Failure” if the move was not able to be saved		Object is sent to the client as a JSON object with a response string

				and send to the opponent.		
37	1.3.4	The server receives a request to send all moves from a game to the client	1. Receive request from client 2. Parse request 3. Obtain all moves from a game by game ID 4. Return moves to client	Respond “Success” if server was able to find a game and return all of the moves and “Failure” if the game was not valid		Object is sent to the client as a JSON object with a response string and a list of moves
38	1.1.2	The client application has started, and the login screen is shown.	1. Launch the application.	Login Screen is initialized and shown.		
39	1.1.3	The user types it’s correct login info and server responds that info is correct. Home screen is shown.	1. Launch Application. 2. Type Proper Login Information. 3. Press Login.	Login is shown as successful and Home screen is initialized.		
40	1.1.3	The user types it’s incorrect login info and server responds that info is correct. Home screen is shown.	1. Launch Application. 2. Type incorrect Login Information. 3. Press Login.	Login error is shown.		
41	1.1.1	User creates account	1. Launch application 2. Click register button	Register screen is initialized and shown.		
42	1.1.3	User creates account, and is shown as successful	1. Click register button. 2. Enter new user information 3. Click Create	Create account is shown as successful and home screen is initialized and rendered.		
43	1.1.3	User creates account, and is shown as unsuccessful	1. Click register button. 2. Enter old user information 3. Click Create	Error message that username already exists is shown.		
44	1.1.3	User presses Continue as Guest at Login Screen.	1. Launch Application 2. Press Continue as Guest	Home screen is initialized and shown.		

45	1.2.1.1	User presses Create Game on Home Screen.	1. Start Game 2. Login/continue as guest. 3. Click Create Game	Create Game screen is initialized, and rendered.		
46	1.2.1.3 /1.2.1. 2	User goes to join game, game list is shown.	1. Start Game 2. Login/continue as guest. 3. Click join game	Game list screen is initialized with all components and rendered.		
47	1.2.1	User logs out.	1. Login 2. Logout	Logout screen is initialized and rendered.		
48	1.4.1	User Joins game, game is shown.	1. Go to Join Game 2. Enter existing Game ID. 3. Press Join	Gameplay screen is initialized and rendered.		
49	1.2.1	User goes back to home screen from Join game	1. Go to Join Game 2. Press Back	User is returned to Home Screen		
50	1.4.1	User creates new game.	1. Go Home screen 2. Press Create Game 3. Enter Game Name and choose Color	Gameplay screen is initialized and rendered with the proper color on users side.		
51	1.2.1	User goes back to home screen from create game.	1. Go to Create Game Screen 2. Press Back	User is returned to Home Screen		
52	1.5.1/1 .5.2/1. 4.3	User moves chess piece.	1. Go to gameplay screen 2. Click piece to move 3. Select proper movement	Returns as a proper move and the piece graphic is move to proper place on board.		
53	1.5.1/1 .5.2/1. 4.3	User tries to make invalid move.	1. Go to gameplay screen 2. Click piece to move 3. Select invalid move	Returns as invalid move and Error is shown on GUI.		
54	1.5.1/1 .5.2/1. 4.3	User clicks piece to move	1. Go to gameplay screen. 2. Click piece to move	Possible moves are highlighted on screen. (Highlight for those spaces are initialized and rendered).		Piece and coordinates are sent to server and returned

						with possible coordinates to more.
55	1.4.3.2	User has won game.	1. Go to gameplay screen. 2. Get to end game where user has won.	End Game screen is initialized with win message, and rendered.		User can win in any way, does not matter.
56	1.4.3.2	User has lost game.	1. Go to gameplay screen 2. Get to end where user has lost.	End game screen initialized with loss message and rendered.		User can lose in any way
57	1.4.3.3	Game is stalemate	1. Go to game play screen 2. End game in stalemate	End game screen is initialized with stalemate message and shown.		
58	1.4.2	User presses options button during gameplay	1. Go to gameplay screen 2. Press options	Options menu is initialized and shown		
59	1.2.1	User returns to main screen from end game	1. End the game in any way. 2. Press return to main menu	Main menu is reinitialized and rendered.		