P3 Testing Document

CS414 - And Yet it Compiles

Laura South, Isaac Mauro, Brian Larson, Yan Wang, Duck Keun Yang

JungleBoard Tests:

Test Case 1

Purpose: Verify that a piece's power is reduced to zero when it moves onto one of the opposing player's traps and that the power is restored when the piece leaves the trap.

Action: Get example piece from JungleBoard. Move it to the opposing team's trap. Verify that the piece's power is equal to zero. Move the piece out of the trap and verify that the piece's power is the same as what it was before entering the trap.

Test Case 2

Purpose: Verify that the getPiece method returns correct pieces and responds reasonably to invalid input.

Action: Verify that the id of an example piece ("lion") matches the id attached the the piece that is returned by getPiece() when the arguments are the coordinates of the lion in the current baord state.

Test Case 3

Purpose: Verify output of getWinner() method within JungleBoard.

Action: Before making any moves, assert that getWinner() returns null because no one has won the game yet. Then, move a white piece into the black trap and assert that getWinner() returns white.

Test Case 4

Purpose: Test the behavior of the movePieceToTile method within JungleBoard.

Action: Move the white rat to a new tile. Check that the tile that the white rat used to be on no longer has a JunglePiece assigned to it. Check that the tile that white rat has been moved to has the correct JunglePiece (white rat) assigned to it. Verify that the rows and columns of the white rat piece match the rows and columns of the new tile it has been moved to.

JungleGame Tests
Test Case 1
Purpose: ensure that the systems responds properly when the player tries to move a piece off of the board.

Action: Move the black wolf to an invalid tile (for example, the tile at row -1 and column 4). Verify that the black wolf's position has not changed, because the move was not executed.

Test Case 2
Purpose: Check that the method getWinner returns null if the game has not completed yet.

Action: Simply check that the value of getWinner is null before any game moves have been executed.

Test Case 3
Purpose: Verify that getWinner returns Player1 if Player1 has won the game.

Action: Move the white leopard one tile at a time, respecting the rules of the game, so that it lands in the black trap. Check that getWinner() returns Player1.

Test Case 3
Purpose: Verify that getWinner returns Player2 if Player2 has won the game.

Action: Move the black leopard one tile at a time, respecting the rules of the game, so that it lands in the white trap. Check that getWinner() returns Player2.

Test Case 4
Purpose: Verify that the getPiece method returns the correct JunglePiece

Action: Call getPiece on an example piece and verify that the returned JunglePiece has coordinates matching the expected initial location of that particular example piece.

Test Case 5
Purpose: Check the behavior of getValidMove when the player tries to move to a non-adjacent piece.

Action: Attempt to move a JunglePiece to a non-adjacent tile and check that no changes were made to its position.

Test Case 6
Purpose: Check the behavior of the game when a player attempts to move onto the same square as, or capture, a piece that is on their own team.

Action: Try to move a JunglePiece onto the same tile as a piece that is on the same team (i.e. has the same color). Assert that no changes have been made the the first pieces location and that the second piece has not been captured by its own teammate.

Test Case 7
Purpose: Check that behavior of the system when a valid capture is executed.

Action: Move a black piece through the board, one tile at a time, until it reaches a white piece with lesser or equal power. Capture the piece. Assert that the captured piece has been removed from the board and that the attacking piece's coordinates are the same as what the captured piece's coordinates were previously.

Test Case 8
Purpose: Verify that the system properly handles the few cases where a piece may move to a non-adjecent tile -- i.e. when a tiger or a lion jumps over a river.

Action: Move a tiger piece so it is adjacent to the river, ensuring that there is no rat in the river currently. Move the tiger to the next land piece on the opposite side of the river. Assert that the tiger's coordinates are now equal to the location of that next land piece, despite the fact that the land tile is non-adjacent to the tiger's current spot.

Test Case 9
Purpose: Check that the system does not allow pieces that are not either tigers or lions to jump over river pieces.

Action: Move a wolf piece so that it is adjacent to a river tile. Attempt to move the wolf to the next land tile on the opposite side of the river. Assert that the wolf's location has not changed, because the move was not successful.

Test Case 10
Purpose: Check that the resetGame method moves all JunglePieces to their initial locations on the board.

Action: Move several pieces to spots that are different from their initial tiles. Call resetGame. Assert that the pieces that were moved earlier are not back in their expected starting locations.

Test Case 11
Purpose: Verify that the capturePiece method does not allow pieces to capture other pieces that have lesser power than they currently have.

Action: Attempt to capture a lion with a rat piece. Assert that capturePiece returns false and that no changes have been made to the board state.

JungleClient Tests
Set up class:
Start a mock server that simply stores the last received packet, and can send packets to the client
Set up:
Start a new JungelClient for every test case

Test case 1
Purpose: verify that login successfully sends the correct message, and that the client can handle receiving a login packet

Action: call login method, check if mock server received expected packet. Send login response packet to client, check if the client sets logged in status properly

Test case 3
Purpose: verify that register successfully sends the correct packet, and that the client can handle a registration packet

Action: call register method, check if the mock server received the expected packet. Client prints a message when it receives the register packet

Test case 4
Purpose: verify that unregister successfully sends the correct packet, and that the client can handle receiving an unregister packet

Action: call unregister method, check if the mock server received the expected packet. Client prints a message when it receives an unregister packet

Test case 5
Purpose: verify that findUser successfully sends the correct packet, and that the client can handle receiving a findUser packet

Action: call findUser method, check if the mock server received the expected packet. Client prints a message when it receives the user packet

Test case 6
Purpose: verify that invite successfully sends the correct packet, and that the client can handle receiving an invite response packet

Action: call invite method, check if the mock server received the expected packet. Client prints a message when it receives the Invite response

Test case 7
Purpose: verify that the client can handle receiving an invite from another user

Action: send an InviteRequest packet to the client from the mock server, check if the client receives the expected packet

ClientGameController test
Setup class
Start a mock server that simply stores the last received packet, and can send packets to the client
Set up
Start a new mock client that connects to the mock server and passes messages to the ClientGameController under test

Test case 1
Purpose: Verify that the game controller can make a valid move on the game board

Action: Figure out a valid move on the current board, and call the makeMove method. Check to see if the mock server received the expected packet

Test case 2
Purpose: Verify that the game controller won't make an invalid move on the board

Action: Figure out an invalid move, and call the makeMove method. Ensure that the mock server did not receive a make move packet

Test case 3
Purpose: Verify that that game controller won't make a move if it is not the players turn

Action: Set the controller so it is not the players turn and call the makeMove method. Ensure the mock server did not receive a make move packet

Test case 4
Purpose: Verify that that game controller won't allow the player to move a piece they don't own.

Action: Figure out a valid move for the other player, and call the makeMove method. Ensure that the mock server did not receive a make move packet

Test case 5
Purpose: Verify that quitting the game sends the correct packet to the server

Action: Call the quitGame method. Ensure that the mock server received a quitGame packet

Test case 6
Purpose: Verify that the game controller can handle a received move from the server

Action: calculate a valid move, and send a make move packet to the client. Verify that the game controller made the expected move on the game board.

Test case 7
Purpose: Verify that the game controller can handle a having its turn set by the server.

Action: send a set turn packet from the mock server, verify that the turn is set correctly in the game controller

Test case 8
Purpose: Verify that the game controller can handle a game over notification from the server

Action: send a game over packet from the mock server, verify that the game controller receives that packet and disposes of the game properly.

JungleServer Tests
Set up class:
Start a mock client that simply stores the last received packet, and can send packets to the server
Set up:
Start a new JungelServer and MockClient (user) for every test case.

Test case 1
Purpose: Verify a user can register successfully.

Action: Client sends a registration request to the server. Verify the client’s valid registration request works correctly and is stored in the database.

Test case 2
Purpose: Verify only one account can exist with a given email or username.

Action: Client sends a registration request to the server. Verify the client’s valid registration request works correctly and is stored in the database. Then send another registration request to the server using the same email, but a different username. Verify registration request is denied. Then send another registration request to the server using the same nickname, but a different email. Verify registration request is denied.

Test case 3
Purpose: Verify invalid emails are denied.

Action: Client sends a registration request to the server with an invalid email address. Verify registration request is denied.

Test case 4
Purpose: Verify registered user can login

Action: Client sends a registration request, followed by a login request. Verify the registered user is able to login successfully.

Test case 5
Purpose: Verify registered user can not login with invalid nickname.

Action: Client sends a registration request, followed by an invalid login request (non-registered username). Verify the registered user login request is denied with message ‘User not found’.

Test case 6
Purpose: Verify registered user can not login with invalid password.

Action: Client sends a registration request, followed by an invalid login request (incorrect password). Verify the registered user login request is denied with message ‘Incorrect password’.

Test case 7
Purpose: Verify registered user can unregister.

Action: Client sends a registration request, followed by an unregister request. Verify the user is no longer registered in the database.

Test case 8
Purpose: Verify user can not unregister invalid accounts (ie. not in database).

Action: Client sends an unregister request for a user not in the database. Verify the user is no longer registered in the database.

Test case 9
Purpose: Verify registered user can not unregister with incorrect password.

Action: Client sends a registration request, followed by an unregister request with an incorrect password. Verify the client request is denied.

ServerGameControllerTest
Set up class:
Start a mock server and 2 mock clients

Test case 1
Purpose: Verify that the game controller initializes the game state correctly and sends the turn notification to both players

Action: Create a new game controller for each client, and a game controller for the server with the 2 clients as the players. Verify that both clients have their turn flag set correctly. Verify that the server game and both client games are in the same state

Test case 2
Purpose: Verify that the server game controller recognizes when the game has been won, and sends the appropriate message to the clients

Action: make a winning move on one of the clients, verify that the server disposes of the game instance, and that both clients receive a game over packet.

Test case 3
Purpose: Verify that the server game controller correctly handles a move received from a client, and forwards the move to the other client

Action: make a valid move on one of the client game controllers, verify that the game state is still consistent between the server and both clients

Test case 4
Purpose: Verify that the server game controller correctly handles a client quitting the game

Action: quit the game from one of the client game controllers, verify that the server disposes of the game, and both clients receive a game over packet

ServerClientIntegrationTest
Set up class:
Start a JungleServer
Start 2 JungleClients

Test case 1 (currently only one huge test with very few asserts that mostly just prints information to the console)
Purpose: verify that the JungleServer and JungleClient backend communicate as expected

Action: Register and login with both clients, invite client 2 from client 1, check that the game is starting correctly, make a move on the gameboard, and check that both clients received the board update.