Server System Testing: Defect Documentation

Garrett Dieckmann, Kyle Monnett, Jordan Apele, Kyle Wenholz

November 10, 2011

Contents

1 Documentation **2**

1.1 Install Node.js . . . 2

1.2 Install MongoDB . . . 2

1.3 Develop for the server . . . 3

2 Communication **4**

2.1 Running an initial test of the server code . . . 4

2.2 First client connecting to system . . . 4

2.3 Client ID generated and signaled . . . 5

2.4 Client connecting to an open game . . . 5

2.5 Client connecting to a full game . . . 6

2.6 Request client start new game . . . 6

2.7 Many spectators . . . 6

3 Game Logic **8**

3.1 Two players play a game, disregarding paddle logic . . . 8

3.2 Update player scores-Client1 scores 1 point . . . 8

3.3 Paddle logic . . . 9

3.4 Ball logic usability . . . 9

4 Performance **11**

4.1 Between client latency . . . 11

4.2 Two clients play a game until a winner is determined . . . 11

4.3 Available game information sent . . . 12

4.4 Desired communication time . . . 12

4.5 Client 1 waiting for Client 2 . . . 13

4.6 Ability to store data . . . 14

5 Style Guide **15**

5.1 Whitespace . . . 15

5.2 Comments . . . 15

5.3 Functions & braces . . . 16

5.4 Variables & nomenclature . . . 16

6 Awesomeness **17**

1 Documentation

Note that not all test cases include an expected results section. This is only in cases where the *Post − conditions* are clear enough. Throughout the document there are references to files used in testing. These should be mostly obvious (i.e. VisualClient refers to a client capable of visualizing the ball and paddle data). Some of the score clients were special and are in the testing files. Also in the testing files, you will find notes, copies of clients used and various other possibly helpful tidbits.

1.1 Install Node.js

Version:1.0
Description: Attempt to use documentation for Node.js installation.
Pre-conditions: Ubuntu 10.10 or later is installed on a machine with network access.
Post-conditions: Node.js is installed on the server and ready for use.
Scenario:

1. Go to section 5.2.1 of the Server Team intermediate report.
2. Follow each step indicated to install Node.js.
3. Follow the instructions to install socket.io and express.

Pass/Fail: Fail
Comments: The user should be encouraged to install the build packages necessary on their system. Sources should be cited for credit and more assistance should it be needed. Also, the instruction to `''<cd your directory name>''` for express installation is entirely unhelpful.

1.2 Install MongoDB

Version:1.0
Description: Attempt to use documentation for MongoDB installation.
Pre-conditions: Ubuntu 10.10 or later is installed on a machine with network access.
Post-conditions: MongoDB is installed on the server and ready for use.
Scenario:

1. Go to section 5.2.1 of the Server Team intermediate report.
2. Follow each step indicated to install MongoDB.

Pass/Fail: Fail
Comments: The install documentation should provide references or links to resources for installation. Installation following the procedures given is incorrect. The downloaded source is never

actually used. Also, the use of sudo apt-get mongodb assumes that the package is in the repository of the users system and that he or she is using a Debian based system. Lastly, the final paragraph of the installation instructions is very confusing.

1.3 Develop for the server

Version:1.0
Description: Design a client to interact with the server.
Pre-conditions: Be prepared to develop a client for interacting with the server.
Post-conditions: A client exists that can communicate with the server in a meaningful way.
Scenario:

1. Understand the server through documentation and an API.
2. Develop a client to interact with the server.
3. Use the client.

Pass/Fail: Fail
Comments: There needs to be an API to understand the server. As is, the documentation and looking at the code are very confusing for a new developer. While an API may seem like a secondary goal, developing without one is impossible without intimate knowledge of the server code.

2 Communication

2.1 Running an initial test of the server code

Version:1.0
Description: Using the most recent version of the paddle-meister code, run the server using node and try to connect with WebUI code from the mid-term demo.
Pre-conditions: Node.js installed and paddle-meister up to date. A web browser and the WebUIs demo code for the pong view are also required on a test machine (server or other machine).
Post-conditions: Server logs output for client connections and client side draws or responds in any fashion.
Scenario:

1. Run the server file using node server.js.
2. Open the client files in browser (twice, once per tab in browser).
3. Observe output.

Pass/Fail: Fail
Comments: The most recent server.js file does not work. We had to revert to the demo version after discussing with the server team.

2.2 First client connecting to system

Version:1.0
Description: Test successful client initiated communication between the server and client.
Pre-conditions: Client has access to network, system is connected to network and accepting connections, system has no other clients connected.
Post-conditions: Client has initiated connection and server has responded.
Scenario:

1. Client requests connection with system.
2. System accepts connection...

Pass/Fail: Pass
Comments: System makes connection with client.

2.3 Client ID generated and signaled

Version: 1.0
Description: Test for successful generation and passing of client ID.
Pre-conditions: Client has access to network, system is connected to network and accepting connections, system has just received a request to connect from client.
Post-conditions: Client has received system generated ID and ready signal.
Scenario:

1. Test case one passes.
2. System generates Client ID.
3. System sends client its ID and ready signal.

Pass/Fail: Fail
Comments: Client sent ID client indicated in initiating game.

2.4 Client connecting to an open game

Version: 1.0
Description: A Client connects to system requesting a specific game session, and system connects Client to existing game session with an already-connected Client and begins game session, communicating game data.
Pre-conditions: System and Clients operational on the same network, One Player is already connected waiting for a challenger. Using the VisualClient.
Post-conditions: Game has begun between Clients.
Scenario:

1. Client communicates to server which open game it wants to play, and System opens communication between client and client already in open game.
2. System initializes the game state, the game is started.
3. Paddle position is transmitted from client to the system.
4. System transmits clients paddle position, ball position, and score data back to client.

Expected Results: The changes in paddle position of both Clients are recorded by the System and communicated to both clients in synchronization; both Clients are displaying same game state.
Pass/Fail: Mostly success.
Comments: The client can not select a game; however, the client can connect to a single game.

2.5 Client connecting to a full game

Version: 1.0
Description: A Client connects to system requesting a specific game session, System notifies Client of full status of session and does not give Client access.
Pre-conditions: System and Clients operational on the same network, Game Session occupied by two Clients. Using VisualClient.
Post-conditions: Client returns to game listing.
Scenario:

1. Client communicates to server which game it wants to play
2. System discovers game session is full.
3. System notifies Client of error and does not admit to game session.

Expected Results: Client receives message.
Pass/Fail: Fail
Comments: This feature is not yet implemented.

2.6 Request client start new game

Version: 1.0
Description: Test system behavior if no current games are available.
Pre-conditions: Client has access to network, system is connected to network and accepting connections, system has just received a request to connect from client and received ID and ready signal.
Post-conditions: Client is asked to start a new game.
Scenario:

1. Test case two passes.
2. System discovers no games currently exist.
3. System asks client to start a new game.

Pass/Fail: Feature not implemented yet.
Comments: The server doesn't ask; it just starts new game.

2.7 Many spectators

Version: 1.0
Description: Spectators connect to watch a game.

Pre-conditions: System and Clients operational on the same network. Game has begun for playing clients (using PlayerClient or VisualClient). SpectatorClient is used to connect as a spectator.
Post-conditions: Game proceeds for clients and spectators are able to view the game.
Scenario:

1. Spectator selects to watch a game.
2. Spectator registers with server.
3. Repeat steps 1 and 2 several times. (Refreshing the page can do this.)

Expected Results: The game proceeds up to reasonable expectations.
Pass/Fail: Fail
Comments: For each newly connected spectator, or player, the ball visibly gains speed. Head to line 144 for the if-statement that causes this issue.

3 Game Logic

3.1 Two players play a game, disregarding paddle logic

Version: 1.0
Description: Two PlayerClients play a Pong game.
Pre-conditions: System and Clients operational on the same network. Game has begun for two playing clients (using PlayerClient or VisualClient).
Post-conditions: Game continues for both players.
Scenario:

1. System sends ball movement to both users.
2. System receives position data information from client.
3. System informs game logic of movement.
4. System sends position data to other client.

Expected Results: The game will continue to be playable between two users.
Pass/Fail: Fail
Comments: When a new PlayerClient or a SpectatorClient joins an existing game between two PlayerClients, both paddles (Player1s paddle and Player2s paddle) will be moved to the default positions.

3.2 Update player scores-Client1 scores 1 point

Version: 1.0
Description: This will test if any defects appear when a client scores a point. The game logic should determine no collision, and the system should increment the point winner's score, and send the updated score to each client.
Pre-conditions: System Operational, Client1 and Client2 logged in and participating in game
Post-conditions: Game continues
Scenario:

1. Client1 moves to hit the ball.
2. Game logic determines ball movement.
3. Client2 does not move.
4. Game logic determines no collision.
5. System should send updated score to both Client1 and Client2.

Expected Results: Client receives updated score reflected as Client1 has 1 point, Client 2 has 0 points. Game should continue.
Pass/Fail: Fail
Comment: Due to the fact that the server does not send the updated score to the clients, boundary and functionality testing was not conducted on the score keeping.

3.3 Paddle logic

Version: 1.0
Description: This will test if any defects appear when a client is using the paddle to hit the ball.
Pre-conditions: System Operational, Client1 and Client2 logged in and participating in game
Post-conditions: Game continues
Scenario:

1. Client1 moves to hit the ball.
2. Game logic determines collision when Client1's paddle comes in any contact with the ball.
3. Game logic determines ball movement.

Expected Results: Client can hit ball with any part of their paddle.
Pass/Fail: Fail
Comments: Client would like to be able to hit the ball with the side of their paddle, not just the face.

3.4 Ball logic usability

Version: 1.0
Description: This will test if any defects appear in ball when client plays a game.
Pre-conditions: System Operational, Client1 and Client2 logged in and participating in game
Post-conditions: Game continues
Scenario:

1. Client1 moves to hit the ball.
2. Game logic determines collision when Client1's paddle comes in any contact with the ball.
3. Game logic determines ball movement.
4. Client2 moves to hit the ball.
5. Game logic determines collision when Client2's paddle comes in any contact with the ball.

6. Game logic determines ball movement.

Expected Results: Client finds ball movement to be logical and usable.
Pass/Fail: Pass
Comments: Client would like more diversity in initial ball movement, both in which direction the ball is served as well as to which client the ball is served.

4 Performance

4.1 Between client latency

Version: 1.0
Description: This will test if any defects appear in the latency between separate clients (both players and spectators) logged into the same game.
Pre-conditions: System Operational, Client1 and Client2 logged in and participating in game, Client3 (a spectator) is logged on watching.
Post-conditions: Game continues
Scenario:

1. Clients continually play and watch the game from different computers.
2. Clients notice the latency between the computers, both from the players' and spectators' computers.

Expected Results: Client finds that the game is running simultaneously on every computer.
Pass/Fail: Pass

4.2 Two clients play a game until a winner is determined

Description: This test will determine whether a game informs a client of the score based upon the score limit for a game.
Pre-conditions: The system is in operation, two clients are connected to the system, clients are prepared to start a game
Post-conditions: The game is complete.
Scenario:

1. System pairs users
2. Game logic sets players and ball at default positions
3. System informs clients of game start
4. Game logic starts ball movement
5. System sends ball movement to both users
6. System receives position data information from client
7. System informs game logic of movement
8. System sends position data to other client

9. Game logic determines no collision
10. User receives a point
    Repeat 4 - 10 until score limit - Score limit not reached by 50 non-collisions
11. System informs client of score

Return to 5
Expected Results: A game informs client of the score, after reaching the game score limit.
Pass/Fail: Fail
Comments: No score limit is reached. After 50 non-collisions, or scores, the game did not inform the client of the score

4.3 Available game information sent

Version: 1.0
Description: Test for transmission of current available games.
Pre-conditions: Client has access to network, system is connected to network and accepting connections, system has just received a request to connect from client and received ID and ready signal.
Post-conditions: Client knows what current games are available.
Scenario:

1. Connect to the server and request a game.
2. Server sends client available game information.
3. Client receives information.

Expected Results: Client receives information about current games.
Results: This feature is not yet implemented.

4.4 Desired communication time

Version: 1.0
Description: This test will quantify the latency between issuing a paddle movement on the website, and when the client receives a gameUpdate event corresponding to the paddle movement issued.
Pre-conditions: Server is running Pong code, two clients are connected to a Pong game, a Pong game is in progress.
Test structure: A timing-event start is placed at the updatePaddleToServer method, and a timing-event stop is placed at the updateGame event listener.
Scenario:

1. A client moves their pong Paddle
2. The client code sends updated paddle location to server
3. The client code starts a timer
4. The system sends updated game information to the client code
5. The client code stops the timer

Outcome: Average time for 25 time-points is 29.52milliseconds with standard deviation of 14.08 milliseconds
Comments:

1. Server code has a hard-coded response interval of 50 milliseconds
2. In the test, the client machine was able to ping the server and get an average response of 1 millisecond

4.5 Client 1 waiting for Client 2

Version: 1.0
Description: This test will check whether Client 1 is able to wait for Client 2 to join.
Pre-conditions: The server is running Pong code, no clients are connected to the game
Post-conditions: Game begins when Client 2 joins the game
Scenario:

1. Client 1 navigates to the gamePlay webpage
2. Client code notifies server that they have connected
3. Server accepts client as a player 1
4. Server waits for another client to join
5. Client 2 connects to gamePlay webpage
6. Client code notifies server that they have connected
7. Server accepts client 2 as player 2
8. Game begins

Expected Results:

1. The game will wait for two players to join

2. The game will begin when two players join

Pass/Fail: Will commonly pass, sometimes fail.
Comments: If only Client 1 is connected, and Client 2 has not yet joined, Client 1 can crash the server. If Client 1 presses any key (whether it is a movement key, or a any keyboard input) before before Client 2 has connected, this will cause the server to crash, and not recover. The server crashes at line 152 of Server.js

4.6 Ability to store data

Version: 1.0
Description: Can the server store data?
Pre-conditions: Client has access to network, system is connected to network and accepting connections, client connected.
Post-conditions: Some information is stored by the server.
Scenario:

1. Play a game.

2. Try to retrieve information afterwards.

Expected Results: Server stores game, player, or anything information.
Pass/Fail: Fail
Comments: No database exists.

5 Style Guide

5.1 White space

Version: 1.0
Description: Look for inconsistent white space.
Pre-conditions: Have code
Post-conditions:N/A
Scenario:N/A
Expected Results:N/A
Pass/Fail: Fail
Comments:

1. 9: Why is there so much white space here?
2. 34: Missing whitespace.
3. 66/67: White space needs cleaning up.
4. 70-77: White space needs cleaning up.
5. 70-77 Suggestion: Use a switch with cases instead of multiple IF statements.

5.2 Comments

Version: 1.0
Description: Proper comments.
Pre-conditions: Have code
Post-conditions:N/A
Scenario:N/A
Expected Results:N/A
Pass/Fail: Fail
Comments:

1. Comments should be more specific and fleshed-out for functions (i.e. multi-line comments here). Use JavaDoc style for these (Lines 28, 45, 54, 60, 90).
2. TODO Comments should be noted as such.
3. Code commented out should have a comment describing why said code is commented out.
4. 81: Why is this block temporary for testing?

5.3 Functions & braces

Version: 1.0
Description: Functions and their braces.
Pre-conditions: Have code
Post-conditions:N/A
Scenario:N/A
Expected Results:N/A
Pass/Fail: Fail
Comments:

1. Check spacing with "{" character on lines 28, 33, 36, 39, 46, 65, 69, 70, 85. Refine style definition – is the space before "{" just for function definitions or for all code-blocking?

5.4 Variables & nomenclature

Version: 1.0
Description: Variables & Nomenclature.
Pre-conditions:N/A
Post-conditions:N/A
Scenario:N/A
Expected Results:N/A
Pass/Fail: Fail
Comments:

1. What variables are global? They should be noted as such.

2. "a" prefix required for parameters (54, 60, 90).

6 Awesomeness

A final note: great coding Server Team. It has been a true pleasure reviewing your work.

All the best, Kyle.