**COIT13229**

**Applied Distributed System**

**ZMQ Test Documentation**

|  |  |
| --- | --- |
| **Name:** | Ujjwal Dhakal |
| **Student Number:** | 12222900 |

|  |  |
| --- | --- |
| **Name:** | Sultan Zahid |
| **Student Number:** |  |

Table of Contents

[Tests 3](#_Toc199119608)

[Functional Test Cases 3](#_Toc199119609)

[TC-1: Send base64 chat message 3](#_Toc199119610)

[TC-2: Click tower and trigger GET 4](#_Toc199119611)

[TC-3: Reset button clicked 5](#_Toc199119612)

[TC-4 Browsers Polling /update 6](#_Toc199119613)

[Fault Tolerance Test Cases 9](#_Toc199119614)

[FT-1: Inject Delay Before Socket Receive 9](#_Toc199119615)

[FT-2: Reorder Tower Messages in ZMQ 10](#_Toc199119616)

[FT-3: Kill Peer During Active Game 11](#_Toc199119617)

[FT-4: Send Same Message Twice 11](#_Toc199119618)

[Test Summary 13](#_Toc199119619)

# Tests

## Functional Test Cases

### TC-1: Send base64 chat message

**Test Name:** test\_message\_hello\_world()

**Test Type**: Automatic and manual

**Description:** Sends “Hello world” as a chat message from one peer and checked via /update

**Expected**: Boolean value to verify that chat message is received (True)

**Actual:** True

**Status:** Pass

**Screenshot:**

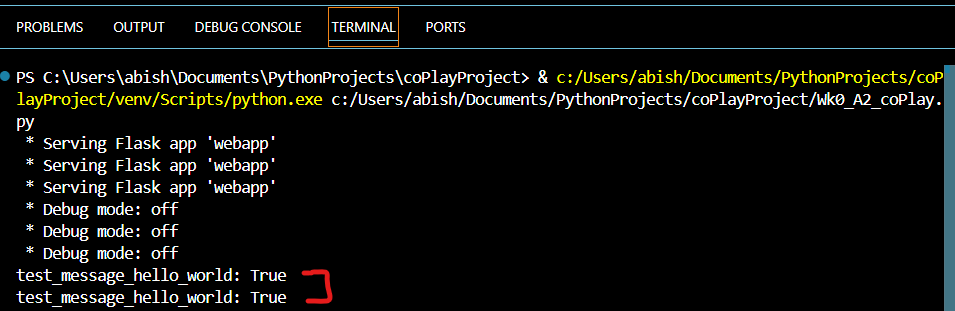


Fig: hello world message test output

**A screenshot of a computer

AI-generated content may be incorrect.**

Fig: Manual hello world message test

### TC-2: Click tower and trigger GET

**Test Name:** Click Tower Test

**Test Type**: Manual

**Description:** Triggers disk movement via GET and reflects across all peers

**Expected**: Same disk moves across all peers

**Actual:** Exact same disk movement among host and peers.

**Status:** Pass

**Screenshot:**

A screenshot of a computer

AI-generated content may be incorrect.

Fig: Initial Game State

A screenshot of a computer

AI-generated content may be incorrect.

Fig: Same disk moves across all peers

### TC-3: Reset button clicked

**Test Name:** Click Reset button

**Test Type**: Manual

**Description:** Game resets after reset button is clicked.

**Expected**: Game resets across all peers when reset button is clicked

**Actual:** Game resets but only on the peer whose reset button is clicked

***Actual After Update:*** Game resets with reset message sent among peers.

**Status:** Partial; [Pass (Game resets); Fail (Game resets only for that specific peer)]

***Status After Update:*** Pass

**Screenshot:**

A screenshot of a computer

AI-generated content may be incorrect.

Fig: Game completed in fewest moves

A screenshot of a computer

AI-generated content may be incorrect.

Fig: Games resets for only one peer.

Screens screenshot of a computer

AI-generated content may be incorrect.

Fig: Reset request message [After code update]

Screens screenshot of a computer

AI-generated content may be incorrect.

Fig: Game resets consistently across all peers [After code update]

TC-4 Browsers Polling /update

**Test Name:** Polling /update across peers to test consistency **Scope:** In-scope   
**Test Type:** Manual   
**Description:** Checking the consistency across 2 peers considering chat, disk movement and reset

**Expected:** Chats, Disk movements and Reset are in sync and consistent at all times.

**Actual:** Initial chat and disk moves are consistent after update

**Status:** Pass

**Screenshots:**

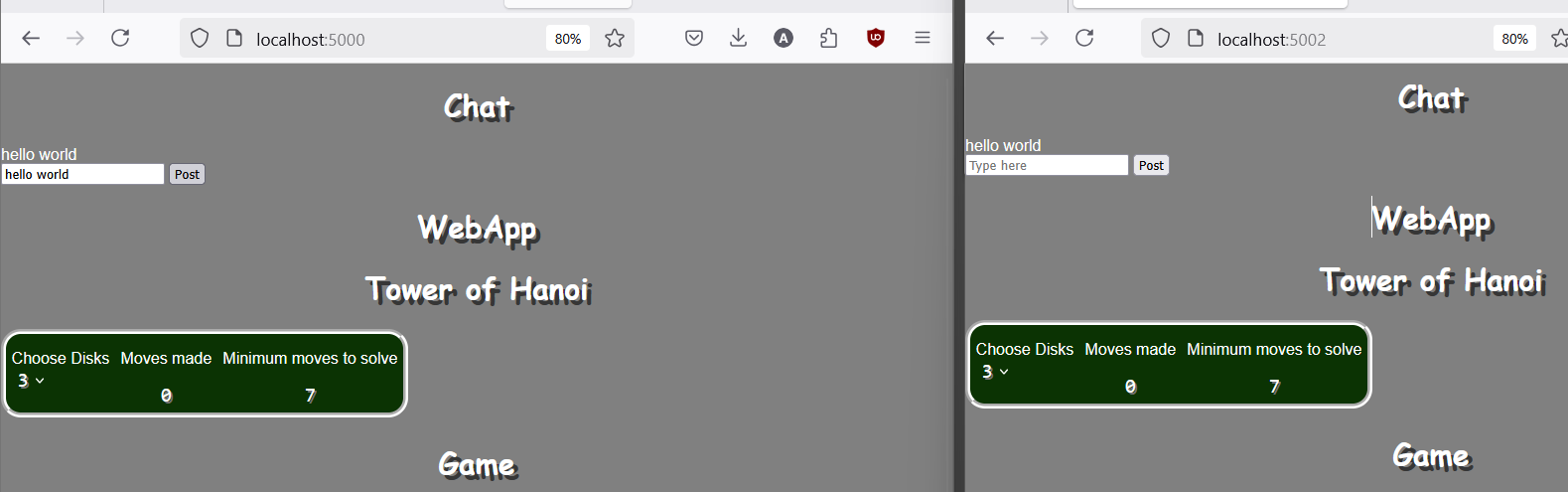
****

Fig: Consistent Chat State

A screenshot of a computer

AI-generated content may be incorrect.

Fig: Consistent Disk Movements

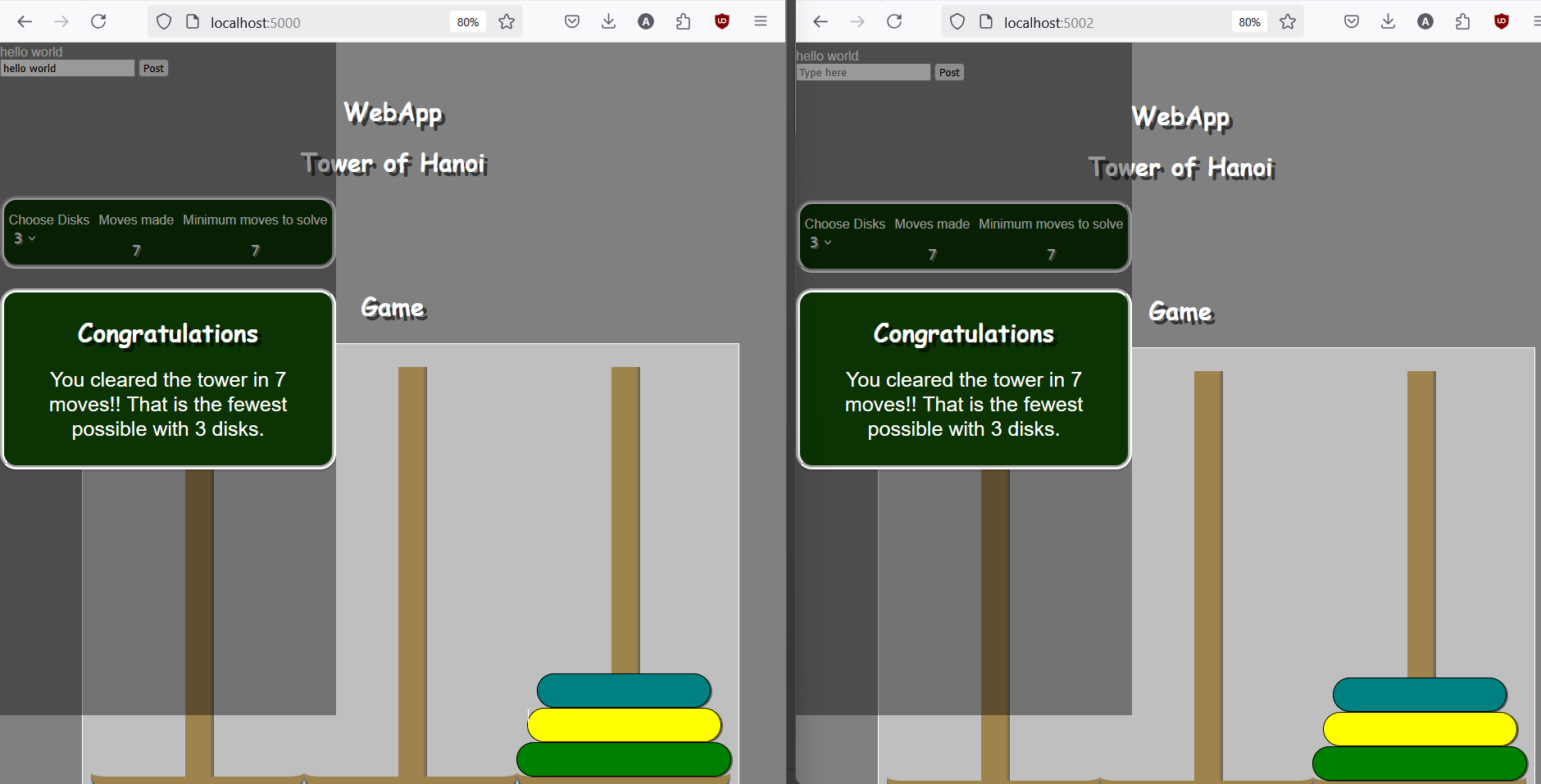
****

Fig: Consistent game clearence

A screenshot of a computer

AI-generated content may be incorrect.

Fig: Inconsistent Game Reset

A screenshot of a computer

AI-generated content may be incorrect.

Fig: Inconsistent disk movement

A screenshot of a computer

AI-generated content may be incorrect.

Fig: Consistent no. of disk changes [After code update]

A screenshot of a computer

AI-generated content may be incorrect.

Fig: Consistent chats [After code update]

## Fault Tolerance Test Cases

### FT-1: Inject Delay Before Socket Receive

**Test Name:** test\_simulated\_lag()

**Test Type:** Automatic

**Description:** Simulates 1.5s delay before receiving message via /update.

**Test Scope:** In-Scope

**Expected**: Message still received after delay.

**Actual:** Message received successfully with 1.5s delay.

**Status:** Pass

**Screenshot:**

A screen shot of a computer

AI-generated content may be incorrect.

Fig: Lag Test Output

### FT-2: Reorder Tower Messages in ZMQ

**Test Name:** test\_message\_reorder()

**Test Type**: Automatic

**Description:** Sends multiple messages quickly to test out-of-order delivery.

**Test Scope:** In-Scope

**Expected:** All messages received, regardless of order.

**Actual:** Messages received in different order. Logic preserved.

**Status:** Pass

**Screenshot:**

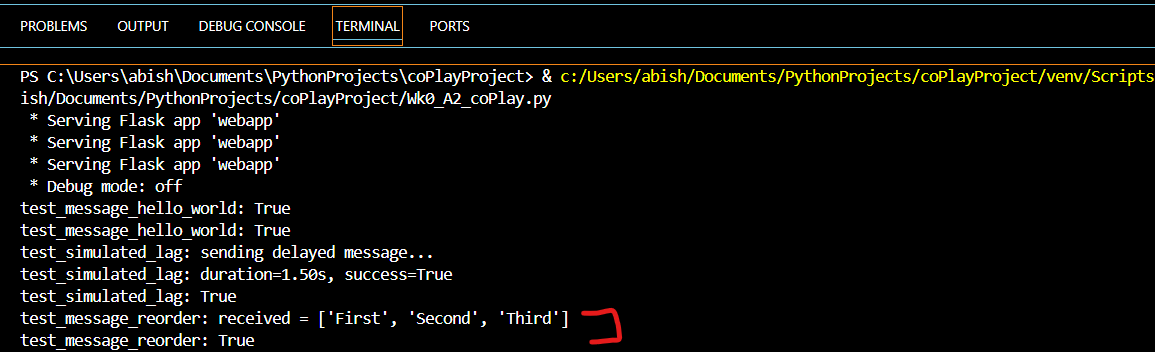


Fig: Message Reorder Test Output

### FT-3: Kill Peer During Active Game

**Test Name:** test\_peer\_crash\_mid\_broadcast()

**Test Type:** Automatic

**Description:** Simulates shutdown of peer during broadcast.

**Test Scope:** Out-of-scope

**Expected:** Other peers continue functioning.

**Actual:** Runtime error occurred on live peer during update.

**Status:** Fail (expected; crash not handled in prototype)

**Screenshot:**

A screenshot of a computer program

AI-generated content may be incorrect.

Fig: Peer Crash Output

### FT-4: Send Same Message Twice

**Test Name:** Duplicate Message Test

**Test Type:** Manual

**Description:** Send identical chat message twice.

**Test Scope:** Out-of-scope

**Expected:** Only one copy displayed.

**Actual:** Duplicate shown in chat.

**Status:** Fail (expected; deduplication not implemented in prototype)

**Screenshot:**

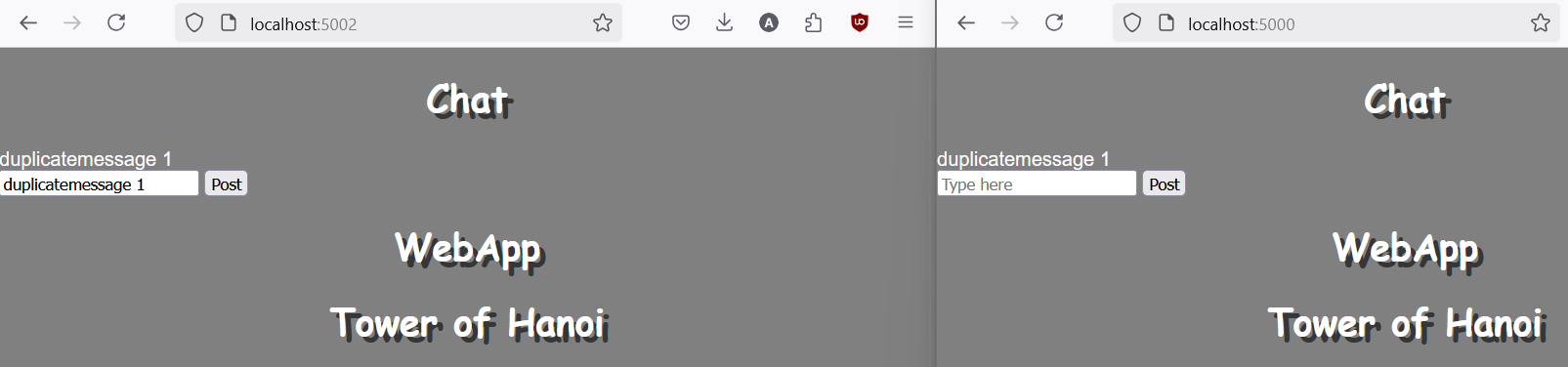


Fig: Duplicate message Test

A screenshot of a computer

AI-generated content may be incorrect.

Fig: Duplicate message Retest

A screen shot of a computer

AI-generated content may be incorrect.

Fig: Duplicate message logs

# Test Summary

The ZMQ-based coPlay system successfully passed all core functional test cases, including chat synchronization, tower movement, and reset functionality across peers.

However, in TC-4, after one peer triggered a reset, subsequent disk movements became inconsistent across peers due to lack of shared game state synchronization.

For fault tolerance, the system handled injected delay (FT-1) and message reordering (FT-2) correctly, maintaining stable behaviour.

A peer crash scenario (FT-3) resulted in a acceptable failure as crash recovery is out-of-scope for the ZMQ phase.

Duplicate chat messages (FT-4) are displayed multiple times, as deduplication is not implemented.

Overall, the ZMQ implementation fulfills in-scope requirements and appropriately demonstrates limitations for out-of-scope fault scenarios.

# Appendix

Test methods:

# Test that simulates lag by introducing delay in receiving update

def test\_simulated\_lag():

    global TESTING\_DELAY

    TESTING\_DELAY = 1.5  # 1.5 second processing delay

    print("test\_simulated\_lag: sending delayed message...")

    url1 = 'http://127.0.0.1:5000/message'

    json\_data = base64.b64encode("Delayed message".encode('utf-8'))

    requests.post(url1, json\_data)

    # Try to fetch with delay and check result

    url2 = 'http://127.0.0.1:5002/update'

    start\_time = time.time()

    response = requests.get(url2).json()

    end\_time = time.time()

    TESTING\_DELAY = 0  # Reset processing delay

    success = any("Delayed message" in msg.get("message", "") for msg in response)

    print(f"test\_simulated\_lag: duration={end\_time-start\_time:.2f}s, success={success}")

    return success

# Test that simulates message reorder by sending multiple messages

def test\_message\_reorder():

    messages = ["First", "Second", "Third"]

    for msg in messages:

        url = 'http://127.0.0.1:5000/message'

        json\_data = base64.b64encode(msg.encode('utf-8'))

        requests.post(url, json\_data)

        time.sleep(0.1)

    url = 'http://127.0.0.1:5002/update'

    response = requests.get(url).json()

    received = [msg.get("message") for msg in response if "message" in msg]

    print("test\_message\_reorder: received =", received)

    return set(messages).issubset(set(received))

#Test that simulates peer crash mid broadcast

def test\_peer\_crash\_mid\_broadcast():

    print("test\_peer\_crash\_mid\_broadcast: Simulating crash")

    # Normal message post

    url1 = 'http://127.0.0.1:5000/message'

    json\_data = base64.b64encode("Crash Test Msg".encode('utf-8'))

    requests.post(url1, json\_data)

    #  Kill peer on port 5004 (simulate crash)

    try:

        print("Attempting to kill peer on port 5004...")

        requests.get('http://127.0.0.1:5004/shutdown')

    except Exception as e:

        print("Peer 5004 crashed or unreachable:", e)

    # Broadcast another message

    json\_data = base64.b64encode("Post-crash Msg".encode('utf-8'))

    requests.post(url1, json\_data)

    # Try to fetch updates from another peer who is live

    url2 = 'http://127.0.0.1:5002/update'

    try:

        result = requests.get(url2).json()

        found = any("Post-crash Msg" in msg.get("message", "") for msg in result)

        print("test\_peer\_crash\_mid\_broadcast: remaining peers received message =", found)

        return found

    except Exception as e:

        print("Live peer failed:", e)

        return False

# Tests

def tests():

    do\_test\_message\_hello\_world=True

    if do\_test\_message\_hello\_world:

        print(f"test\_message\_hello\_world: {test\_message\_hello\_world()}")

        test\_clean()

        print(f"test\_message\_hello\_world: {test\_message\_hello\_world()}")

        test\_clean()

        print(f"test\_simulated\_lag: {test\_simulated\_lag()}")

        test\_clean()

        print(f"test\_message\_reorder: {test\_message\_reorder()}")

        test\_clean()

        print(f"test\_peer\_crash\_mid\_broadcast: {test\_peer\_crash\_mid\_broadcast()}")

        test\_clean()