--- BCE_test_close_browser.py ---

```
from test_init import BaseTestCase, patch, logging, unittest


class TestBrowserFunctionality(BaseTestCase):


    @patch('entity.BrowserEntity.BrowserEntity.close_browser')
    def test_close_browser_success(self, mock_close):
        """Test successful browser close."""
        print("\nTest Started for: test_close_browser_success")
        mock_close.return_value = "Browser closed."
        expected_entity_result = "Browser closed."
        expected_control_result = "Control Object Result: Browser closed."
        result = self.control.receive_command("close_browser")

        logging.info(f"Entity Layer Expected: {expected_entity_result}")
        logging.info(f"Entity Layer Received: {mock_close.return_value}")
              self.assertEqual(mock_close.return_value,  expected_entity_result,  "Entity  layer  assertion
failed.")
        logging.info("Unit Test Passed for entity layer.\n")

        logging.info(f"Control Layer Expected: {expected_control_result}")
        logging.info(f"Control Layer Received: {result}")
        self.assertEqual(result, expected_control_result, "Control layer assertion failed.")
        logging.info("Unit Test Passed for control layer.\n")

    @patch('entity.BrowserEntity.BrowserEntity.close_browser')
    def test_close_browser_not_open(self, mock_close):
        """Test closing a browser that is not open."""
        print("\nTest Started for: test_close_browser_not_open")
        mock_close.return_value = "No browser is currently open."
        expected_entity_result = "No browser is currently open."
        expected_control_result = "Control Object Result: No browser is currently open."
        result = self.control.receive_command("close_browser")

        logging.info(f"Entity Layer Expected: {expected_entity_result}")
        logging.info(f"Entity Layer Received: {mock_close.return_value}")
            self.assertEqual(mock_close.return_value, expected_entity_result, "Entity layer assertion
failed.")
        logging.info("Unit Test Passed for entity layer.\n")

        logging.info(f"Control Layer Expected: {expected_control_result}")
        logging.info(f"Control Layer Received: {result}")
        self.assertEqual(result, expected_control_result, "Control layer assertion failed.")
        logging.info("Unit Test Passed for control layer.\n")

    @patch('entity.BrowserEntity.BrowserEntity.close_browser')
    def test_close_browser_failure(self, mock_close):
        """Test control layer's handling of an unexpected error during browser close."""
        print("\nTest Started for: test_close_browser_failure")
        mock_close.side_effect = Exception("Unexpected error")
        expected_result = "Control Layer Exception: Unexpected error"
        result = self.control.receive_command("close_browser")

        logging.info(f"Control Layer Expected to Report: {expected_result}")
        logging.info(f"Control Layer Received: {result}")
        self.assertEqual(result, expected_result, "Control layer failed to handle or report the error correctly.")
        logging.info("Unit Test Passed for control layer error handling.\n")


    @patch('entity.BrowserEntity.BrowserEntity.close_browser')
    def test_close_browser_failure_entity(self, mock_close):
        """Test failure to close the browser due to an internal error in the entity layer."""
        print("\nTest Started for: test_close_browser_failure_entity")
        internal_error_message = "BrowserEntity_Failed to close browser: Internal error"
        mock_close.side_effect = Exception(internal_error_message)  # Simulate an exception on error
        expected_control_result = f"Control Layer Exception: {internal_error_message}"

        # Execute command
        result = self.control.receive_command("close_browser")

        # Check if the control layer returns the correct error message
        logging.info(f"Entity Layer Expected Failure: {internal_error_message}")
        logging.info(f"Control Layer Received: {result}")
        self.assertEqual(result, expected_control_result, "Control layer failed to report entity error correctly.")
        logging.info("Unit Test Passed for entity layer error handling.\n")


if __name__ == '__main__':
    unittest.main()
```

--- BCE_test_launch_browser.py ---

```
from test_init import BaseTestCase, patch, logging, unittest


class TestBrowserFunctionality(BaseTestCase):


    @patch('entity.BrowserEntity.BrowserEntity.launch_browser')
    def test_launch_browser_success(self, mock_launch):
        """Test successful browser launch."""
        print("\nTest Started for: test_launch_browser_success")
        mock_launch.return_value = "Browser launched."
        expected_entity_result = "Browser launched."
        expected_control_result = "Control Object Result: Browser launched."
        result = self.control.receive_command("launch_browser")

        logging.info(f"Entity Layer Expected: {expected_entity_result}")
        logging.info(f"Entity Layer Received: {mock_launch.return_value}")
            self.assertEqual(mock_launch.return_value, expected_entity_result, "Entity layer assertion
failed.")
        logging.info("Unit Test Passed for entity layer.\n")

        logging.info(f"Control Layer Expected: {expected_control_result}")
        logging.info(f"Control Layer Received: {result}")
        self.assertEqual(result, expected_control_result, "Control layer assertion failed.")
        logging.info("Unit Test Passed for control layer.\n")
    @patch('entity.BrowserEntity.BrowserEntity.launch_browser')
    def test_launch_browser_already_running(self, mock_launch):
        """Test launch browser when already running."""
        print("\nTest Started for: test_launch_browser_already_running")
        mock_launch.return_value = "Browser is already running."
        expected_entity_result = "Browser is already running."
        expected_control_result = "Control Object Result: Browser is already running."
        result = self.control.receive_command("launch_browser")

        logging.info(f"Entity Layer Expected: {expected_entity_result}")
        logging.info(f"Entity Layer Received: {mock_launch.return_value}")
            self.assertEqual(mock_launch.return_value, expected_entity_result, "Entity layer assertion
failed.")
        logging.info("Unit Test Passed for entity layer.\n")

        logging.info(f"Control Layer Expected: {expected_control_result}")
        logging.info(f"Control Layer Received: {result}")
        self.assertEqual(result, expected_control_result, "Control layer assertion failed.")
        logging.info("Unit Test Passed for control layer.\n")

    @patch('entity.BrowserEntity.BrowserEntity.launch_browser')
    def test_launch_browser_failure_control(self, mock_launch):
        """Test control layer's handling of the entity layer failure."""
        print("\nTest Started for: test_launch_browser_failure_control")
        mock_launch.side_effect = Exception("Internal error")
        expected_result = "Control Layer Exception: Internal error"
        result = self.control.receive_command("launch_browser")
```

```python
        logging.info(f"Control Layer Expected to Report: {expected_result}")
        logging.info(f"Control Layer Received: {result}")
        self.assertEqual(result, expected_result, "Control layer failed to handle or report the entity error correctly.")
        logging.info("Unit Test Passed for control layer error handling.\n")

    @patch('entity.BrowserEntity.BrowserEntity.launch_browser')
    def test_launch_browser_failure_entity(self, mock_launch):
        """Test failure to launch browser due to an internal error in the entity layer."""
        print("\nTest Started for: test_launch_browser_failure_entity")
        internal_error_message = "Failed to launch browser: Internal error"
        mock_launch.side_effect = Exception(internal_error_message)  # Simulate an exception on error
        expected_control_result = f"Control Layer Exception: {internal_error_message}"

        # Execute command
        result = self.control.receive_command("launch_browser")

        # Check if the control layer returns the correct error message
        logging.info(f"Entity Layer Expected Failure: {internal_error_message}")
        logging.info(f"Control Layer Received: {result}")
        self.assertEqual(result, expected_control_result, "Control layer failed to report entity error correctly.")
        logging.info("Unit Test Passed for entity layer error handling.\n")
```

```
if __name__ == '__main__':
    unittest.main()



--- temporary.py ---
import unittest
from unittest.mock import patch, AsyncMock
import logging
import sys, os, discord, logging, unittest
sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))

# Setup logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

# Import your classes
from control.BrowserControl import BrowserControl

class TestBrowserFunctionality(unittest.TestCase):

    def setUp(self):
        """Set up BrowserControl and context for each test."""
        self.control = BrowserControl()
        self.ctx = AsyncMock()  # Mocking the context to use in the control object

    @patch('entity.BrowserEntity.BrowserEntity.launch_browser')
    def test_launch_browser_failure_entity(self, mock_launch):
        """Test failure to launch browser due to an internal error in the entity layer."""
        internal_error_message = "Failed to launch browser: Internal error"
        mock_launch.side_effect = Exception(internal_error_message)  # Simulate an exception on error
        expected_control_result = f"Control Layer Exception: {internal_error_message}"

        # Execute command
        result = self.control.receive_command("launch_browser")

        # Check if the control layer returns the correct error message
        logging.info(f"Entity Layer Expected Failure: {internal_error_message}")
        logging.info(f"Control Layer Received: {result}")
        self.assertEqual(result, expected_control_result, "Control layer failed to report entity error correctly.")
        logging.info("Unit Test Passed for entity layer error handling.")

if __name__ == '__main__':
    unittest.main()
```

--- test_init.py ---

```
# test_init.py
import sys
import os
import unittest
from unittest.mock import patch, AsyncMock
import logging
```

```
# Ensure all necessary paths are included for modules that tests need to access
sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
# Setting up logging without timestamp
logging.basicConfig(level=logging.INFO, format='%(levelname)s - %(message)s')


# Import your BrowserControl class and any other common classes
from control.BrowserControl import BrowserControl


class BaseTestCase(unittest.TestCase):
    """Base test class that can be extended by other test modules."""


    def setUp(self):
        """Set up BrowserControl and context for each test."""
        self.control = BrowserControl()
        self.ctx = AsyncMock()  # Mocking the context to use in the control object
```