CS 1632 - DELIVERABLE 1: Test Plan and Traceability Matrix

JBefunge

Zachary Whitney (zdw9, MisterZW)

Michael Schropp (mfs35, mfschropp)

Division of requirements and work:

We split requirements between ourselves based on even-odd ordering – that is, Zachary took the odd-numbered requirements and Michael took the even-numbered ones. Specifically, that meant that the following requirements went to each tester:

Zachary: FUN-TEXT-DISPLAY, FUN-FILE-LOADING, FUN-RUN-SPEED, FUN-STOP, FUN-TRACE

Michael: FUN-MENUS, FUN-BEFUNGE, FUN-STEP, FUN-TIME, PERF-EXECUTION-TIME

Before we began writing test plan specifics, we spent some time together generally discussing overall test coverage and which types of tests we planned to write for each requirement. We then wrote out our selected test cases independently. We simply updated the traceability matrix as we added new tests for each requirement. We met again in-person to review our test cases for clarity and coverage, execute the test run, and summarize the results and defects found during the run. We triaged the defects together to select the most interesting and pressing three to report, as well as agreeing on a suitable enhancement.

Strategy:

Like most testers, we informed our test plan design first with some informal exploratory testing to gain a better idea of the system's limits and inner workings. We cross-referenced the requirements with the behavior we were seeing to try to determine emphasis. Even in these early stages, we noticed several defects (Program Area not labeled, Save calling on Save As erroneously). We documented them well enough to make sure that we could reproduce the defects when writing formal test cases later.

After this, we worked through requirements and came up with a battery of simple, minimal tests meant to check the happy path for each requirement and lay down a baseline of simple test coverage. While we figured most of these tests would pass, they are nonetheless critical to possess in a good test plan for completeness. What if we change the software later and this causes a core feature which has always functioned correctly to break?

It was after this point that we began to really try to press up against and cross the boundaries of individual requirements with edge cases. We started *trying* to break the IDE. We focused more heavily on the saving and loading features because a) there's lots that can go wrong in I/O, and b) it's very important that these features work correctly in an editor.

Test Plan:

IDENTIFIER: TEXT_DISPLAY_LAYOUT_TEST

DESCRIPTION:

This is a simple test to make sure the text box layouts are correctly formatted when starting the JBefunge IDE.

PRECONDITIONS:

JBefunge has been properly compiled on the test environment but is not running.

EXECUTION STEPS:

1) Execute the run.sh script in the directory where JBefunge is located to start JBefunge.

POSTCONDITIONS:

a) There shall be three text boxes in the GUI display. One shall be labeled Program Area, one shall be labeled Stack, and one shall be labeled Output.

b) The JBefunge IDE title should read "UNTITLED".

IDENTIFIER: TEXT_DISPLAY_EDITABILITY_TEST

DESCRIPTION:

Ensure that users can edit the Program Area but not the Stack and Output displays in the GUI.

PRECONDITIONS:

a) JBefunge is running.

b) The Program Area, Stack, and Output fields are all blank.

EXECUTION STEPS:

1) Click anywhere in the Program Area text box.

2) Attempt to type "Program Area Test" into the corresponding text field.

3) Click anywhere in the Stack text box.

4) Attempt to type "Stack Test" into the corresponding text field.

5) Click anywhere in the Output text box.

6) Attempt to type "Output Test" into the corresponding text field.

POSTCONDITIONS:

The Program Area text field shall display "Program Area Test" whereas the other two text fields shall remain blank.

IDENTIFIER: DISPLAY_MENUS_TEST

DESCRIPTION:

Test that loads the JBefunge IDE and verifies that the only menu options available are File, Color, and Options.

PRECONDITIONS:

The JBefunge IDE has been downloaded and compiled on the test environment.

EXECUTION STEPS:

1) Execute the run.sh script in the directory where the JBefunge IDE is located.

POSTCONDITIONS:

In the menu bar only File, Color, and Options should be visible.

IDENTIFIER: FILE_MENU_TEST
DESCRIPTION:
    Test that the File menu can be opened and displays four menu options:
    Open File, Save File, Save As, and Quit.
PRECONDITIONS:
    a) The JBefunge IDE has been compiled and is running.
    b) “File” should be visible on the menu bar.
EXECUTION STEPS:
    1) Click on the menu option titled “File”.
POSTCONDITIONS:
    The File menu item displays 4 more menu options: Open File, Save File, Save As, and Quit.

IDENTIFIER: COLOR_MENU_TEST
DESCRIPTION:
    Test that the Color menu can be opened and displays six menu options:
    Red, Yellow, Blue, Pink, Green, and Orange.
PRECONDITIONS:
    a) The JBefunge IDE has been compiled and is running.
    b) “Color should be visible on the menu bar.
EXECUTION STEPS:
    1) Click on the menu option titled “Color”.
    2) Select the color Red.
    3) Repeat steps 1 and 2 for the remaining color options.
POSTCONDITIONS:
    a) The color menu item opens 6 more menu options:
       Red, Yellow, Blue, Pink, Green, and Orange.
    b) After clicking on a color option, that option becomes checked and the previous selected
       become unchecked.

IDENTIFIER: OPTIONS_MENU_TEST
DESCRIPTION:
    Test that the Options menu can be opened and displays two checkable menu options:
    Time Program and Check for End Opcode.
PRECONDITIONS:
    a) The JBefunge IDE has been compiled and is running.
    b) “Options” should be visible on the menu bar.
EXECUTION STEPS:
    1) Click on the menu option titled “Options”.
    2) Click on the Time Program option.
    3) Click on the menu option titled “Options”.
    4) Click on the Check for End Opcode option.

POSTCONDITIONS:

a) The options menu item opens 2 more menu options:
   Time Program and Check for End Opcode.

b) A check mark appears by each menu option, indicating it has been enabled.

IDENTIFIER: SAVE_NEW_FILE_TEST

DESCRIPTION:

Verify that JBefunge can save a file which does not yet exist with the “Save File” command.

PRECONDITIONS:

a) File menu is operational.

b) JBefunge is running and a blank, new (unnamed) file is open.

c) User has write access to the Desktop.

d) No files on the Desktop exist named test.bf.

EXECUTION STEPS:

1) Type “TEST FILE” into the Program Area text field.

2) Click the File menu button.

3) Select the Save file option from the menu.

   Assertion: Save As menu should appear and prompt for a filename.

4) Navigate to the desktop.

5) Type test.bf as the filename.

6) Click Save.

POSTCONDITIONS:

a) Assertion should occur as specified.

b) Title bar for the GUI should now read the Desktop path followed by test.bf
   (i.e., it should no longer read UNTITLED)

c) The user desktop should contain a file named test.bf.

d) test.bf contents on disk should be “TEST FILE” (verify with known working text editor).

IDENTIFIER: SAVE_EXISTING_FILE_TEST

DESCRIPTION:

Verify that JBefunge can save a file which already exists on disk with the “Save File” command.

PRECONDITIONS:

a) File menu is operational.

b) JBefunge is editing a blank file which has already been saved to disk with a known working method (e.g., created by the Unix touch command).

c) User has write permissions for the blank test file.

EXECUTION STEPS:

1) Type “TEST FILE” into the Program Area text field.

2) Click the File menu button.

3) Select the Save File option from the menu.

POSTCONDITIONS:

a) The system should save to the preexisting file without prompting for a filename.
b) File contents on disk should be “TEST FILE” (verify with known working text editor).

IDENTIFIER: SAVE_AS_TEST
DESCRIPTION:

Verify that the Save As correctly prompts the user for a filename and saves data to disk with that filename.

PRECONDITIONS:

a) File menu is operational.
b) JBefunge is running and a blank, new (unnamed) file is open.
c) User has write access to the Desktop.
d) No files on the Desktop exist which are named test1.bf or test2.bf.

EXECUTION STEPS:

1) Type “TEST FILE” into the Program Area text field.
2) Click the File menu button.
3) Select the Save As file option from the menu.
   Assertion: Save As menu should appear and prompt for a filename.
4) Navigate to the Desktop.
5) Type test1.bf as the filename.
6) Click Save.
7) Repeat steps 2-4.
8) Type test2.bf as the filename this time.
9) Click Save.

POSTCONDITIONS:

a) All assertions occur as specified.
b) The JBefunge editor title should list the file path for the new file titled “test2.bf” (i.e., it should not read UNTITLED or test1.bf).
c) The Desktop should contain 2 new files named test1.bf and test2.bf.
d) Both file contents on disk should be “TEST FILE” (verify with known working text editor).

IDENTIFIER: OPEN_FILE_TEST
DESCRIPTION:

Verify that JBefunge faithfully renders file data when opening an existing Befunge file.

PRECONDITIONS:

a) JBefunge has just been run but no other actions have yet been taken.
b) A file called FizzBuzz.bf with known contents exists in a known disk location.

EXECUTION STEPS:

1) Click the File menu button.
2) Click the Open file button.
   Assertion: A file explorer GUI loads which allows the user to browse for the file

path.
3) Navigate to the FizzBuzz.bf file and select it.
4) Click the open button.

POSTCONDITIONS:
a) Assertion should occur as specified.
b) The contents of the file shall be displayed in the Program Area field of the GUI.
c) The Stack and Output shall be blank.
d) The JBefunge editor title shall show the correct file path and name of the FizzBuzz.bf file as its title (i.e., it should no longer be listed as “UNTITLED”).

IDENTIFIER: SAVE_REALLY_LONG_FILENAME_TEST
DESCRIPTION:
This is an edge case test. It verifies that JBefunge appropriately handles user attempts to save a file with an extremely long filename.

PRECONDITIONS:
a) No files exist on the Desktop with a filename of the lowercase alphabet 10x over and a .txt extension.
b) JBefunge is properly compiled and set up in the test environment.
c) The test user has write permission to the Desktop.

EXECUTION STEPS:
1) Execute the run.sh script in the /src directory where the JBefunge IDE is located.
2) Type “SAVE_REALLY_LONG_FILENAME_TEST” in the Program Area.
3) Click the File menu.
4) Click the Save file option.
5) Navigate to the Desktop in the File Explorer window.
6) Type a filename consisting of the entire lowercase alphabet repeated 10 times. Append .txt to the file. You can just copy and paste the alphabet to make this less onerous.
7) Click Save.

POSTCONDITIONS:
The exact expected behavior here is not specified exactly in the requirements since it is reasonable to consider this a likely failure case in many test environments.

Success Conditions:
a) Save feature works as expected – the file is created successfully and the JBefunge IDE updates the IDE title to the new file path.
b) The Operating System rejects the I/O operation because the filename is beyond its maximum allowed length, but the IDE handles the failure gracefully. Specifically, it shall minimally inform the user that an error occurred (ideally it would also specify what caused this error).

Failure Conditions:
a) The save operation fails silently. That is, it appears to succeed, but does not create the file as specified with the correct contents.
b) The IDE crashes, data is corrupted, or something else astonishing occurs which is not a straightforward and expected result from this kind of operation.

IDENTIFIER: PROGRAM_OUTPUT_TEST
DESCRIPTION:
Verify that the JBefunge IDE will show the expected output of a program.
PRECONDITIONS:
a) The JBefunge IDE has been compiled and is running.
b) HelloWorld.bf has been downloaded and opened in the IDE.
EXECUTION STEPS:
1) Execute the program using the “Run” execution option.
POSTCONDITIONS:
After execution completes, “Hello World!” should appear in the Output textbox.

IDENTIFIER: PROGRAM_COUNTER_TEST
DESCRIPTION:
Verify that the JBefunge IDE program counter is accurate.
PRECONDITIONS:
a) The JBefunge IDE has been compiled and is running.
b) FizzBuzz.bf has been downloaded and opened in the IDE.
EXECUTION STEPS:
1) Execute the program using the “Walk” execution option.
2) Observer the cursor as the program executes.
POSTCONDITIONS:
a) When encountering a '^', the program counter should move up.
b) When encountering a 'v', the program counter should move down.
c) When encountering a '>', the program counter should move right.
d) When encountering a '<', the program counter should move left.

IDENTIFIER: STACK_INTEGRITY_TEST
DESCRIPTION:
Verify that that stack is being correctly shown and values are being added and removed from the stack.
PRECONDITIONS:
a) The JBefunge IDE has been compiled and is running.
b) HelloWorld.bf has been downloaded and opened in the IDE.
c) The stack is empty at the beginning.
EXECUTION STEPS:
1) Execute the program using the "Walk" execution option.
2) Observe that the stack is updated by adding and removing values.
POSTCONDITIONS:
a) The stack successfully was shown.
b) The stack was updated and both added and removed values throughout execution.
c) The stack is empty at the end of execution.

IDENTIFIER: INVALID_PROGRAM_TEST
DESCRIPTION:
    This is an edge case test. Verify that the JBefunge IDE cannot execute non-JBefunge files.
PRECONDITIONS:
    a) The JBefunge IDE has been compiled and is running.
    b) README.md has been downloaded and opened in the IDE.
EXECUTION STEPS:
    1) Execute the program using the "Run" execution option.
POSTCONDITIONS:
    Handling of invalid programs is not specified in the requirements.
    Expectation is the IDE handles the execution of invalid programs by failing the execution.

IDENTIFIER: RUN_SPEED_SMELL_TEST
DESCRIPTION:
    Simple smoke test to make sure execution speeds are: run > walk > mosey.
PRECONDITIONS:
    a) JBefunge is running and test file Fizzbuzz.bf is opened but not running.
EXECUTION STEPS:
    1) Click the Run button.
    2) Wait for program to complete execution, making note of the execution speed.
    3) Click the Walk button.
    4) Wait for program to complete execution, making note of the execution speed.
    5) Click the Mosey button.
    6) Wait for program to complete execution, making note of the execution speed.
POSTCONDITIONS:
    a) Run shall take the shortest time to execute.
    b) Mosey shall take the longest time to execute.
    c) Walk shall take more time than Run but less time than Mosey to execute.
    Note: A formal timing is not necessary for this test -- the timing differences must be clearly apparent. If they are not, this test shall be considered failed.

IDENTIFIER: WALK_MOSEY_TIME_COMPARISON_TEST
DESCRIPTION:
    Simple smoke test to make sure execution speeds are: run > walk > mosey.
PRECONDITIONS:
    a) JBefunge is running and test file Fizzbuzz.bf is opened but not running.
    b) Tester has a stopwatch or other external timer to user for testing. It must be possible to press the start button on the timer at the same time as beginning program execution. DO NOT use the timer function in JBefunge as it is part of the software under test!
EXECUTION STEPS:
    1) Simultaneously press the start button of the external timer device and the Walk button in JBefunge.

2) On a best-effort basis, stop the timer when the FizzBuzz program terminates.
3) Record the elapsed time as WALK_TIME.
4) Reset the timer.
5) Simultaneously press the start button of the external timer device and the Mosey button in JBefunge.
6) On a best-effort basis, stop the timer when the FizzBuzz program terminates.
7) Record the elapsed time as MOSEY_TIME.
8) Calculate MOSEY_WALK_RATIO as (MOSEY_TIME / WALK_TIME), rounded to one decimal.

POSTCONDITIONS:

PASSED SCENARIO: 7.5 <= MOSEY_WALK_RATIO <= 12.5

FAILED SCENARIO: The calculated ratio falls outside the specified bounds or the ratio cannot be derived due to some other error or failure.

IDENTIFIER: STEP_THROUGH_TEST

DESCRIPTION:

Verify that the "Step" execution option correctly executes a program one opcode at a time.

PRECONDITIONS:

a) The JBefunge IDE has been compiled and is running.
b) HelloWorld.bf has been downloaded and opened in the IDE.

EXECUTION STEPS:

1) Execute the program using the "Step" execution option.
2) Repeat step 1 until the program has been completed.

POSTCONDITIONS:

a) The program has finished and was executed one opcode at a time.
b) Throughout execution, the program appropriately updated the stack.
c) Throughout execution, the program appropriately updated the output, resulting in "Hello World!".

IDENTIFIER: STOP_BUTTON_DISABLED_TEST

DESCRIPTION:

Verify that stop button is enabled during program execution and disabled before and after execution.

PRECONDITIONS:

JBefunge is running and test file Fizzbuzz.bf is opened but not running.

EXECUTION STEPS:

Assertion: The Stop Button should be disabled prior to executing the program.

1) Execute the program using the Run option.

Assertion: The Stop Button should become enabled during the program's execution.

2) Wait for the program to finish execution.

Assertion: The Stop Button should become disabled after the program's execution.

3) Repeat steps 1 & 2 with the Walk and Mosey execution speeds.

POSTCONDITIONS:
  a) The Stop button should be disabled after all 3 run-speeds have been tested.
  b) All assertions shall have occurred as specified during the test run for each run speed.

IDENTIFIER: STEP_BUTTON_WORKS_TEST
DESCRIPTION:
  Verify that the stop button correctly halts program execution.
PRECONDITIONS:
  a) JBefunge is running and test file Fizzbuzz.bf is opened but not running.
  b) Stop button functionality is enabled during program execution.
EXECUTION STEPS:
  1) Execute the program using the Run option.
  2) Press the Stop button before the program is able to finish execution.
    Assertion: The stack and output areas shall stop changing and there shall be no cursor movement.
  3) Repeat steps 1 & 2 with the Walk and Mosey execution speeds.
POSTCONDITIONS:
  All assertions shall have occurred as specified during the test run for each run speed.

IDENTIFIER: TIME_PROGRAM_ENABLED_TEST
DESCRIPTION:
  Verify that the Time Program menu item in the Options menu results in the execution time being shown.
PRECONDITIONS:
  a) The JBefunge IDE has been compiled and is running.
  b) FizzBuzz.bf has been downloaded and opened in the IDE.
  c) The Time Program menu item has been checked under the Options menu.
EXECUTION STEPS:
  1) Execute the program using the "Run" execution option.
POSTCONDITIONS:
  The system informed the user how long the program took to execute after execution completed.

IDENTIFIER: TIME_PROGRAM_ACCURACY_TEST
DESCRIPTION:
  Verify that the Time Program menu item in the Options menu results in the execution time being shown after the stop button has been hit.
PRECONDITIONS:
  a) The JBefunge IDE has been compiled and is running.
  b) FizzBuzz.bf has been downloaded and opened in the IDE.
  c) The Time Program menu item has been checked under the Options menu.
  d) An external and working stopwatch is available.

EXECUTION STEPS:

1) At the same time, execute the program using the “Run” execution option and start the stopwatch.
2) End the stopwatch when the program has finished.

POSTCONDITIONS:

The execution time given by the IDE will match the execution time on the stopwatch after accounting for some human error.

IDENTIFIER: STOP_PROGRAM_WITH_TIME_PROGRAM_ENABLED_TEST

DESCRIPTION:

Verify that the Time Program menu item in the Options menu results in the execution time being shown after the stop button has been hit.

PRECONDITIONS:

a) The JBefunge IDE has been compiled and is running.
b) FizzBuzz.bf has been downloaded and opened in the IDE.
c) The Time Program menu item has been checked under the Options menu.

EXECUTION STEPS:

1) Execute the program using the "Mosey" execution option.
2) Wait for 5 seconds to pass.
3) Stop the execution using the "Stop" option.

POSTCONDITIONS:

The system informed the user how long the program took to execute after the execution was stopped.

IDENTIFIER: TIME_PROGRAM_LONG_EXECUTION_TEST (EDGE)

DESCRIPTION:

This is an edge case test. Verify that the Time Program option will give the correct execution time of a program that has ran for a long period of time.

PRECONDITIONS:

a) The JBefunge IDE has been compiled and is running.
b) FizzBuff.bf has been downloaded and opened in the IDE.
c) The Time Program menu item has not been checked under the Options menu.

EXECUTION STEPS:

1) Execute the program using the "Mosey" execution option.

POSTCONDITIONS:

Upon program completion, verify that a valid execution time is shown.

IDENTIFIER: TIME_PROGRAM_DISABLED_TEST

DESCRIPTION:

Verify that the program execution time is not shown when the Time Program menu item is not checked.

PRECONDITIONS:

a) The JBefunge IDE has been compiled and is running.

b) FizzBuff.bf has been downloaded and opened in the IDE.
c) The Time Program menu item has not been checked under the Options menu.

EXECUTION STEPS:
1) Execute the program using the "Run" execution option.

POSTCONDITIONS:
Upon program completion, the system will not inform the user of execution time.

IDENTIFIER: CURSOR_DISPLAY_TEST

DESCRIPTION:
Verify that execution cursor displays while program runs and disappears after program termination.

PRECONDITIONS:
JBefunge is running and test file HelloWorld.bf is opened but not running.

EXECUTION STEPS:
Assertion: The cursor should be disabled/not visible prior to executing the program.
1) Execute the program using the Run option.
Assertion: The cursor should become visible and move through the program source in the Program Area during the program's execution.
2) Wait for the program to finish execution.
Assertion: The cursor should become disabled/disappear after the program's execution.
3) Repeat steps 1 & 2 with the Walk and Mosey execution speeds.
4) Repeat steps 1 & 2 by stepping through the program execution.

POSTCONDITIONS:
a) The cursor should be disabled/disappear after each execution speed is tested.
b) All assertions shall have occurred as specified during the test run for each run speed.

IDENTIFIER: CURSOR_COLORS_TEST

DESCRIPTION:
Verify that the cursor displays correctly in all six color options.
Note: This behavior is specified implicitly in the requirements.

PRECONDITIONS:
JBefunge is running and test file HelloWorld.bf is opened but not running.

EXECUTION STEPS:
1) Select the Colors menu with the mouse.
2) Select the Red color option.
3) Click Run.
Assertion: The cursor in the Program Area should appear in the chosen color.
4) Wait for the execution to terminate.
5) Repeat steps 1 through 4 five times, each time substituting one of Yellow, Blue, Pink, Green, and Orange in for Red in Step 2.

POSTCONDITIONS:
Assertion shall have occurred as specified for all six colors.

IDENTIFIER: SWITCH_FILE_STEP_TEST
DESCRIPTION:
Verify that opening a new file during Step execution does not corrupt the trace.
PRECONDITIONS:
a) The JBefunge IDE has been downloaded and compiled on the test environment.
b) The tester has access to test files Fizzbuzz.bf and HelloWorld.bf.
EXECUTION STEPS:
1) Execute the run.sh script in the /src directory where the JBefunge IDE is located.
2) Click the File menu and select the Open file option.
3) Open Fizzbuzz.bf.
4) Click the Step button 30 times, noting the state of the Stack.
5) Click the File menu and select the Open file option.
6) Open HelloWorld.bf.
7) Click the Step button 30 times.
POSTCONDITIONS:
The program shall have highlighted only those opcodes which are being executed by the IDE during the step procedure.

IDENTIFIER: MAC_FIZZBUZZ_PERFORMANCE_TEST
DESCRIPTION:
Verify that the FizzBuzz.bf program will execute using the "Run" execution option in under 30 seconds.
PRECONDITIONS:
a) The JBefunge IDE has been compiled and is running on a machine running MacOS.
b) FizzBuff.bf has been downloaded and opened in the IDE.
c) An external and working stopwatch is available.
EXECUTION STEPS:
1) At the same time, execute the program using the "Run" execution option and start the stopwatch.
2) End the stopwatch when the program has finished.
POSTCONDITIONS:
Upon program completion, the stopwatch will display less than 30 seconds.

IDENTIFIER: LINUX_FIZZBUZZ_PERFORMANCE_TEST
DESCRIPTION:
Verify that the FizzBuzz.bf program will execute using the "Run" execution option in under 30 seconds.
PRECONDITIONS:
a) The JBefunge IDE has been compiled and is running on a machine running Linux based OS.
b) FizzBuff.bf has been downloaded and opened in the IDE.
c) An external and working stopwatch is available.

EXECUTION STEPS:

1) At the same time, execute the program using the "Run" execution option and start the stopwatch.
2) End the stopwatch when the program has finished.

POSTCONDITIONS:

Upon program completion, the stopwatch will display less than 30 seconds.

IDENTIFIER: WINDOWS_FIZZBUZZ_PERFORMANCE_TEST
DESCRIPTION:

Verify that the FizzBuzz.bf program will execute using the "Run" execution option in under 30 seconds.

PRECONDITIONS:

a) The JBefunge IDE has been compiled and is running on a machine running Windows.
b) FizzBuff.bf has been downloaded and opened in the IDE.
c) An external and working stopwatch is available.

EXECUTION STEPS:

1) At the same time, execute the program using the "Run" execution option and start the stopwatch.
2) End the stopwatch when the program has finished.

POSTCONDITIONS:

Upon program completion, the stopwatch will display less than 30 seconds.

IDENTIFIER: MOBILE_FIZZBUZZ_PERFORMANCE_TEST (EDGE)
DESCRIPTION:

Verify that the FizzBuzz.bf program will execute using the "Run" execution option in under 30 seconds.

PRECONDITIONS:

a) The JBefunge IDE has been compiled and is running on a mobile device.
b) FizzBuff.bf has been downloaded and opened in the IDE.
c) An external and working stopwatch is available.

EXECUTION STEPS:

1) At the same time, execute the program using the "Run" execution option and start the stopwatch.
2) End the stopwatch when the program has finished.

POSTCONDITIONS:

Upon program completion, the stopwatch will display less than 30 seconds.

Traceability Matrix for JBefunge Test Plan

FUN-TEXT-DISPLAY:
- TEXT_DISPLAY_LAYOUT_TEST,
- TEXT_DISPLAY_EDITABILITY_TEST

FUN-MENUS:
- DISPLAY_MENUS_TEST,
- FILE_MENU_TEST,
- COLOR_MENU_TEST,
- OPTIONS_MENU_TEST

FUN-FILE-LOADING:
- SAVE_NEW_FILE_TEST,
- SAVE_EXISTING_FILE_TEST,
- SAVE_AS_TEST,
- OPEN_FILE_TEST,
- SAVE_REALLY_LONG_FILENAME_TEST (EDGE)

FUN-BEFUNGE:
- PROGRAM_OUTPUT_TEST,
- PROGRAM_COUNTER_TEST,
- STACK_INTEGRITY_TEST,
- INVALID_PROGRAM_TEST

FUN-RUN-SPEED:
- RUN_SPEED_SMELL_TEST,
- WALK_MOSEY_TIME_COMPARISON_TEST

FUN-STEP:
- STEP_THROUGH_TEST

FUN-STOP:
- STOP_BUTTON_DISABLED_TEST,
- STOP_BUTTON_WORKS_TEST

FUN-TIME:
- TIME_PROGRAM_ENABLED_TEST,
- TIME_PROGRAM_ACCURACY,
- STOP_PROGRAM_WITH_TIME_PROGRAM_ENABLED_TEST,
- TIME_PROGRAM_LONG_EXECUTION_TEST (EDGE),
- TIME_PROGRAM_DISABLED_TEST

FUN-TRACE:
- CURSOR_DISPLAY_TEST,
- CURSOR_COLORS_TEST,
- SWITCH_FILE_STEP_TEST (EDGE)

PERF-EXECUTION-TIME:
- FIZZBUZZ_PERFORMANCE_TEST,
- LINUX_FIZZBUZZ_PERFORMANCE_TEST,
- MAC_FIZZBUZZ_PERFORMANCE_TEST,
- WINDOWS_FIZZBUZZ_PERFORMANCE_TEST,
- MOBILE_FIZZBUZZ_PERFORMANCE_TEST (EDGE)

Test Run Results:
TEXT_DISPLAY_LAYOUT_TEST: FAILED

Reason: Program Area text field is not labeled at all

TEXT_DISPLAY_EDITABILITY_TEST: PASSED

SAVE_NEW_FILE_TEST: PASSED

SAVE_EXISTING_FILE_TEST: FAILED

Reason: When saving an existing file, the Save File feature prompts for a file name, when it should save as the preexisting file name.

SAVE_AS_TEST: PASSED

OPEN_FILE_TEST: FAILED

Reason: When opening a file, the file name is not shown at the top of the text editor.

SAVE_REALLY_LONG_FILENAME_TEST: FAILED

Reason: When saving a file with a long filename, the editor acts as though the save was successful, but no actually has an error when saving. The specific error is "Could not write to file /Users/Michael/Desktop/abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz.txt".

RUN_SPEED_SMELL_TEST: PASSED

WALK_MOSEY_TIME_COMPARISON_TEST: PASSED

Note: MOSEY_WALK_RATIO = 8.6

STOP_BUTTON_DISABLED_TEST: PASSED

STOP_BUTTON_WORKS_TEST: PASSED

CURSOR_DISPLAY_TEST: FAILED

Reason: When executing a program, the cursor is still displayed after execution when using the "Run", "Walk", or "Mosey" execution option. It is not displayed when using "Step" execution option.

CURSOR_COLORS_TEST: PASSED

SWITCH_FILE_STEP_TEST: FAILED

Reason: When executing a program using the "Step" execution option, if you stop in the middle of executing and open a new program, then start to "Step" through that program, it will continue execution of the first program.

DISPLAY_MENUS_TEST: PASSED

FILE_MENU_TEST: PASSED

COLOR_MENU_TEST: PASSED

OPTIONS_MENU_TEST: PASSED

PROGRAM_OUTPUT_TEST: PASSED

PROGRAM_COUNTER_TEST: PASSED

STACK_INTEGRITY_TEST: PASSED

INVALID_PROGRAM_TEST: FAILED

Reason: When encountering an invalid program or non-Befunge file, the IDE does not fail the execution and instead enters an endless loop.

STEP_THROUGH_TEST: PASSED

TIME_PROGRAM_ENABLED_TEST: PASSED

TIME_PROGRAM_ACCURACY_TEST: PASSED

STOP_PROGRAM_WITH_TIME_PROGRAM_ENABLED_TEST: PASSED

TIME_PROGRAM_LONG_EXECUTION_TEST: PASSED

TIME_PROGRAM_DISABLED: PASSED

LINUX_FIZZBUZZ_PERFORMANCE_TEST: PASSED

MAC_FIZZBUZZ_PERFORMANCE_TEST: PASSED

WINDOWS_FIZZBUZZ_PERFORMANCE_TEST: PASSED

MOBILE_FIZZBUZZ_PERFORMANCE_TEST: BLOCKED

Reason: Unknown procedure to configure IDE on mobile device.

Defects:

IDENTIFIER: OPEN_FILE_DURING_EXECUTION_DEFECT
SUMMARY:
The IDE continues execution of the previous program even after a new program is opened.
DESCRIPTION:
If a user opens a new program during the execution of the current program, the current program will continue running. During this time, the cursor will appear on some opcodes, but will also disappear at times, running invisible opcodes that are from the previous program. Even when the cursor is appearing, the opcode it is over is not being run; instead, it is running the opcode that the previous program had placed there. This is a defect as when a new program is opened, it should halt the execution of the previous program and clear the stack, preparing for the execution of the new program.
REPRODUCTION STEPS:
Preconditions:
a) The JBefunge IDE has been downloaded and compiled on the test environment.
b) The tester has access to test files Fizzbuzz.bf and HelloWorld.bf.
Execution:
1) Execute the run.sh script in the /src directory where the JBefunge IDE is located.
2) Click the File menu and select the Open file option.
3) Open Fizzbuzz.bf.
4) Click the Step button 30 times, noting the state of the Stack.
5) Click the File menu and select the Open file option.
6) Open HelloWorld.bf.
7) Click the Step button 30 times.
EXPECTED BEHAVIOR:
While not explicitly specified in the requirements, an expectation is that the IDE will not let you open the file or the program execution will stop. It is likely also expected that the IDE will reset the stack and show the cursor on only the current program.
OBSERVED BEHAVIOR:
The previous program continued execution despite the new programming being opened. The stack was not reset and the cursor would disappear at times, continuing on the execution of the previous program.
SEVERITY:
Moderate – The cursor issue breaks the requirements directly but in a minor way. More problematically, this also breaks a core behavior that is expected out of text editors: we expect isolation from one file to the next.
IMPACT:
User can become confused as they will see a file different from the one being executed.

IDENTIFIER: SAVE_FILE_DEFECT
SUMMARY:

Saving preexisting files triggers Save As functionality instead of simply saving.

DESCRIPTION:

Attempting to use Save File on a preexisting file will prompt the user for a filename as if the Save As button were clicked. This is a defect as when the Save File is used on a preexisting file it should save the file with its current file name.

REPRODUCTION STEPS:

1) Start with JBefunge open with no file loaded.
2) In the program area text field, enter “Test File”.
3) In the “File” menu, hit “Save File”.
5) Enter the file name as “defectTest.bf”.
6) In the program area text field, append “2” to “Test File”.
7) In the “File” menu hit “Save File”.

EXPECTED BEHAVIOR:

The IDE saves the file without prompting for a filename.

OBSERVED BEHAVIOR:

The IDE prompts the user for a filename before saving.

SEVERITY:

Moderate – This is a defect that you can easily work around. That said, it is a very common defect and the user will encounter this defect frequently.

IMPACT:

The IDE will needlessly prompt for a filename for files that already have names.

IDENTIFIER: SAVING_REALLY_LONG_FILENAME_DEFECT
SUMMARY:

Saving exceedingly long filenames will fail without alerting the user.

DESCRIPTION:

When attempting to save filenames that are really long the IDE will not show an error to the user. Instead, the IDE will change the title to the new file path while the file was never actually saved. This is a defect because the IDE should be alerting users when their file was not saved correctly.

REPRODUCTION STEPS:

1) Start with JBefunge open with no file loaded.
2) In the program area text field, enter “Test File”.
3) In the “File” menu, hit “Save File”.
4) Enter the file name as the alphabet 10 times, append with .txt.
5) Click “Save”.

EXPECTED BEHAVIOR:

The file will be successfully saved or the IDE will alert the user that the save failed.

OBSERVED BEHAVIOR:

The IDE changes the title to the new file path despite the file not being saved. It will then only show an error in the command line, where a user will not be looking.

SEVERITY:

Minor – This defect will rarely affect a user and is easy to work around by saving files with reasonable filename lengths. When the defect is encountered, the severity is considerable because losing data is a major shortcoming in a text editor.

IMPACT:

The user may believe their file has been saved when it has not.

Enhancement Request:

JBefunge, like any IDE, is a text editing program first-and-foremost. As such, maintaining data integrity for users is of paramount importance. If I'm worried that I might lose my code by using an IDE, it doesn't matter how good the software is otherwise. No one would want to use it.

JBefunge would benefit from a feature which tracks when unsaved data is modified and alerts users when their actions risk data loss. For example, it would be very helpful to users to ask them if they'd like to save their file changes whenever they attempt to open a new file or exit from the IDE.

Since the requirements don't explicitly address this behavior, it would be an enhancement rather than an explicit defect. That said, such user-friendly features are so commonplace for software like this that they are likely within the realm of behavior a user will anticipate and expect. As such, implementing this may well be of higher priority than fixing minor defects which are less central to the common-case usage of and IDE.