# CS 1632 – Deliverable 4: Performance Testing Using VisualVM

https://github.com/sjd69/CS1632-Deliverable-4

Alejandro Porras

Ajporrasm

Stephen Dowhy

Sjd69

Summary:

At the start of the project, we decided that appropriate Key Performance Indicators (KPI’s) for our improved JBefunge would be:

1. A faster overall execution/runtime for any given JBefunge program (<2000 ms)
2. Improvement of code within a specific method to improve its speed and efficiency (<1500ms)
3. Optimization for reducing CPU workload for any given JBefunge program (<50% CPU Usage)

My partner and I first tackled the secondary KPI in the hopes that a faster and more appropriately optimized method would reduce the overall execution time for JBefunge programs, specifically Fizzbuzz, our main program used during testing. It was decided that our first modification would be to the method convertLocation() inside of MainPanel.java. We were unsure as to how well made the compiler is made for JBefunge so we used a common compiler optimization technique known as ‘loop unrolling’ inside of the convertLocation() method. Upon a single unroll, execution times as well as the method refresh(), were significantly reduced and can be reduced further with additional unrolls for programs of even relatively small size.

Our second modification to the JBefunge code would be to the print() method located inside of MainPanel.java, as we noticed that print sets the JTextArea \_output to an entirely new string, causing the JTextArea to parse and repaint the entire string. We changed the print method to only append the new character to the end of the string rather than creating a new immutable string object which was a concatenation of the prior text area value plus the new character.

Our successful optimizations can be seen in the VisualVM screenshots shown below with the results before and after the changes:

Overall, we conclusively met our desired KPI’s for the project with great success for each indicator as there are many places in which the code can be further optimized. Our approach for the KPI’s did not have a quantifiable value at the time due to little knowledge of what appropriate efficiency values would be, and so chose for our goals to be a significant improvement (>20%) in the goals stated and were effectively achieved.

**MANUAL TEST PLAN**

**IDENTIFIER:** TEST-HIGHLIGHT-END

**TEST CASE:** Tests that end opcode is highlighted at the end of execution.

**PRECONDITIONS:** JDK is properly installed and configured as well as proper compilation and installion of Befunge.

**EXECUTION STEPS:** Run fizzbuzz Befunge program by pasting the program into the text box and clicking run button.

**POSTCONDITIONS:** The end opcode character is highlighted (@).

**IDENTIFIER:** TEST-HIGHLIGHT-BEGINNING

**TEST CASE:** Tests that beginning character is highlighted at the start of execution.

**PRECONDITIONS:** JDK is properly installed and configured as well as proper compilation and installion of Befunge.

**EXECUTION STEPS:** Paste fizzbuzz Befunge program by pasting the program into the text box and clicking the step button.

**POSTCONDITIONS:** The first character is highlighted (0).

**IDENTIFIER:** TEST-INFINITE-LOOP

**TEST CASE: (EDGE CASE)** Tests that characters are highlighted in an infinite loop.

**PRECONDITIONS:** JDK is properly installed and configured as well as proper compilation and installion of Befunge.

**EXECUTION STEPS:** Run program with the code:  
><

Then click the run button.

**POSTCONDITIONS:** The two characters will alternate being highlighted indefinitely or until program is told to stop.

**IDENTIFIER:** TEST-HIGHLIGHT-ONLY-ENDOPCODE

**TEST CASE:** Tests that end opcode is highlighted when it’s the only character.

**PRECONDITIONS:** JDK is properly installed and configured as well as proper compilation and installion of Befunge.

**EXECUTION STEPS:** Input into the text box the code:  
@

Then click the run button.

**POSTCONDITIONS:** The end opcode character is highlighted (@).

Performance Measurements:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Before Changes | Run 1 | Run 2 | Run 3 | Mean | After Changes | Run 1 | Run 2 | Run 3 | Mean |
| Run time | 2140 ms | 2130 ms | 2138 ms | 2136 ms |  | 1704 ms | 1848 ms | 2028 ms | 1860 ms |
| Refresh() time | 1854 ms | 1765 ms | 1900 ms | 1840 ms |  | 1256 ms | 1548 ms | 1660 ms | 1448 ms |
| CPU workload | 44% | 43% | 44% | 44% |  | 44% | 42% | 41% | 42% |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

SCREENSHOTS