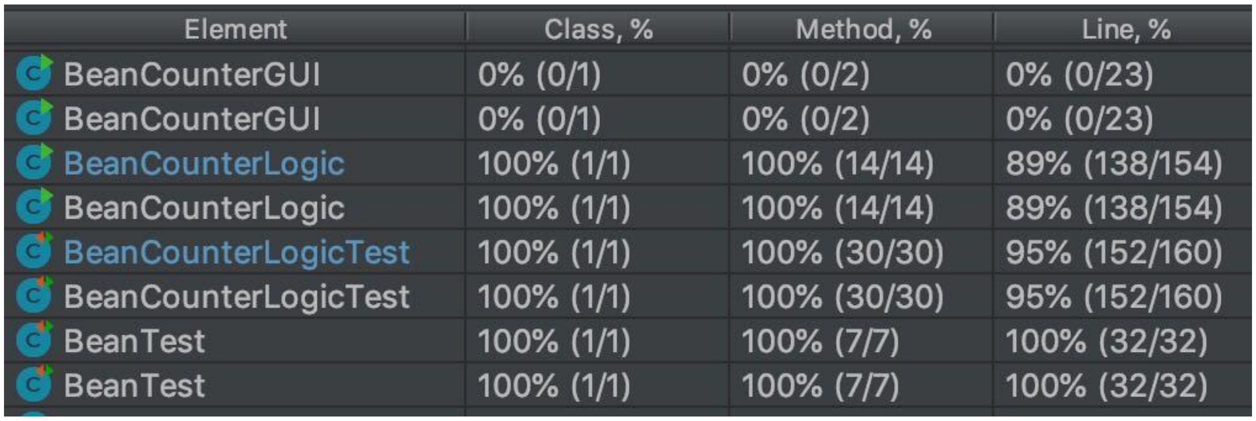
James Pickering, Ali Zaman

CS 1632 - DELIVERABLE 5: End to end testing BeanCounter

TDD is a helpful way of writing tests. The simple concept of TDD is to write and correct the failed tests before writing new code. This helps in avoiding duplication of code as we write a small amount of code at a time in order to pass tests. However, it can be a little cumbersome when you are not able to figure out the method or don't know where to start.

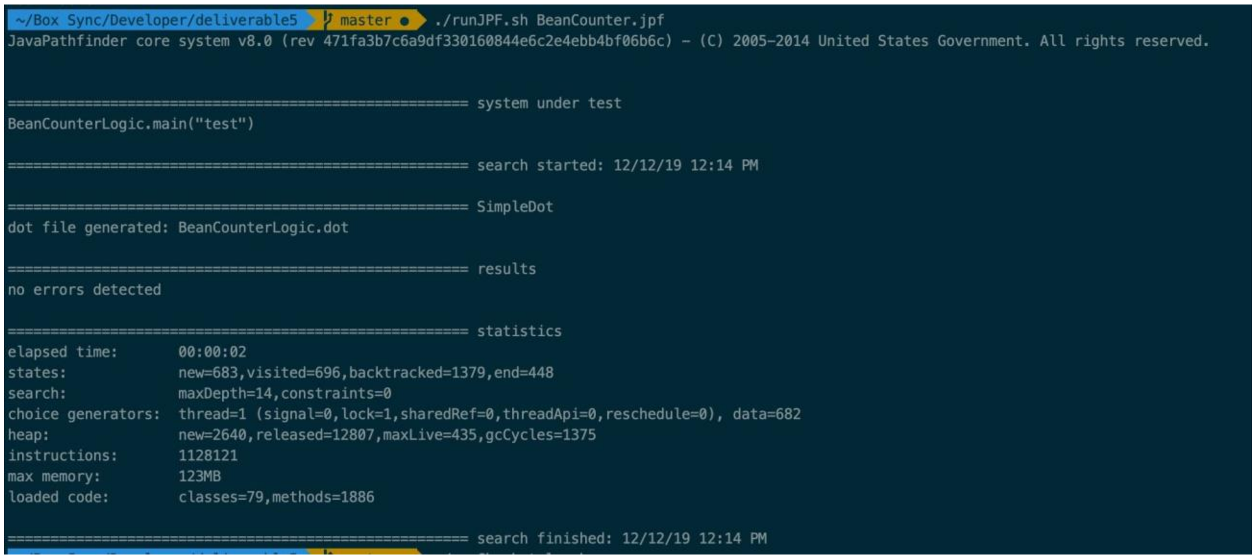
Furthermore, we were able to do end to end testing with the help of model checking, static and manual systems testing at various layers. As code reviews are one of the most helpful coding strategies in industry. Static testing is kind of similar to that, I think it was helpful to read my own code manually because it actually helped me improve my tests and figure out how I can optimize my code to get as much coverage as possible.

Unit Testing



Element	Class, %	Method, %	Line, %
BeanCounterGUI	0% (0/1)	0% (0/2)	0% (0/23)
BeanCounterGUI	0% (0/1)	0% (0/2)	0% (0/23)
BeanCounterLogic	100% (1/1)	100% (14/14)	89% (138/154)
BeanCounterLogic	100% (1/1)	100% (14/14)	89% (138/154)
BeanCounterLogicTest	100% (1/1)	100% (30/30)	95% (152/160)
BeanCounterLogicTest	100% (1/1)	100% (30/30)	95% (152/160)
BeanTest	100% (1/1)	100% (7/7)	100% (32/32)
BeanTest	100% (1/1)	100% (7/7)	100% (32/32)

Model Checking



```
~/Box Sync/Developer/deliverable5  master ●  ./runJPF.sh BeanCounter.jpf
JavaPathfinder core system v8.0 (rev 471fa3b7c6a9df330160844e6c2e4ebb4bf06b6c) - (C) 2005-2014 United States Government. All rights reserved.


====================================================== system under test
BeanCounterLogic.main("test")

====================================================== search started: 12/12/19 12:14 PM

====================================================== SimpleDot
dot file generated: BeanCounterLogic.dot

====================================================== results
no errors detected

====================================================== statistics
elapsed time:       00:00:02
states:             new=683,visited=696,backtracked=1379,end=448
search:             maxDepth=14,constraints=0
choice generators:  thread=1 (signal=0,lock=1,sharedRef=0,threadApi=0,reschedule=0), data=682
heap:               new=2640,released=12807,maxLive=435,gcCycles=1375
instructions:       1128121
max memory:         123MB
loaded code:        classes=79,methods=1886

====================================================== search finished: 12/12/19 12:14 PM
```

Manual testing

Identifier
fun_usage
Test Case
Tests to see that command line argument runs with a number greater than 0 and a second argument of luck or skill
Preconditions
A new iteration must begin
Execution Steps
1. Run java -jar BeanCounter.jar [arg1] [arg2]
2. Arg1 should be greater than 0
3. Arg2 should be luck or skill

Postconditions
You should see a starting window for the game.

Identifier
fun_textui
Test Case
Tests to see that the invoked UI the output is “Slot beans count:” followed by a row of 10 numbers representing 10 slots. The sum of the numbers shall equal to the initial bean count passed into the command line
Preconditions
Bean on top with average being 0 and remaining being 9
Execution Steps
1. Run java -jar BeanCounter.jar 10 skill|luck

Postconditions
Sum of number should equal to the initial bean count passed into command line

Identifier
fun_gui-init
Test Case
Tests to see that GUI consists of the pegs, empty slots, labels and buttons,
Preconditions
A new iteration must begin
Execution Steps
1. Run java -jar BeanCounter.jar 10 skill|luck

Postconditions
When the GUI is invoked, the window shall display 9 rows of pegs in a triangular formation where the top row has 1 peg and the bottom row has 9 pegs. Below the pegs, there shall be 10 empty slots numbered from 0-9. Below the slots, there shall be 8 buttons: "Step", "Slow", "Fast", "Stop", "Lower Half", "Upper Half", "Repeat", "Reset". There shall be a bean above the top row peg, unless the initial bean count is 0. The top right corner shall display two strings: "Average = 0" and "Remaining = <num>", where <num> is the initial bean count minus 1, or 0 if the initial bean count is 0.

Identifier

fun_slots

Test Case

Tests to see that the bottom slots display number of beans in the form of a bar graph

Preconditions

A new iteration

Execution Steps

1. Run java -jar BeanCounter.jar 10 luck|skill
2. Press Fast
3. View the bottom slots

Postconditions

Bar graphs in black color are populated

Identifier

fun_step

Test Case

Tests to see that step button advances bean one step

Preconditions

A new iteration must begin

Execution Steps

1. Run java -jar BeanCounter.jar 10 luck|skill
2. Press step

Postconditions

Bean should move to a second level and another bean appears on the top

Identifier

fun_slow

Test Case

Tests to see that that beans advance in a slow manner

Preconditions

A new iteration must begin

Execution Steps

1. Run java -jar BeanCounter.jar skill|luck
2. Press slow

Postconditions

Populates bar graph at the bottom and remaining beans are zero

Identifier

fun_fast

Test Case

Tests to see that beans advance continuously in a fast manner

Preconditions

A new iteration must begin

Execution Steps

1. Run java -jar BeanCounter.jar
2. Press Fast

Postconditions

Populates bar chart at the bottom with no remaining beans left

Identifier

fun_stop

Test Case

Tests to see that beans are stopped from advancing

Preconditions

A new iteration must begin

Execution Steps

1. Run java -jar BeanCounter.jar 10 luck|skill
2. Press Slow
3. Press stops

Postconditions

The beans should stop advancing from the level you pressed stop

Identifier

fun_lower_half

Test Case

Tests to see that all beans other than lower half are discarded

Preconditions

A new iteration must begin

Execution Steps

3. Run java -jar BeanCounter.jar 10 luck|skill
4. Press Fast
4. Press lower half

Postconditions

Average and bar charts should only output calculation for lower half

Identifier

fun_upper_half

Test Case

Tests to see that all beans other than upper half are discarded

Preconditions

A new iteration must begin

Execution Steps

1. Run java -jar BeanCounter.jar 10 luck|skill
2. Press fast
3. Press upper half

Postconditions

Average and bar charts should output calculate for upper half

Identifier

fun_repeat

Test Case

Tests to see that all beans in flight in the slots are added back to the pool of remaining beans

Preconditions

A new iteration must begin

Execution Steps

1. Run java -jar BeanCounter.jar 10 luck|skill
2. Press fast
3. Press repeat

Postconditions

Remaining beans should be 9

Identifier

fun_reset

Test Case

Tests to see that the machine is set to initial values

Preconditions

A new iteration must begin

Execution Steps

1. Run java -jar BeanCounter.jar 10 luck|skill
2. Press Fast
3. Press Reset

Postconditions

Bean should be sitting on the top and remaining should be 9

Identifier

fun_luck

Test Case

Tests the luck mode, bean should have an equal chance of falling left or right

Preconditions

A new iteration must begin

Execution Steps

1. Run java -jar BeanCounter.jar 10 luck
2. Press step continuously until remaining is 0

Postconditions

Remaining should be zero and average should vary, while you will be able to notice the random falling of beans in GUI

Identifier

fun_skilll

Test Case

Tests the skill mode, bean shall be assigned a skill level 0 to 9

Preconditions

A new iteration must begin

Execution Steps

1. Run java -jar BeanCounter.jar 10 skill
2. Press fast

Postconditions

Remaining should be zero and average should be 4.9

Traceability Matrix

FUN_USAGE: fun_usage

FUN_TEXTUI: fun_textui

FUN_GUI_INIT: fun_gui-nit

FUN_SLOTS: fun_slots

FUN_STEP: fun_step

FUN_SLOW: fun_slow

FUN_FAST: fun_fast

FUN_STOP: fun_stop

FUN_LOWER_HALF: fun_lower_half

FUN_UPPER_HALF: fun_upper_half

FUN_REPEAT: fun_repeat

FUN_RESET: fun_reset

FUN_LUCK: fun_luck

FUN_SKILL: fun_skill