

## **Assignment 3: Mutation Testing**

### **List of Files Modified**

- Test cases modified (for PIT)
  - <https://github.com/UBC-TestingCourse/group4/tree/master/src/test/java/org/jpacman/test/framework/model/GameTest.java>
- Test cases modified (for Major)
  - <https://github.com/UBC-TestingCourse/group4/tree/master/src/test/java/org/jpacman/test/framework/model/GameTest.java>
  - <https://github.com/UBC-TestingCourse/group4/blob/master/src/test/java/org/jpacman/test/framework/model/BoardWithinBordersTest.java>
  - <https://github.com/UBC-TestingCourse/group4/blob/master/src/test/java/org/jpacman/test/framework/model/PointManagerTest.java>
- Files generated from Major:
  - <https://github.com/UBC-TestingCourse/group4/tree/master/major-results/final-analysis>
  - <https://github.com/UBC-TestingCourse/group4/tree/master/major-results/initial-analysis>
  - <https://github.com/UBC-TestingCourse/group4/tree/master/major-results/after-withinBordersTest>
- Files generated from PIT:
  - [https://github.com/UBC-TestingCourse/group4/tree/master/pit-results/PIT\\_Exercise\\_3.2\\_3.3\\_Report](https://github.com/UBC-TestingCourse/group4/tree/master/pit-results/PIT_Exercise_3.2_3.3_Report)
  - [https://github.com/UBC-TestingCourse/group4/tree/master/pit-results/PIT\\_Exercise\\_3.4\\_Report](https://github.com/UBC-TestingCourse/group4/tree/master/pit-results/PIT_Exercise_3.4_Report)

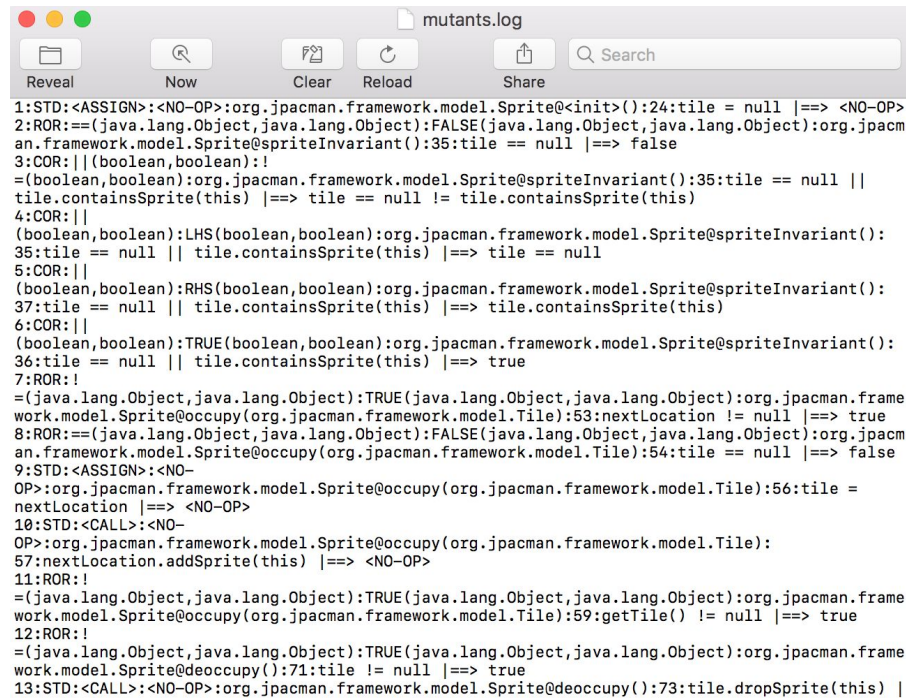
### **Exercise 3.1.**

#### **Major**

##### *(1) Ease of use*

Major is a difficult tool to set up. It supports only a much older build tool, Apache Ant. It does provide some command options to directly mutate simple, standalone Java files, but the easiest method of using Major is to use build using Ant. Users cannot simply copy and paste from the documentation, as the examples provided are very trivial. They need to adapt the recommended settings from the documentation according to their project settings, so the setup process is not very straightforward. After initial setup, however, Major is not too difficult to use, as it can be triggered from the command line. Users can specify the mutation operators they want to use and what the scope of the mutation should be in a mml file.

The logs and reports generated by Major can be read, but understanding the results can be difficult, because information is dispersed between several different files. Some of these files are also not standalone and references other files. The information the mutants generated is especially difficult to see. Figure 1.1 shows the information provided on each mutant generated by Major in the mutants.log file.



```

1:STD:<ASSIGN>:<NO-OP>:org.jpacman.framework.model.Sprite@<init>():24:tile = null |==> <NO-OP>
2:ROR:==(java.lang.Object,java.lang.Object):FALSE(java.lang.Object,java.lang.Object):org.jpacman.framework.model.Sprite@spriteInvariant():35:tile = null |==> false
3:COR:|| (boolean,boolean):!
=(boolean,boolean):org.jpacman.framework.model.Sprite@spriteInvariant():35:tile = null ||
tile.containsSprite(this) |==> tile == null != tile.containsSprite(this)
4:COR:||
(boolean,boolean):LHS(boolean,boolean):org.jpacman.framework.model.Sprite@spriteInvariant():
35:tile == null || tile.containsSprite(this) |==> tile == null
5:COR:||
(boolean,boolean):RHS(boolean,boolean):org.jpacman.framework.model.Sprite@spriteInvariant():
37:tile == null || tile.containsSprite(this) |==> tile.containsSprite(this)
6:COR:||
(boolean,boolean):TRUE(boolean,boolean):org.jpacman.framework.model.Sprite@spriteInvariant():
36:tile == null || tile.containsSprite(this) |==> true
7:ROR:!
=(java.lang.Object,java.lang.Object):TRUE(java.lang.Object,java.lang.Object):org.jpacman.framework.model.Sprite@occupy(org.jpacman.framework.model.Tile):53:nextLocation != null |==> true
8:ROR:==(java.lang.Object,java.lang.Object):FALSE(java.lang.Object,java.lang.Object):org.jpacman.framework.model.Sprite@occupy(org.jpacman.framework.model.Tile):54:tile == null |==> false
9:STD:<ASSIGN>:<NO-OP>:org.jpacman.framework.model.Sprite@occupy(org.jpacman.framework.model.Tile):56:tile = nextLocation |==> <NO-OP>
10:STD:<CALL>:<NO-OP>:org.jpacman.framework.model.Sprite@occupy(org.jpacman.framework.model.Tile):
57:nextLocation.addSprite(this) |==> <NO-OP>
11:ROR:!
=(java.lang.Object,java.lang.Object):TRUE(java.lang.Object,java.lang.Object):org.jpacman.framework.model.Sprite@occupy(org.jpacman.framework.model.Tile):59:getTile() != null |==> true
12:ROR:!
=(java.lang.Object,java.lang.Object):TRUE(java.lang.Object,java.lang.Object):org.jpacman.framework.model.Sprite@deoccupy():71:tile != null |==> true
13:STD:<CALL>:<NO-OP>:org.jpacman.framework.model.Sprite@deoccupy():73:tile.dropSprite(this) |

```

Figure 1.1. mutants.log file

For example, the killed mutant file only refers to mutants by their number in the mutants.log file, so a user needs to check against mutant.log to determine which mutants have been killed, which are live. Figure 1.2 shows a small screenshot of killed.csv, which lists out the mutants that are killed and briefly list why they are killed (exception, assertion failure, or timeout).

MutantNo	[FAIL   TIME   EXC   LIVE]
1	LIVE
9	EXC
10	EXC
13	FAIL
14	FAIL
15	EXC
17	EXC
19	EXC
21	FAIL
23	EXC
26	EXC
28	LIVE
29	FAIL
30	LIVE
31	FAIL
32	FAIL
33	FAIL
34	LIVE

Figure 1.2. Sample killed.csv file.

To conduct this assessment, we followed the documentation provided by Major to mutate a non-trivial test suite, JPacman. The setup process was not at all straightforward.

## (2) Set of mutation operators

Major provides a large set of mutation operators. These statements include changes in logical operators, replacement of unary operators, and replacing of expression values. We retrieved this information from *The Major Mutation Framework* documentation.

Table 3.1. below summarizes several of the operators that are replaced by Major.

Operator	Replacement Type	Description
AOR	Arithmetic Operator Replacement	Replaces mathematical operators such as +, -, *, /
LOR	Logical Operator Replacement	Replaces bitwise operators with other possible bitwise operators. For example, the ^ operator can be swapped with the   operator
COR	Conditional Operator Replacement	Replaces the    and && with the other for condition statements
ROR	Relational Operator Replacement	Changes relational values in the code, often in conditions. For example, > may be changed to >=

SOR	Shift Operator Replacement	Replaces a shift statement (say >>) with a shift in the other direction (say <<)
ORU	Operator Replacement Unary	Replaces operators with a single operand with another unary operator. This includes increments and decrements, or swaps from +ve to -ve

Table 3.1. Arithmetic/Logical Operator Replacements Supported by Major

Major can also perform scalar value replacements and statement deletions to mutate the program. This includes actual numerical values, booleans, strings, return expressions, and deleting statements. Table 3.2. below lists out these scalar value mutations.

Operator	Replacement Type	Description
EVR	Expression Value Replacement	Takes expressions, such as return statements or assignments, and replaces them with default values (e.g. initializing to 0 or an empty string)
LVR	Literal Value Replacement	Takes a literal (number, boolean, or string) and replaces it with specific values. For numbers, this tends to be 0, a negative value, and a positive value. Booleans are replaced True -> False and False -> True. A String is replaced by an empty string
STD	Statement Deletion	Removing statements from the code, including return, break, continue, method calls, assignments, pre/post increment/decrement

Table 3.2. Scalar Value Replacements Supported by Major

Users may also define their own mutation scripts to reduce the number of mutants generated by Major. Users can also specify the types of operators they would like to use using the command line options, or use a wildcard "ALL" to generate all possible mutations.

### *(3) Mutation testing strategy and effectiveness*

The testing strategy used by Major supports a large number of operators. If users do not like using so many operators, they may specify this either through the command line or

in a mML script. By supporting so many operands and a wide range of mutations, Major is highly effective and can catch a lot of weaknesses in a test suite. However, addressing all the mutants that were not killed by the test suite and not an equivalent mutant is actually very difficult using Major. Due to the large number of operands, if users use all mutation operands supported by mutant, there would be often too many live mutants to address.

We determined this by running Major on a moderate-sized program. JPacman is a fairly small project with a non-trivial code base. But Major generated 984 mutants for the entire project. Users can specify the scope and types of mutants to generate, but this implies that mutations generated by Major can be difficult for developers to resolve.

## **PIT**

### *(1) Ease of use*

PIT is a relatively easy tool to set up. It supports several build tools such as Apache Ant, Gradle, and Maven, and provides simple mutation tests for Java files.

In the case of using Maven, user could easily set up PIT by adding the plugin into the pom.xml and user could then easily run PIT with a single command through the command line. The resulting test coverage report includes statement coverage and mutation coverage and it is written in a index.html file. User could also modify the pom.xml file for specifying several settings such as the different types of mutation operations, the number of mutation tests, and the files to be tested.

Overall, PIT provides a simple yet easy to follow documentation that includes how to set up their tool and the different number of mutation operations that you could do with PIT.

To assess this, we followed the documentation provided by PIT and run the existing pom.xml setup provided in JPacman. The setup process is really simple and straightforward.

### *(2) Set of mutation operators*

PIT provides several mutation operators which some of them are not enabled by default. The table below shows the different set of mutation operators available in PIT as mentioned in the PIT website documentation.

Operator	Description
CONDITIONALS_BOUNDARY	Replaces the relational operators <, <=, >, >= with their boundary counterpart.
INCREMENTS	Replace increments with decrements and vice versa.
INVERT_NEGS	Inverts negation of integer and floating point numbers.
MATH	Replaces binary arithmetic operations for either integer or floating-point arithmetic with another operation.
NEGATE_CONDITIONALS	Mutate all conditionals by negating and creating its conditional counterpart.
RETURN_VALS	Mutates the return values of method calls.
VOID_METHOD_CALLS	Removes method calls to void methods.

Table 3.3. Activated by default Mutation Operators Supported by PIT

Operator	Description
CONSTRUCTOR_CALLS	Replaces constructor calls with null values.
INLINE_CONSTS	Mutates inline constants. An inline constant is a literal value assigned to a non-final variable.
NON_VOID_METHOD_CALLS	Removes method calls to non void methods.
REMOVE_CONDITIONALS	Remove all conditionals statements such that the guarded statements always execute.

EXPERIMENTAL_MEMBER_VARIABLE	Removing assignments to member variables.
EXPERIMENTAL_SWITCH	Finds the first label within a switch statement that differs from the default label and mutates the switch statement by replacing the default label (wherever it is used) with this label.

Table 3.4. Deactivated by default Mutation Operators Supported by PIT

### *(3) Mutation testing strategy and effectiveness*

The testing strategy used by PIT supports 13 different mutation operators. Users are able to easily specify the operators that they want to use by specifying it on the pom.xml file. Furthermore, the test reports that are generated by PIT are easily understandable which make addressing all the mutants that were not killed by the test suite and not an equivalent mutant is relatively easy using PIT.

In general, PIT is quite effective in testing small to medium size project due to its ease of use and fast performance. PIT is not really effective for larger size projects that require a more complex and thorough testing because PIT only allows simple mutation operators and does not support more complex mutation operations.

## **Tool Selection**

For general usages, we decided to select PIT. PIT is quick and easy to set up, and produces an intuitive and readable analysis that allows developers to quickly address mutations. It is not nearly as thorough as Major is with the mutations it generates (Major has many more mutation operators), but for general use, PIT is sufficient.

Major shines when a lot of mutations are needed. However, setting up Major is costly, since the documentation does not explain more complex project setups. The files generated by Major are also difficult to understand, so fixing mutations generated by Major would be a costly endeavor. As mutation testing is only one type of coverage, the effort required to learn to use Major might not be necessary.

### Exercise 3.2.

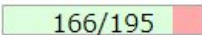
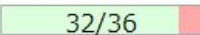
We modified the default settings provided in the pom.xml file by adding a new target class, Food.java, and a new mutation operator, INLINE\_CONSTS, in the pom.xml. As we want to use this same test suite for Major, which does not support assertions at runtime, we disabled the assertions option for PIT. When using this tool, we used our assignment 2 code, but commented out any tests that relied on assertions being enabled.

In total, PIT generated 36 mutations for 6 classes. These classes are: DefaultGameFactory, Sprite, Board, PointManager, Food, and Game. These classes include 5 target test files, which are: FactoryIntegrationTest, SpriteTest, PointManagerTest, BoardTileAtTest, and GameTest.

Of the 36 generated mutants, we were able to kill **32** of these mutants simply using our existing test suite. This means that our mutation coverage was around 86%. The mutant coverage for each classes is shown in the figures below.

## Pit Test Coverage Report

### Project Summary

Number of Classes	Line Coverage	Mutation Coverage
6	85%  166/195	89%  32/36

### Breakdown by Package

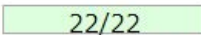
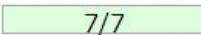
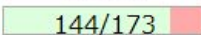
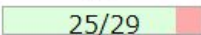
Name	Number of Classes	Line Coverage	Mutation Coverage
<a href="#">org.jpacman.framework.factory</a>	1	100%  22/22	100%  7/7
<a href="#">org.jpacman.framework.model</a>	5	83%  144/173	86%  25/29

Figure 3.2.1. Overall PIT Coverage Report



# Pit Test Coverage Report

## Package Summary

org.jpacman.framework.factory

Number of Classes	Line Coverage	Mutation Coverage
1	100% <div><div>22/22</div></div>	100% <div><div>7/7</div></div>

### Breakdown by Class

Name	Line Coverage	Mutation Coverage
<a href="#">DefaultGameFactory.java</a>	100% <div><div>22/22</div></div>	100% <div><div>7/7</div></div>

Figure 3.2.2. org.jpacman.framework.factory package PIT Coverage Report

# Pit Test Coverage Report

## Package Summary

org.jpacman.framework.model

Number of Classes	Line Coverage	Mutation Coverage
5	83% <div><div>144/173</div></div>	86% <div><div>25/29</div></div>

### Breakdown by Class

Name	Line Coverage	Mutation Coverage
<a href="#">Board.java</a>	75% <div><div>45/60</div></div>	57% <div><div>4/7</div></div>
<a href="#">Food.java</a>	67% <div><div>4/6</div></div>	100% <div><div>3/3</div></div>
<a href="#">Game.java</a>	90% <div><div>53/59</div></div>	86% <div><div>6/7</div></div>
<a href="#">PointManager.java</a>	90% <div><div>18/20</div></div>	100% <div><div>7/7</div></div>
<a href="#">Sprite.java</a>	86% <div><div>24/28</div></div>	100% <div><div>5/5</div></div>

Figure 3.2.3. org.jpacman.framework.model package PIT Coverage Report

### Exercise 3.3.

There were 4 mutants that were not killed by our existing test suite. 3 of these are from Board.java and 1 from Game.java.

The mutants not killed in Board.java are caused by a lack of test coverage on the withinBorders and onBoardMessage methods. For the withinBorders method, PIT mutated the method by negating the conditionals. (For example, `x >= 0 && x < width` was changed to `!(x >= 0 && x < width)`) and also substituted the value of 1 with 0. For the onBoardMessage method, PIT mutates the return Object value of the method by returning with null and throw a `java.lang.RuntimeException` if the unmutated method would return null. The last mutant involved PIT negating the conditional `!getPlayer().isAlive()` to `getPlayer().isAlive()`. This was not killed in Game.java, because there are no tests that covered the `died()` method which contains this conditional.

### Mutations

<a href="#">27</a>	negated conditional : KILLED -> org.jpacman.test.framework.factory.FactoryIntegrationTest.testFullMap (org.jpacman.test.framework.factory.FactoryIntegrationTest)
<a href="#">63</a>	replaced return of integer sized value with (x == 0 ? 1 : 0) : KILLED -> org.jpacman.test.framework.model.BoardTileAtTest.testTileAtDirection[0] (org.jpacman.test.framework.model.BoardTileAtTest)
<a href="#">87</a>	Substituted 1 with 0 : NO_COVERAGE
<a href="#">88</a>	negated conditional : NO_COVERAGE
<a href="#">103</a>	negated conditional : KILLED -> org.jpacman.test.framework.factory.FactoryIntegrationTest.testFullMap (org.jpacman.test.framework.factory.FactoryIntegrationTest)
<a href="#">157</a>	Replaced integer modulus with multiplication : KILLED -> org.jpacman.test.framework.model.BoardTileAtTest.testTileAtDirection[0] (org.jpacman.test.framework.model.BoardTileAtTest)
<a href="#">183</a>	mutated return of Object value for org.jpacman.framework.model.Board::onBoardMessage to ( if (x != null) null else throw new RuntimeException ) : NO_COVERAGE

Figure 3.3.1. Board.java mutations report

## Mutations

<a href="#">59</a>	removed call to org.jpacman/framework/model/Player::deoccupy : KILLED -> org.jpacman.test.framework.model.GameTest.movePlayerInGame(org.jpacman.test.framework.model.GameTest)
<a href="#">75</a>	removed call to org.jpacman/framework/model/PointManager::consumePointsOnBoard : KILLED -> org.jpacman.test.framework.model.GameTest.testC5_PlayerMovesToFood(org.jpacman.test.framework.model.GameTest)
<a href="#">97</a>	negated conditional : KILLED -> org.jpacman.test.framework.model.GameTest.testC1b_GhostMovesToEmpty(org.jpacman.test.framework.model.GameTest)
<a href="#">114</a>	replaced return of integer sized value with (x == 0 ? 1 : 0) : KILLED -> org.jpacman.test.framework.model.GameTest.movePlayerInGame(org.jpacman.test.framework.model.GameTest)
	negated conditional : KILLED -> org.jpacman.test.framework.model.GameTest.moveToWall(org.jpacman.test.framework.model.GameTest)
<a href="#">170</a>	removed call to org.jpacman/framework/model/Game::setChanged : KILLED -> org.jpacman.test.framework.model.GameTest.testObserverAfterGhostMove(org.jpacman.test.framework.model.GameTest)
<a href="#">198</a>	negated conditional : NO_COVERAGE

Figure 3.3.2. Game.java mutations report

### Exercise 3.4.

In order to kill the 2 non-equivalent mutants that were not killed in the `withinBorders` method in `Board.java`, we extended our test suite. The mutants were not killed because there was a lack of test coverage. As `Board.java` contains test cases for `withinBorders`, we added it to our test targets and found that the existing tests were sufficient to kill those mutants.

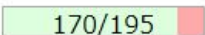
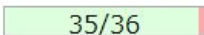
We were not able to kill the remaining mutant, because that last mutant was never executed. The mutant is located in the `onBoardMessage` method of `Board.java`, but this is a private method that is called only by assert statements in `Board.java`. As mentioned in section 3.2, we disabled assertions, so there is no way to execute and kill this mutant.

Our last non-equivalent mutant is located in `Game.java`. As mentioned in sections 3.3., PIT mutated the `died()` method by negating the conditional it returned, so we added a new test case in `GameTest.java` to test the `died()` method. We asserted that method returns true if the player died. This killed the remaining mutant.

Overall, by extending our test suite we manage to kill the **35 out of the 36** mutants that PIT generated which give us a total mutation coverage of **97%**. The last mutant that we are not able to kill is caused by an untestable private method that is only used in assert statements. The new overall mutant coverage and coverage for each class are provided by the figures below.

## Pit Test Coverage Report

### Project Summary

Number of Classes	Line Coverage	Mutation Coverage
6	87%  170/195	97%  35/36

### Breakdown by Package

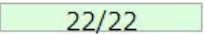
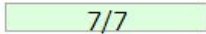
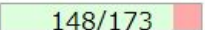
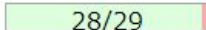
Name	Number of Classes	Line Coverage	Mutation Coverage
<a href="#">org.jpacman.framework.factory</a>	1	100%  22/22	100%  7/7
<a href="#">org.jpacman.framework.model</a>	5	86%  148/173	97%  28/29

Figure 3.4.1. New overall PIT Coverage Report

# Pit Test Coverage Report

## Package Summary

org.jpacman.framework.factory

Number of Classes	Line Coverage	Mutation Coverage
1	100% <div><div>22/22</div></div>	100% <div><div>7/7</div></div>

## Breakdown by Class

Name	Line Coverage	Mutation Coverage
<a href="#">DefaultGameFactory.java</a>	100% <div><div>22/22</div></div>	100% <div><div>7/7</div></div>

Figure 3.4.2. New org.jpacman.framework.factory package PIT Coverage Report

# Pit Test Coverage Report

## Package Summary

org.jpacman.framework.model

Number of Classes	Line Coverage	Mutation Coverage
5	86% <div><div>148/173</div></div>	97% <div><div>28/29</div></div>

## Breakdown by Class

Name	Line Coverage	Mutation Coverage
<a href="#">Board.java</a>	80% <div><div>48/60</div></div>	86% <div><div>6/7</div></div>
<a href="#">Food.java</a>	67% <div><div>4/6</div></div>	100% <div><div>3/3</div></div>
<a href="#">Game.java</a>	92% <div><div>54/59</div></div>	100% <div><div>7/7</div></div>
<a href="#">PointManager.java</a>	90% <div><div>18/20</div></div>	100% <div><div>7/7</div></div>
<a href="#">Sprite.java</a>	86% <div><div>24/28</div></div>	100% <div><div>5/5</div></div>

Figure 3.4.3. New org.jpacman.framework.model package PIT Coverage Report



### Exercise 4.5.

Using the test suite we used for PIT, we were able to generate **174** mutants. We managed to kill **75** of these mutants, but only **23** mutants are live. This is because a large number of mutants were actually not executed by our test code, giving us a mutation score of only **43.10%**. If we only consider executed mutants, then that score is higher at **76.53%**. In other words, we actually managed to kill a fair number of the mutants we ran. A significant number of mutants, however, were never executed. This implies that our test coverage is either rather low, or there is a lot of dead code. Further investigation showed that the reason that many mutants were not executed was because there are many runtime assertions in the codebase. For example, mutants 2-6 are generated in the `spriteInvariant` method of `Sprite.java`. As Major does not support assertions in the source code, `spriteInvariant`, which is only ever called as part of an `assert` statement never gets executed.

```
[junit] MAJOR: Summary:
[junit] MAJOR:
[junit] MAJOR: Analysis time: 4.1 seconds
[junit] MAJOR: Mutation score: 43.10% (76.53%)
[junit] MAJOR: Mutants killed / live: 75 (55-20-0) / 23
[junit] MAJOR: Mutant executions: 114
[junit] MAJOR: -----
[junit] MAJOR: Export summary of results (to summary.csv)
[junit] MAJOR: Export run-time results (to results.csv)
[junit] MAJOR: Export mutant kill details (to killed.csv)
```

Figure 4.5.1. Output from running Major on the test suite we used for PIT

The files generated by Major by running our test suite for PIT are available at:

<https://github.com/UBC-TestingCourse/group4/tree/master/major-results/initial-analysis>

To extend the test suite, we focus only on the live mutants reported by Major, so we did not attempt to increase the mutation coverage. As many of these live mutants are related to the `withinBorders` method of `Board.java`, we first extended our test suite by added `BoardWithinBordersTest.java`. This resulted in only **12** live tests.

```
[junit] MAJOR: Summary:
[junit] MAJOR:
[junit] MAJOR: Analysis time: 4.2 seconds
[junit] MAJOR: Mutation score: 49.43% (87.76%)
[junit] MAJOR: Mutants killed / live: 86 (66-20-0) / 12
[junit] MAJOR: Mutant executions: 131
[junit] MAJOR: -----
[junit] MAJOR: Export summary of results (to summary.csv)
[junit] MAJOR: Export run-time results (to results.csv)
[junit] MAJOR: Export mutant kill details (to killed.csv)
```

Figure 4.5.2. Output from running Major with `BoardWithinBordersTest.java` from Asgn 2

This addition increased the number of mutants executed, but still left us with fewer live mutants than we started. From there, we extended the test suite to kill any non equivalent mutants.

We include the files generated by Major by running our PIT test suite + BoardWithinBordersTest at:

<https://github.com/UBC-TestingCourse/group4/tree/master/major-results/after-withinBordersTest>

We found that 5 of the mutants are equivalent. They are summarized as follows:

#	File & Line	Change	Reason for Equivalence
1	Sprite.java Line 24	Removed line 24 in Sprite.java	The line originally initializes tile to null. Tile is null by default even without this initialization.
69	Board.java Line 26	<i>for (int x = 0; <b>x != w</b>; x++)</i>	The original value is <b>x &lt; w</b> . Since x is initialized locally, we can never get x > w. A board of size w < 0 is impossible to construct due, as we will get an array out of bounds exception.
72	Board.java Line 27	for (int y = 0; <b>y != h</b> ; y++)	The original value is <b>y &lt; h</b> . Since y is initialized locally, we can never get y > h. A board of size y < 0 is impossible to construct due, as we will get an array out of bounds exception.
106	Board.java Line 88	Replaced line with: <b>x &gt;= 0 == x &lt; width</b>	The original value was <b>x &gt;= 0 &amp;&amp; x &lt; width</b> . The failing condition is to have a value where x < 0 and x > width. But because width > 0 in any board, we cannot construct such a board.
138	Board.java Line 157	int result = ((current + delta) + <b>max</b> + max) % max;	Originally, instead of being added, the operation was (current + delta) % <b>max</b> . This actually is mathematically equivalent.

Table 4.5.1. Equivalent mutants generated by Major

The remaining 7 mutants could be killed by an extended test suite, so we modified several test classes to kill them. The table below summarizes the fix for each mutant.

#	File & Line	Original Code	Mutation	Fix Location
28	Game.java Line 55	if (thePlayer.isAlive() && tileCanBeOccupied(target))	if (thePlayer.isAlive() == tileCanBeOccupied(target))	GameTest.java; testMoveDeadPlayerToWall
30	Game.java Line 55	tileCanBeOccupied(target) && thePlayer.isAlive()	tileCanBeOccupied(target) ( <b>Delete thePlayer.isAlive()</b> )	GameTest.java; testMoveDeadPlayer

34	Game.java Line 59	thePlayer.deOccupy ()	Line is deleted	GameTest.java; movePlayerInGame
103	Board.java Line 88	$x \geq 0 \ \&\& \ x < \textit{width}$	$x \geq 0 \ \&\& \ x \neq \textit{width}$	BoardWithinBordersTest.java TestWithinBorders Line 67
117	Board.java Line 89	$\&\& \ y \geq 0 \ \&\& \ y < \textit{height}$	$\&\& \ y \geq 0 \ \&\& \ y \neq \textit{height}$	BoardWithinBordersTest.java TestWithinBorders Line 69
120	Board.java Line 88-89	$x \geq 0 \ \&\& \ x < \textit{width}$ $\&\& \ y \geq 0 \ \&\& \ y < \textit{height}$	$(x \geq 0 \ \&\& \ x < \textit{width} \ \&\& \ y \geq 0) == y < \textit{height}$	BoardWithinBordersTest.java TestWithinBorders Line 70
173	PointManager.java Line 60	pointsEarned == pointsPutOnBoard	pointsEarned >= pointsPutOnBoard	PointManagerTest.java testEatMoreFoodThanAvailable

Table 4.5.2. Mutants generated by Major that we fixed

We re-ran the extended test suite using Major and confirmed that only the 5 equivalent mutants were not killed. The files generated by Major are available at:

<https://github.com/UBC-TestingCourse/group4/tree/master/major-results/final-analysis>