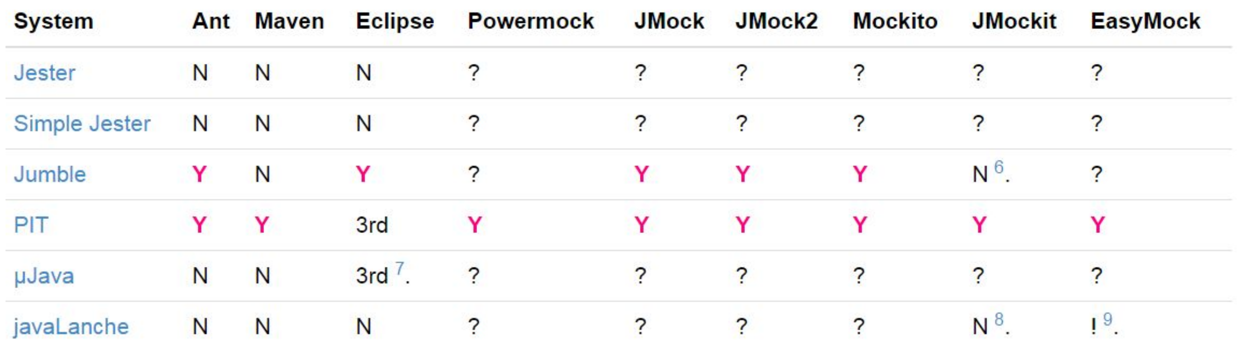
Assignment 3

Kaibo Ma (32400129) -> kiddo122
David Henderson (19475128) -> D-H
Matthew Piaseczny (32993131) -> mpiasek

3.1

a) **Ease of Use:**

i) **Pit:** Pit is much easier to set up. You do not require to set up a database for where the results go like in javalanche. Instead it outputs everything to an html document that you can open up with any web browser. Also Pitest has more documentation on it to access that javalanche. Also Pit is much more compatible with with testing tools when compared to Javalanche.



System	Ant	Maven	Eclipse	Powermock	JMock	JMock2	Mockito	JMockit	EasyMock
Jester	N	N	N	?	?	?	?	?	?
Simple Jester	N	N	N	?	?	?	?	?	?
Jumble	Y	N	Y	?	Y	Y	Y	N [6].	?
PIT	Y	Y	3rd	Y	Y	Y	Y	Y	Y
µJava	N	N	3rd [7].	?	?	?	?	?	?
javaLanche	N	N	N	?	?	?	?	N [8].	! [9].

ii) **Javalanche:** Javalanche is out of date as their github repository's last commit was over 5 years ago. It is also very difficult to install and led to much frustration. The fact that there are separate systems like the database means it is more prone to configuration errors.

b) **Set of Mutation Operators Supported:**

i) **Pit:**

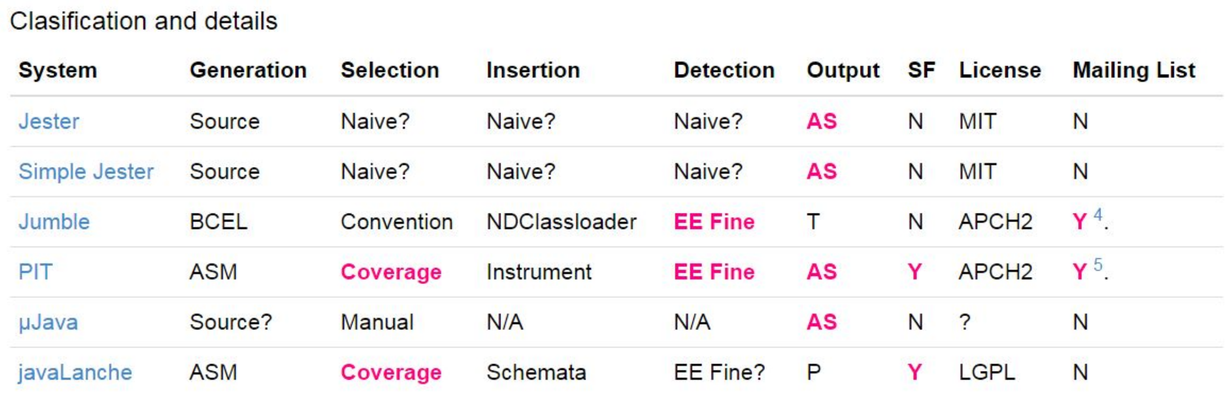
1) Conditionals Boundary Mutator
2) Increments Mutator
3) Invert Negatives Mutator
4) Math Mutator
5) Negate Conditionals Mutator
6) Return Values Mutator
7) Void Method Calls Mutator
8) Constructor Calls Mutator
9) Inline Constant Mutator
10) Remove Conditionals Mutator
11) Experimental Member Variable Mutator
12) Experimental Switch Mutator
13) Non Void Method Calls Mutator

ii) **Javalanche:**

1) negate jump condition
2) Replace numerical constant
3) Replace arithmetic operator,

4) Omit method calls

c) Mutation Testing Strategy and Effectiveness in Mutation Testing:



Clasification and details

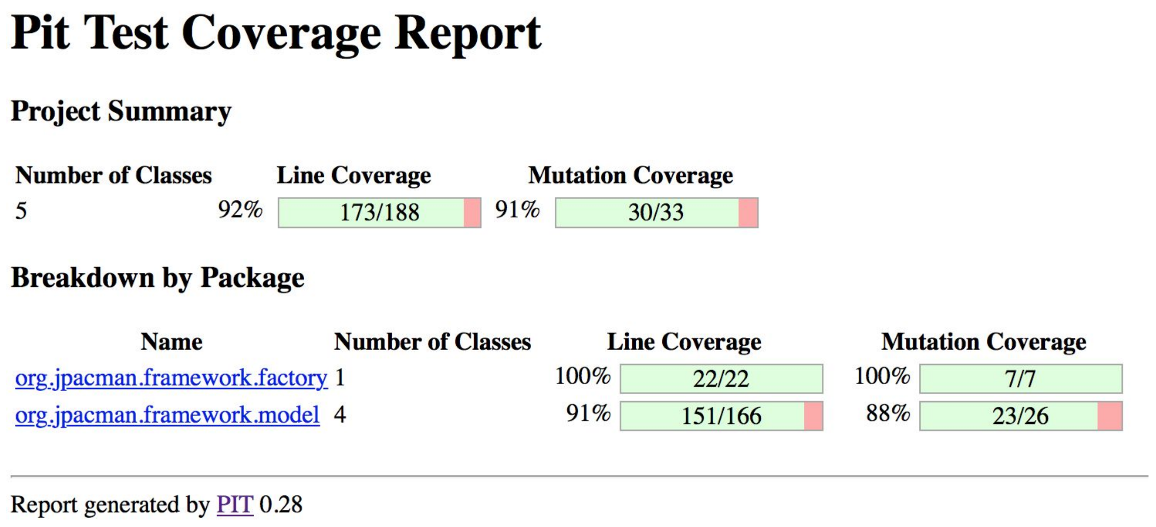
System	Generation	Selection	Insertion	Detection	Output	SF	License	Mailing List
Jester	Source	Naive?	Naive?	Naive?	**AS**	N	MIT	N
Simple Jester	Source	Naive?	Naive?	Naive?	**AS**	N	MIT	N
Jumble	BCEL	Convention	NDClassloader	**EE Fine**	T	N	APCH2	**Y** [4].
PIT	ASM	**Coverage**	Instrument	**EE Fine**	**AS**	**Y**	APCH2	**Y** [5].
µJava	Source?	Manual	N/A	N/A	**AS**	N	?	N
javaLanche	ASM	**Coverage**	Schemata	EE Fine?	P	**Y**	LGPL	N

Both Pit and Javalanche change the byte code to incorporate the mutants into the code. This is done because it faster than changing the actual source code. They also take in account the code coverage of each test when deciding what tests should catch a mutant. This makes the system more informative since it can specify which tests were responsible for catching each mutant. They also share Early Exit Fine as their detection method.

They differ in how they perform the insertion of the mutants. PIT does instrument which is where it creates copies different copies of the class each with a different mutant.and holds them in memory. It will use a different copy of the class every time it needs a different mutant. Javalanche creates a single large class file written onto do disk that has all the mutants inserted into it. This will be slower since something must be written to disk.

Another major difference between Javalanche and PIT is how they output their findings. PIT annotates the source code to highlight where the mutant did not die. This has the advantage of being clear in showing where the test cases fail. Javalanche only prettifies the output which looks good if you are not familiar with the source code but gives you less information.

3.2



Pit Test Coverage Report

Project Summary

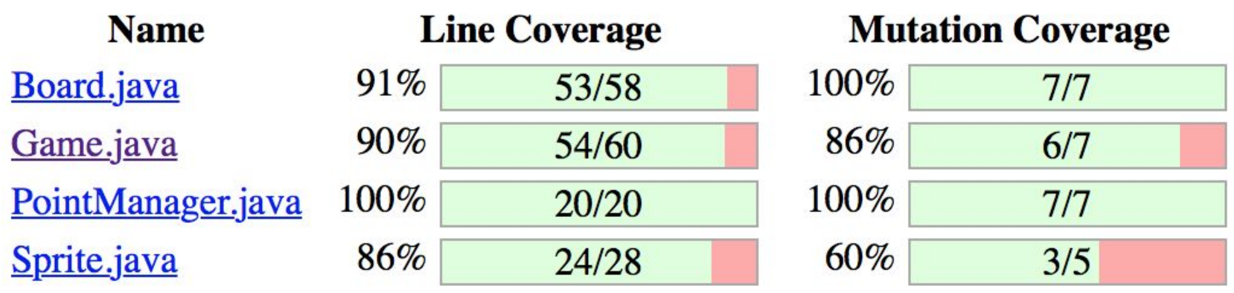
Number of Classes	Line Coverage		Mutation Coverage	
5	92%	173/188	91%	30/33

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
org.jpacman.framework.factory	1	100%	22/22	100%	7/7
org.jpacman.framework.model	4	91%	151/166	88%	23/26

Report generated by PIT 0.28

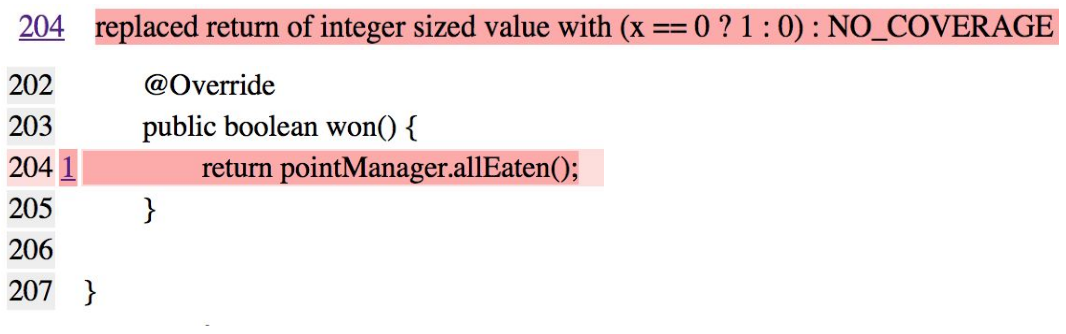
3.3



Name	Line Coverage		Mutation Coverage	
Board.java	91%	53/58	100%	7/7
Game.java	90%	54/60	86%	6/7
PointManager.java	100%	20/20	100%	7/7
Sprite.java	86%	24/28	60%	3/5

In Game.java, there is one Mutator that is not killed and in Sprite.java there are 2 Mutators not killed.

Game.java:

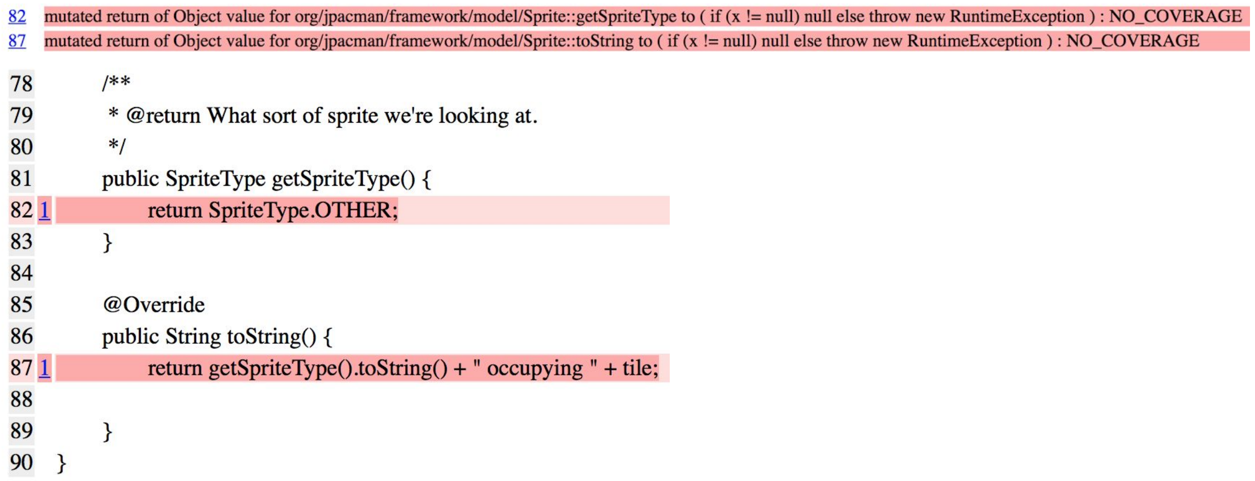


```
204  replaced return of integer sized value with (x == 0 ? 1 : 0) : NO_COVERAGE

202      @Override
203      public boolean won() {
204 1        return pointManager.allEaten();
205      }
206
207  }
```

In the Won() function in Game.java, the test cases do not cover the mutator that changes the boolean value for allEaten(). This mutator is not covered nor killed.

Sprite.java:



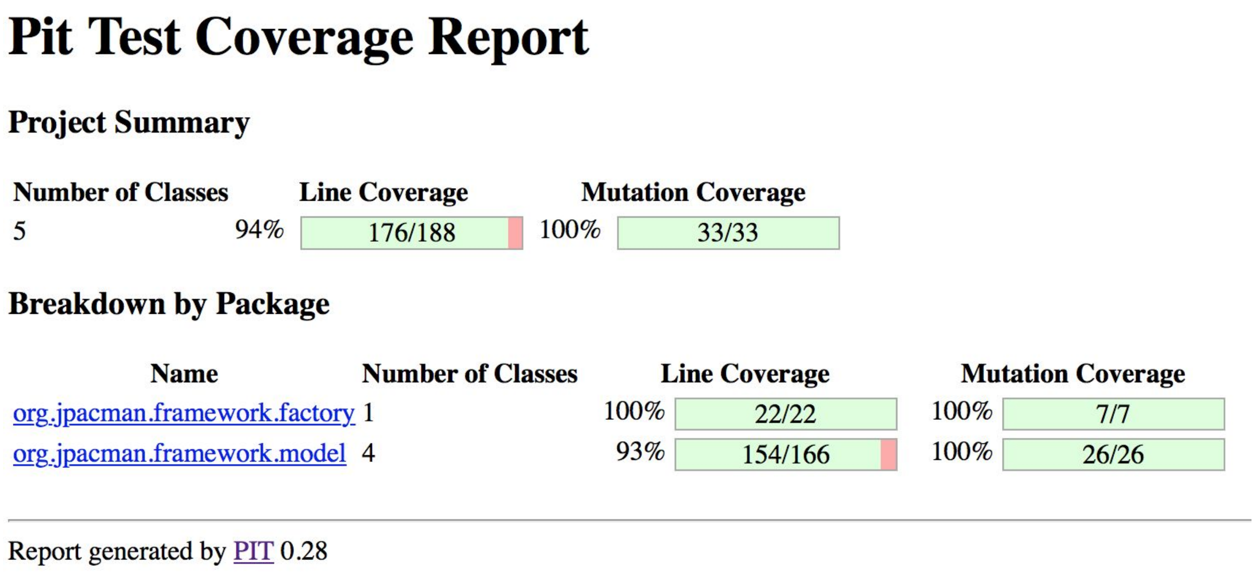
```
82 mutated return of Object value for org/jpacman/framework/model/Sprite::getSpriteType to ( if (x != null) null else throw new RuntimeException ) : NO_COVERAGE
87 mutated return of Object value for org/jpacman/framework/model/Sprite::toString to ( if (x != null) null else throw new RuntimeException ) : NO_COVERAGE
```

```
78        /**
79         * @return What sort of sprite we're looking at.
80         */
81        public SpriteType getSpriteType() {
82 1          return SpriteType.OTHER;
83        }
84
85        @Override
86        public String toString() {
87 1          return getSpriteType().toString() + " occupying " + tile;
88
89        }
90    }
```

getSpriteType(), and toString() are never tested inside SpritTest.java. These two functions have a RETURN_VALS_MUTATOR as the null cases were not checked.

3.4

Our new mutation score is 100%



Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
5	94%	176/188	100%	33/33

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
org.jpacman.framework.factory	1	100%	22/22	100%	7/7
org.jpacman.framework.model	4	93%	154/166	100%	26/26

Report generated by PIT 0.28