Chethan Swaroop
Assignment 1

MONKEY

1. For each tool, generate tests for the appropriate applications. Measure the time taken to generate tests. How many tests does it generate in that duration?

   I ran the monkey tool on the two applications namely MovieGuide and LeafPic.

   The activity required android emulator to setup target devices. I created "Nexus 5X API 26" as virtual device for testing both applications. In the device terminal, I could execute the monkey commands using adb shell prefix. The following commands were executed to induce testing for 2000 events for both the applications respectively.

   ```
   adb shell monkey -p com.esoxjem.movieguide -v 2000
   ```

   ```
   adb shell monkey -p org.horaapps.leafpic.debug -v 2000
   ```

   For the MovieGuide, tests were executed approximately in 12039ms and for LeafPic 2000 events' testing clocked 14263ms.

2. Is the test coverage adequate?

   Code Coverage looked adequate. The events did not follow any specific sequence until the seed value is specified as part of the command. When I checked the application's sequence of operations specific to a package, monkey covered the sequence in depth and touched most the user interface events of the package. startActivity () call used as part of `--pct-appswitch <percent> option` can be helpful for this context.

3. Do the tools find bugs in the applications? You can artificially create bugs to determine if the tools find the bugs.

   I introduced a change not to display the movie title in the MovieGuide application. When I ran the monkey tool, observed that movie title was not displayed appropriately during the testing and also monkey aborted once due to a system crash. I have attached the screenshot where the movie title is unmeaningful.

   But subsequent testing did not report any crashes with the same code change. Screenshots of the testing included as part of the document.

4. Are the tools easy to use?

   The monkey which is part of android studio is very easy to use. There is good documentation about the options to include as part of commands.

5. What are the limitations of the tools?

   For specific instance of system crash, I was not able to reproduce the same issue using monkey stress testing. Probably, the good idea to record/trace such behaviour is to enable debugging option --hprof which would generate profiling reports.

6. How would you improve these tools? Identify at least one improvement for each tool that you believe would be most useful for developers.

   The stress testing tools like monkey are useful to identify the superficial bugs that are exposed as part of black-box testing. One improvement could be that such tools can be a little intelligent in learning about the program behaviour during stress testing by observing the output for the corresponding inputs. This can help in exposing much deeper issues and deeper code coverage for specific set of inputs for which behaviour of the program is vulnerable.

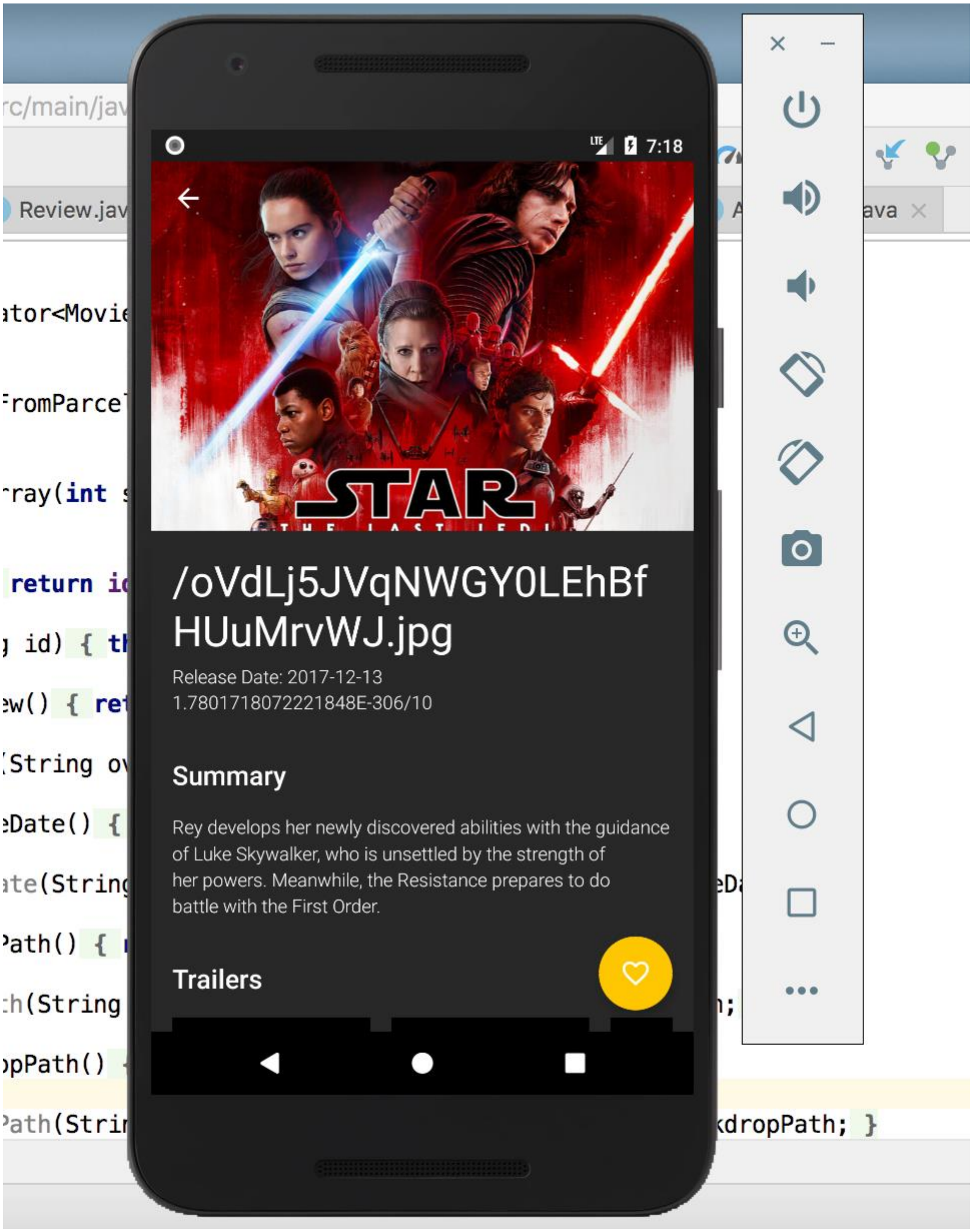
Output captured for MovieGuide

```
Events injected: 2000
: Sending rotation degree=0, persist=false
: Dropped: keys=2 pointers=0 trackballs=0 flips=0 rotations=0
## Network stats: elapsed time=12039ms
// Monkey finished
```

```
** Monkey aborted due to error.
Events injected: 73
: Sending rotation degree=0, persist=false
: Dropped: keys=0 pointers=0 trackballs=0 flips=0 rotations=0
## Network stats: elapsed time=709ms
** System appears to have crashed at event 73 of 2000 using seed 1530701134445
```

Output captured for LeafPic

```
: Sending rotation degree=0, persist=false
: Dropped: keys=2 pointers=0 trackballs=0 flips=0 rotations=0
## Network stats: elapsed time=14263ms
```



/oVdLj5JVqNWGY0LEhBfHUuMrvWJ.jpg

Release Date: 2017-12-13
1.7801718072221848E-306/10

Summary

Rey develops her newly discovered abilities with the guidance of Luke Skywalker, who is unsettled by the strength of her powers. Meanwhile, the Resistance prepares to do battle with the First Order.

Trailers

Chethan Swaroop
Assignment 1

RANDOOP

1. For each tool, generate tests for the appropriate applications. Measure the time taken to generate tests. For example, Randoop uses a time limit to generate tests. How many tests does it generate in that duration?

   I used Randoop plugin which was shared by my batchmate Aditya Barimar.Plugin has been upgraded from version 3.1.5 to v4.0.4. The plugin is not in central repository and had to be deployed in the local machine before using it.

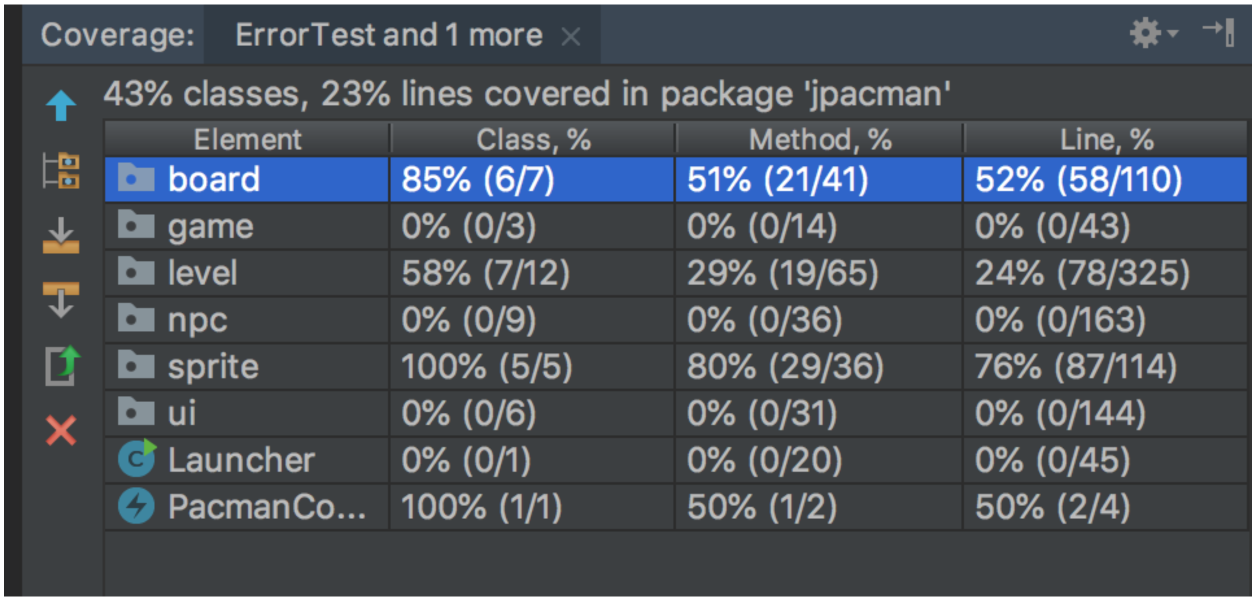
   JPacman:
   Three packages of JPacman were tested with Randoop with a time limit of 30 seconds.
   4 Error revealing tests and 27 Regression tests were generated. The package details are listed in the test coverage section.

   ActiveMQ:
   One package “activemq-jms-pool” was tested with Randoop with a time limit of 30 seconds. 3 Error revealing tests and 984 Regression tests were generated.

2. Is the test coverage adequate?

   JPacman:

   Randoop generated the test cases for three packages namely
   1. nl.tudeleft.jpacman.board
   2. nl.tudeleft.jpacman.level
   3. nl.tudeleft.jpacman.sprite

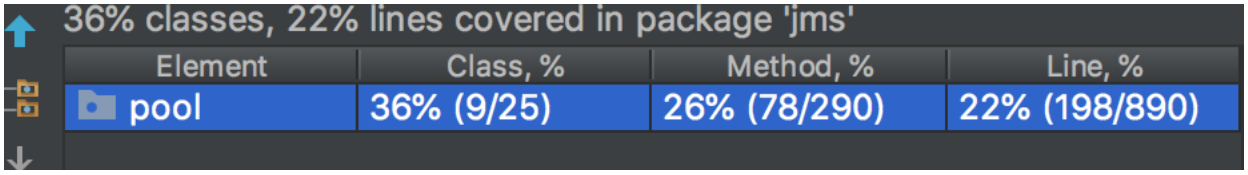


43% classes, 23% lines covered in package 'jpacman'

Element	Class, %	Method, %	Line, %
board	85% (6/7)	51% (21/41)	52% (58/110)
game	0% (0/3)	0% (0/14)	0% (0/43)
level	58% (7/12)	29% (19/65)	24% (78/325)
npc	0% (0/9)	0% (0/36)	0% (0/163)
sprite	100% (5/5)	80% (29/36)	76% (87/114)
ui	0% (0/6)	0% (0/31)	0% (0/144)
Launcher	0% (0/1)	0% (0/20)	0% (0/45)
PacmanCo...	100% (1/1)	50% (1/2)	50% (2/4)

Class coverage is relatively better than the method and line coverage in the three packages.

ActiveMQ:



36% classes, 22% lines covered in package 'jms'

Element	Class, %	Method, %	Line, %
pool	36% (9/25)	26% (78/290)	22% (198/890)

The coverage is less since there are many interdependencies between packages and ActiveMQ is a very large application.

3. Do the tools find bugs in the applications? You can artificially create bugs to determine if the tools find the bugs.

JPacman:

I used Randoop's CheckRep for defining an invariant in nl.tudeleft.jpacman.level package. In the Class *Pellet,* a public method *checkPelletValue ()* returning a Boolean is used as CheckRep.

This method holds the property that value of Pellet should be a positive integer for which any violation would result in failing test.

```
@CheckRep
public boolean checkPelletValue(){
   if(value > 0) return true;
   else return false;
}
```

Following are the logs for which the specification for expected code behaviour has been violated and test case has failed.

[INFO] ANNOTATION: Detected @CheckRep-annotated method "public boolean nl.tudelft.jpacman.level.Pellet.checkPelletValue()". Will use it to check rep invariant of class nl.tudelft.jpacman.level.Pellet during generation.

Failed tests:

ErrorTest0.test1:22 Representation invariant failed: Check rep invariant (method checkPelletValue) for pellet8

Apart from the CheckRep, there have been array out-of-bounds errors observed under failing tests.

ErrorTest0.test2:104 » ArrayIndexOutOfBounds 100

ErrorTest0.test3:190 » ArrayIndexOutOfBounds -1

ActiveMQ:

Once again, I used Randoop's CheckRep annotation for specifying the code behaviour in one of the classes *PooledConnectionFactory*.

The maximum idle time for the *ConnectionsPool* is always 1. The following public method defines the invariant required for the CheckRep.

```
@CheckRep
public boolean maxIdleNotequalto1() {
   if (connectionsPool != null && connectionsPool.getMaxIdlePerKey() != 1){
         return false;
      }
   return true;
}
```

Code change was required to instrument a change in the behaviour of idletime where, I have deliberately changed it to 2.

```
this.connectionsPool.setMaxIdlePerKey(2);// This has to be 1 always
```

Following are the logs which indicate the annotation being registered, condition being violated and subsequent test failures.

[INFO] ANNOTATION: Detected @CheckRep-annotated method "public boolean org.apache.activemq.jms.pool.PooledConnectionFactory.maxIdleNotequalto1()". Will use it to check rep invariant of class org.apache.activemq.jms.pool.PooledConnectionFactory during generation.

Failed tests:
 ErrorTest0.test1:18 Representation invariant failed: Check rep invariant (method maxIdleNotequalto1) for pooledConnectionFactory0
 ErrorTest0.test2:28 Representation invariant failed: Check rep invariant (method maxIdleNotequalto1) for pooledConnectionFactory2
 ErrorTest0.test3:38 Representation invariant failed: Check rep invariant (method maxIdleNotequalto1) for pooledConnectionFactory0

Tests run: 1044, Failures: 3, Errors: 0, Skipped: 0

4. Are the tools easy to use?

   Randoop is relatively easy using command line interface if the application is not large. The plugin provided as part of the documentation is very old and is not compatible. For bigger applications which require maven builds, it becomes a little tedious for new users like me to use Randoop.

5. What are the limitations of the tools?

   Randoop helps in creating tests for the range of the input values and is more of mutation testing for specified classes with primitive data types. Functional testing is still missing, and I am not sure about Randoop's usage for extension of user defined data. Functional testing could be over-expectation from any testing tool until the context of the application has be understood for testing.

6. How would you improve these tools? Identify at least one improvement for each tool that you believe would be most useful for developers.

   While using Randoop on ActiveMQ, it did not generate the test cases for few classes chosen. More investigation is required in terms of adding more classes to analyse for meaningful test cases. It would be helpful if Randoop can suggest why or on what basis the test cases failed to generate. It could mention on what dependencies it failed to generate the tests to help the users to understand if the users are not developers. (Context - Like us, where we are not able to comprehend the context of the ActiveMQ which is a very large package.)