Bug Fixing and Automated Software Testing
- ISE, Assignment 1, Semester 1, 2014 -

Word Count: 3032

by Konrad Janica a1194898

Executing Instructions:

SimtersectionV1.1.jar (found in Simtersection_bugfixed.zip - "/sim/") - Ensure "res" is in directory.

Windows with Java (1.7 or higher) installed: Simply double clicking the .jar file will execute the file.

Linux and most other Unix-based systems (or Windows command prompt) with Java installed:

Using the correct directory, in terminal execute the command "java -jar SimtersectionV1.1.jar"

test.class/test.java (found in Simtersection_bugfixed.zip - "/testHarness/" AND SVN - "/svn/a1194898/2014/s1/ise/assign1/testHarness/")

Linux and most other Unix-based systems (or Windows command prompt) with Java (1.7 or higher) installed:

Using the correct directory, in terminal execute the command "java test"

Introduction:

The propose of this assignment is to apply the strategies learnt from ISE lectures/tutorials to debug and revise a project from a past years Software Engineering Group Project 1A course (2012). By implementing code revisions, an effectual knowledge of programming must be demonstrated as well as the ability to recognize possible improvements to code structure, quality and design.

Summary of Changes:

Bug(s) found include:

i) With Taxi rank checkbox enabled, stopped taxis were not dropping off and/or picking up passengers randomly as required.

ii) Taxis were only spawning North of Frome Road and only with the intention of turning left onto East of North Terrace.

iii) A quit prompt user interface was found in the "TODO:" comments of the code. As per original version, the quit button immediately terminated the program.

iv) In method "public void removePedestrian(Pedestrian p)" located in PedestrianLane.java, the index variable was initialized as 0 and not -1 which could cause problems with thrown exception catching.

Outline of Bug fixes:

i)

1) Referenced both the North-Western and North-Eastern PedestrianLanes into the taxiRank class by altering its constructor and then constructing appropriately.

2) With a new overloaded method in PedestrianLane (addPedestrian(double x, double y, double direction)), new Pedestrian Entities are created within the taxiRank tick during taxi STOP_TIME.

3) A new Rectangle2d object was created and tied to the taxiRank tick to stop Pedestrians entering the loadingArea and hence swapping the now Passengers into a new passengerQueue arrayList.

4) Using appropriate if/else conditions, Passengers are controlled to move/redirect and/or be removed as required.

Note(s):

Taxi STOP_TIME was changed to allow Passengers more time to enter/exit the TaxiRank area.

Several new public methods were created in PedestrianLane to allow for control from the TaxiRank class.

One new public method was added to Pedestrians to be able to transfer PedestrianLanes.

ii)

1) The code that dictates the Taxi spawning probabilities (TAXI_PROB) can be found in AddCarEvent.java

2) All non-exception catching Taxi probabilities was tweaked to a number suitable to eliminate possible congestion from the extra inflow.

iii)

1) Found the signal responsible for the quit button in BottomPanel.java

2) Created a new invisible pop up window once quit button is clicked. This pop-up window then creates a quit prompt using JOptionPane.showConfirmDialog with options that execute appropriately.

3) Found the main JFrame in SimWindow.java and similarly to (2), created a pop-window with the same options triggered by the top right X button.

iv)

1) The index variable was changed from 0 to -1 to allow the if/else statement to catch thrown exceptions on particular cases.

Overview of Software Testing and Bug Fixing Process:

1) Method(s) of Bug Detection

i) Knowing the intended operation for the Taxis from the Assignment instructions, this bug was found simply by running the original .jar file with the "Taxi Rank" checkbox enabled and observing:

a) The Pedestrian Entities never stopped to wait for a taxi.
b) After prolonged program runtime it became obvious there was no random passengers exiting from the Taxis.

Once this bug was initially determined by GUI inspection, the source code was then examined to find the implementation of the Taxi Rank. The Taxi stopping logic was traced down to try and find methods connected to spawning, queuing or removing Pedestrians. With no such implementations in place, the bug was confirmed.

ii) Similar to the method in (i), this bug was again found graphically. After prolonged program runtime, it became evident that Taxis were either not spawning anywhere except North of Frome Road or the probabilities were sent to negligible values.

The source code was then traced down to the AddCarEvent class which was responsible for determining which Vehicle Entity Type would spawn. The taxi probability (TAXI_PROB) in this method was originally set to 0 for all roads except Frome Road, hence the bug was confirmed.

iii) The comments flagged as "TODO:" were searched to find unfinished bugs. The only bug of this category found was an unimplemented quit prompt which was intended to prompt the user upon receiving a signal from the "quit" button on the GUI. Once discovered, bug was tested in the program runtime by manually pressing the button and observing whether or not a prompt appeared.

iv) This bug was found during the fixing of bug (i). Was logically determined to be a bug because the thrown exception catching mechanism was incorrectly set and there will never be an arrayList with index "-1".

2) Use of lectures and other resources

For this assignment I managed to use the information gathered from lectures/tutorials and apply it in a realistic programming situation. Using a combination of validation testing and defect testing, I was able to obtain the relevant information to find the above mentioned bugs and ensure their fix operates as the developer originally intended.

Software inspections were essential in discovering problems within the system. Software testing allowed for the fixes to be properly verified.

Tutorial 1 in particular was helpful for learning to use eclipse. Without eclipse or another suitable IDE, the project would have taken more than double the time.

Due to earlier pre-requisite programming courses only teaching C++, the Java for C++ programmers resource was extremely viable and increased my understanding of the source code.

3) Notes, Managing Code and Organisation

Once a possible implementation for the bug fix was theorized and coding began, it became apparent that some type of logging system had to be started. Ordered notes were used to organize ideas, TODOs and just keep a record of changes in order to resume coding faster after a sustained break or to possibly fall back on previous code if an unrecoverable mistake was encountered.

While changes were being made to the code, a goal based procedure was used to ensure the changes would fit with the overall system. This procedure allowed for coding in small sections, step-by-step with a broader design behind it. As soon as the programming theory strayed off course, the goal would need to be reassessed to make sure the task was being properly completed. By using a broader design, code was theorized to be as efficient and modular as possible.

4) Testing the Fixes

i) For this bug, an automated test harness was written to ensure the implemented fixes were functioning as intended. This testHarness, which is included in the provided .zip as well as in the appropriate SVN directory, tests for boundary and equivalence cases in the Passenger capturing zone and dynamically tests for Taxis' properly picking up Passengers by using user specified input.

The testHarness is controlled by running a similar thread concurrently with the SimWindow thread at the same tick rate. The tick rate is controlled by the Speed slider just as the original application to allow quicker testing, however as documented in the included README file and the runtime notes, increasing the speed may cause inaccurate results due to the multi-threaded nature of the code. All actions performed by the test are synced by counting the current tick (called by i.cTick in the test.java code). Once the current tick condition is satisfied, the test driver will call appropriate methods to either spawn entities, reset the intersection or conduct tests.

The initial boundary and equivalence cases tested are hard-coded to be run every time the application is started. The dynamic tests for Taxi Pick-Ups are conducted from user input, which is recorded and stored before the initialization of the SimWindow (GUI). The required user input is specified as prompts during program runtime. A "Passed!" result from the dynamic tests is obtained when the correct amount of passengers is remaining in the loadingArea (or passengerQueue) compared with the amount of Taxis designated and is calculated at the conclusion of the specific test.

Using the already implemented pop-up window for intersection statistics, the testHarness outputs its results into this window with the conclusion of each testing section. As well as the output window, a "safety" feature exists that prints the test results to the terminal/console (upon the signalling of the stop button) to allow for impartial test viewing.

Mostly the tests yielded expected results, however by conducting the tests a "soft" bug in the fix was raised. With the "FAILED!" result of the last two equivalence tests (as can be seen from running the testHarness), it became evident that due to the capturing process, some pedestrians will pass through the loadingArea without being stopped. These results arise because the capture area (loadingArea) is moved to the right for every new pedestrian that is stopped in the queue, this allows for a better real world simulation, however the cost is that Pedestrians moving from East to West have a chance to avoid capture if they are very near another Pedestrian. A decision was made to leave this "soft" bug unchanged because not only are the chances of this bug occurring in the original application insignificant but also this can further simulate realism by "People" accidentally walking through the taxi queue.

Another bug which was caught by the tests was a bug induced by increasing the taxi stopping time. This bug caused new Pedestrians spawned from stopped taxis to walk too far, hence leaving their designated PedestrianLane. This was due to the passengers travelling distance being attached to the current taxi stopped tick and when the maximum tick was increased the distance also increased. This was fixed by adding extra terminating conditions, but instead tied with current position of the Pedestrian not the tick rate.

ii) As the nature of this bug was an unintended probability variable setting, the fix was only tested by means of the GUI and observing the Taxis spawning correctly from the other road entries.

iii) Due to this bug having only three available inputs (yes,no,X), each input was tested manually through the GUI. As a result, all possible cases were tested and results were as developer intended. Tests were conducted on a idle CPU/Memory load only.

iv) Logically determined bug, testing unnecessary. Code functions exactly the same way except now exceptions will be properly caught.

Lessons Learned:

1) Project Development

After some time spent sorting through the source files, it became very apparent that the code was split into several substantial sections and assigned to each member of the project team. The most evident indication of this was left in the comments by authors signing them with their names and leaving questions for other team members. This structure allows for the greatest coding efficiency, however splitting the workload in such a way necessitates each developer be proficient in their required section. As each person must complete their section individually, an abundance of trust and communication between team members is necessary for this system to deliver a finished product. A project manager is needed to keep such a development system on track, he/she would need to assign tasks/sections whilst never losing perspective of the deadline, functionality and specifications.

2) Software Structure

The software is structured in a very modular manor. Each class is made into an individual .java file and located in appropriately grouped and flowing directories. Every java file contains only methods relevant to its existence and member variables/functions not needed by other classes are encapsulated by flagging them as private. Most methods do one and only one thing and are hence very modular. Modularity is very important to allow functions/variables to be reused in many different ways, thereby increasing code organisation and decreasing its complexity hence making it more understandable.

3) Software Architecture

The program was written entirely in Java (maybe v1.7?). The file format chosen for Sprite images was ".gif" with initial testing done with ".jpg" files as is evident with the file "testDummy.jpg". Model and config data used to input traffic survey information has file extension ".csv" (Comma Separated Value) and is an efficient way to store data on a permanent storage device.

It is evident that a convention for structuring of the code was decided upon because classes have the same indentation and formatting styles.

Subversion control (SVN) was used as a Cloud Storage Version Control between project members to allow for easy sharing of files, communication within comments and progress indications.

A JFrame was used as the main window for the program and the point (0,0) was set in the top left corner of the 2D geometric (X,Y) plane.

Functions were created in Utils class to allow for converting real world data, such as kmph, to programming standards, such as (X,Y) movement per tick.

There is evidence of Entity Unit Tests as one of the testing methods.

4) Maintenance

For a project to be maintained over time it requires consistent motivation.
Possible motivators include:

a) Money - The most relevant to real life software projects
b) Assignments/Grades/Education - Used to test knowledge of code
c) Passions - Most relevant in game development or open source systems, good example can be found at http://www.openttd.org/en/ and http://www.centos.org/

It is clear that after this assignment was originally completed, handed-in and graded, the maintenance of the project ceased which consequently revealed several bugs for this report in 2014. However, there is evidence of project maintenance during near release (hand-in) development stages with comments such as "TODO as pedestrians are an arraylist, do we realise that this is really inefficient to remove them like this."

5) Testing

The only evidence of testing in the project is Unit Entity Tests. These tests were used to catch thrown exceptions by triggering specific else conditions, such as setting the direction of a pedestrian to a negative degree or setting the speed of an entity to a negative number.

6) Understanding the Code

Although, I am relatively new to the Java Programming Language, the code itself had a decent flow making it easier to understand. From my knowledge in a C++ background, I quickly gathered that Java and C++ are very closely related Programming Languages. The use of well placed comments and reasonable variable names allowed a quick grasp of implementations and included objects.

Finding the TaxiRank constructor was simply a matter of searching through the source files using eclipse. Once the constructor was found, tracing the Taxi stopping logics became a clear task of understanding the tick() method used to progress time during the simulation.

With the already implemented draw(Graphics2D g) function, I was able to create temporary Java Rectangles and Ovals using the .drawRect and .drawOval functions included in the base java libraries, which helped in the process of determining appropriate (X,Y) positions on the GUI.

The (0,0) position of the 2D Geometric (X,Y) plane as well as North being 270 degrees really hindered my ability to understand entity positioning.

Without instructions detailing the ability to move around the (X,Y) plane in the GUI by dragging the mouse cursor, initial bug reporting was difficult because it was hard to tell whether the TaxiRank was even spawning upon enabling its checkbox.

Project in Hindsight:

1) Direction

In order to eliminate any possible confusion, it would have been ideal to make North direction "0/360 degrees" as it is in the real world. Interestingly, the code has North set on "270 degrees" which is very misleading and I am yet to discover the reasoning behind this.

2) Geometric 2D (X,Y) plane

The start point (0,0) of the 2D plane is in the top left corner of the GUI. This leads to some confusion because the Y co-ordinate runs oppositely to the X co-ordinate, it would have been a better decision to set up (0,0) in the bottom left corner to position everything accordingly. By doing this, the (X,Y) plane would be more familiar by comparing it to basic geometric mathematics.

3) Use of "Lazy" Booleans

In a few cases of the source code, there could be found "if" statements that had boolean functions in their conditions, these cases were especially hard to understand because there was no context to the boolean function. For example, in AddPedestrian() (found in PedestrianLane.java) there exists: if (!pedestrians.add(p)) {System.out.println("Failed to add Pedestrian");} , this line could be written differently to increase readability whilst still processing the exact same instructions.

4) Commenting

Although the commenting was very adequate in most sections of the source code, there exists areas that are very lacking and make understanding the logics almost impossible. For example, in the road.java constructor there exists over 100 lines of code with very minimal explanation, this code could easily be confused by it's own writer which means it would be especially difficult for an unfamiliar programmer to interpret.

The commenting would need to be properly regulated to ensure it stayed consistent throughout the project and allow for unfamiliar programmers to quickly understand the code.

5) Structure

The structure of the code is excellent throughout the project. It is clear a convention was kept to ensure the indentation and formatting remained consistent between team members.

6) Pedestrian Collision

If the deadline allowed for it, I would have implemented Pedestrian collision similar to vehicle entities to make the simulation more like the real world.

7) Bus Stops

As this is a traffic simulation it would only be appropriate to include the bus stops on North Terrace to induce further traffic congestion.

Conclusion:

In conclusion, the analytical component of this assignment really encouraged me to understand project development from a software engineering perspective. Large projects should be efficiently developed as a team for a scheduled deadline and include satisfactory bug testing, maintenance and function as per specifications.

The practical component of this assignment taught me to understand unfamiliar code on demand, find any bugs, fix the bugs, test the bugs and release an updated product revision.