Elevator Project Report

CSE 2010—DATA STRUCTURES & ALGORITHMS

FLORIDA INSTITUTE OF TECHNOLOGY

FALL 2015

Liandra Bennett | Borja Canseco | Luke Wiskowski
BENNETTL2014@MY.FIT.EDU | BCANSECO2014@MY.FIT.EDU | LWISKOWSKI2014@MY.FIT.EDU

MOTIVATION

This report documents the design of our new elevator control system. Though all elevators serve the same purpose, not all are efficient. The goal for our project was to design an elevator system that works faster and more efficiently than the basic elevator.

A basic elevator, which in this case is our “dummy” elevator, is designed to go all the way up and all the way down continuously, even if nobody is inside. It only stops to let passengers off or to let passengers in (and it will let any passenger in, even if they are going to the lobby and the elevator is on its way to the top floor). It is highly inefficient because it takes too long to deliver passengers to their floors and would realistically waste electricity.

After making modifications to the dummy elevator’s algorithm, we were able to create a more advanced elevator with stricter behavior. If there is no one to serve, the elevator will stop operating and wait. If the elevator is busy, it can pick up passengers moving in the same direction along its way. If it’s not busy, the elevator can change directions to better accommodate passengers. The modifications we’ve made, have drastically improved the elevator’s performance as the pool of passengers and number of floors increases.

SYSTEM OVERVIEW

Our project's source directory consists of the following files:

- AdvancedElevator.java
- BaselineElevator.java
- Elevator.java
- ElevatorGUI.java
- PassengerReleased.java
- PassengerRequest.java
- StdDraw.java
- Visual.java

These files have been packaged into an executable **Elevator.jar** file.

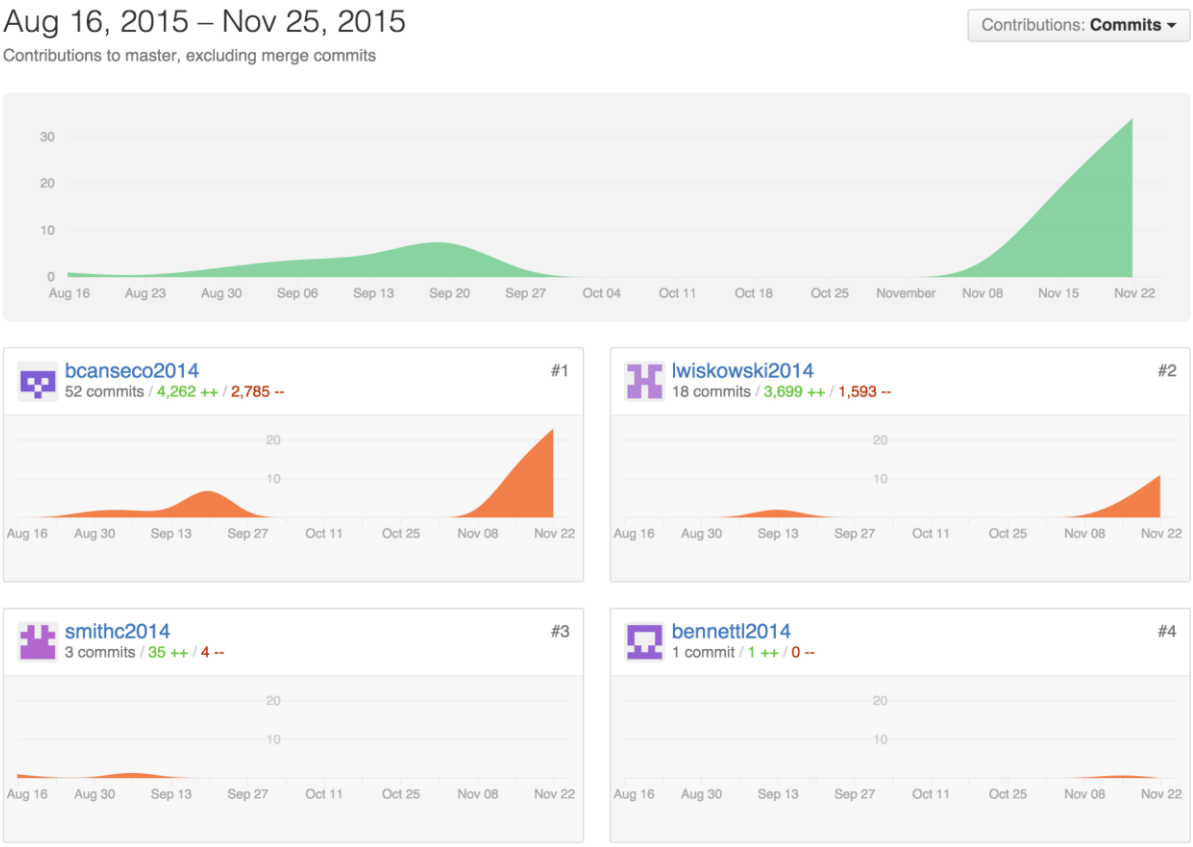
To run, launch the included Elevator.jar file. An installation of Java is required.

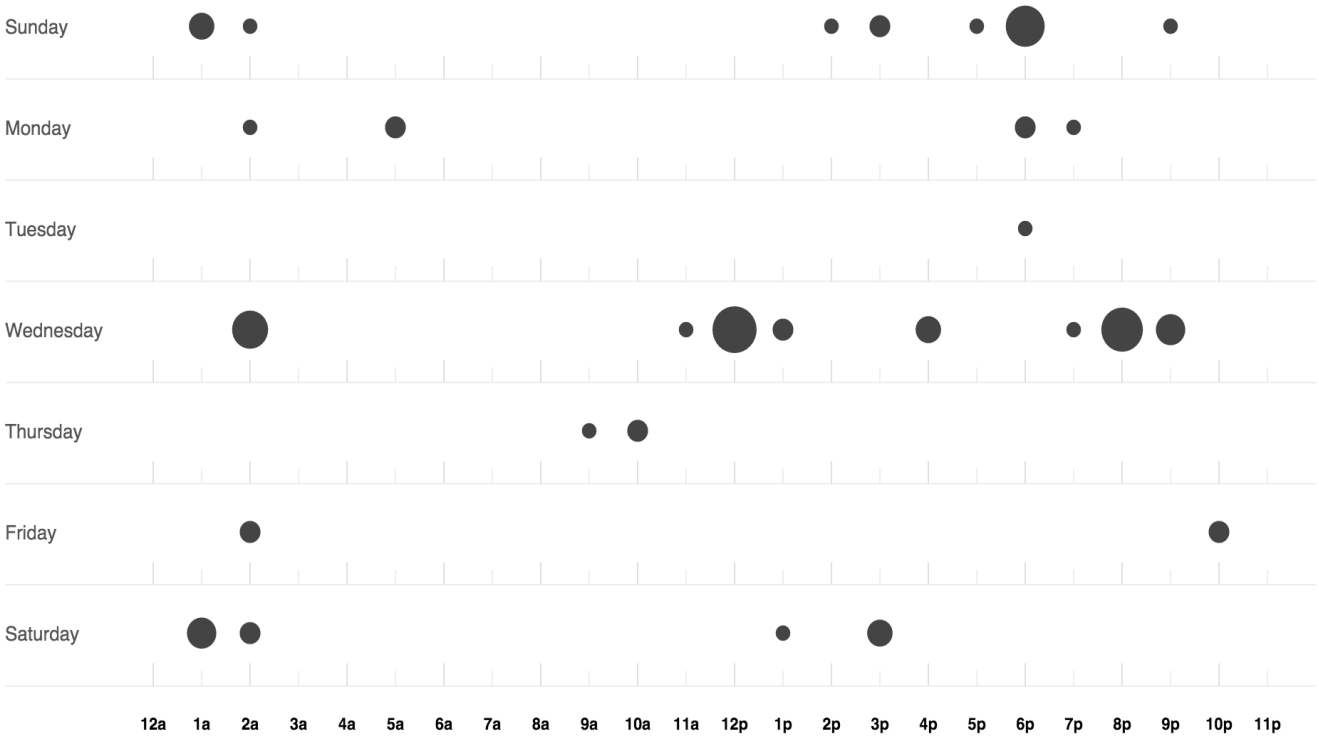
To install Java, go to: https://www.java.com/en/download/

VM/command-line arguments are not parsed. Once the system is running, a control system window will appear on your screen. Using the control system, you will be able to manipulate the amount of floors, passengers, capacity, maximum weight, maximum seconds, door delta, floor delta and time for the simulation. If you are unsure of what any of these input fields do, hover over the input field for a tooltip. Near the bottom of the control window, you will be able to select whether you want to run a simulation of either the Baseline elevator or the Advanced elevator.

PROJECT MANAGEMENT

The project was done through a Version Control System. We chose Github to host our code. We would meet in person to discuss ideas and share our code with each other, as well as work independently and commit our changes to github. The first set of code that we committed was voided after the the classes that we were supposed to extend were later submitted to students. From then on we worked independently. Code was committed after it was finished in order to merge it together. More edits were made to make them work with each other. Below is the commit history from the group members. It contains Information such as when users committed, how many times, and how many lines they wrote or removed.





This plot also generated by Github shows the times when code was committed in the last week, and how many times the changes were made. All of the commits we made were done to the master. We tested all our code before we committed in order to ensure that we wouldn't break what we had. As bugs were found we committed more code to fix them.

DESIGN

1. AdvancedElevator.java

This is our optimized version of the elevator. It implements a few improvements over the baseline in order to behave more like a typical real-world elevator. It will not let anyone in if their destination floor is the same as their origin floor (fault of the RNG). If the elevator is empty and there's nobody to serve, it will skip Time ahead to the time when the Passenger at the top of the service queue presses a button. If multiple people at a floor press the button, the first person to press it decides which floor the elevator will go towards. The elevator will pick up anyone along the way also going that direction. The elevator will not turn directions until its task is complete.

2. BaselineElevator.java

This is our "dummy" elevator. It extends elevator, and initializes it. This elevator starts at the lobby, goes all the way up and all the way down continuously until all passengers have been served. It will only open its doors on a floor if a passenger inside needs to get off or a passenger on the floor wants to get on. It will accept incoming passengers even if the direction they want to go is opposite to the direction the elevator is currently going. All of this is run in a while loop that terminates once all the people are dropped off.

3. Elevator.java

This is the class file that was given as a base for us to work with. It contains all the fields that should be used in the program, as well as the constructor for the elevator. This class is extended by other classes such as BaselineElevator and AdvanceElevator. It contains the fields capacity, timeMoveOneFloor (also referred to as floorDelta), floors, doorDelta, currentFloor. These are all given values in the more advanced implementations of elevator.

4. ElevatorGUI.java

This class is what constructs the graphical user interface and also serves as the Main method/Driver for the entire project. It was coded and designed using a built-in Eclipse tool called WindowBuilder. The WindowBuilder design area allows you to move around Swing and JFrame components in order to create custom GUIs. At the same time, you can dive into the code, change things, and have the design interface re-parse your code to see changes without having to compile. The user interface contains a series of text boxes, drop-down menus, toggles, buttons, and a menu with extra features. Text boxes include entry fields for the number of floors, capacity, door delta, floor delta, passengers, max weight, and max seconds. Mousing over the input fields will show a tooltip explaining the input filter's acceptable bounds for each setting. Running the simulation with an illegal bound will populate the console with explanation(s). The drop down menu allows you to change when the simulation starts. You can toggle between the Baseline and Advanced elevator. If at any time you want to reset the elevator and passenger settings to default, you can press the Reset button. This will also clear the console log. Hitting Run Simulation will launch either

BaselineElevator.java or AdvancedElevator.java, depending on which you have selected. It will also launch our visual demo if that is enabled. You can toggle the visual demo (among other options like log saving, verbosity, console locking) from the Menu bar.

5. PassengerReleased.java

Class that was provided for standardization. Instantiated to initialize values.

6. PassengerRequest.java

Class that was provided for standardization. Instantiated to initialize values.

7. StdDraw.java

This class contains the Standard Draw library. It contains all of the methods that are available to it, as well as a method that we added in order to be able to draw things as we needed for our visualization. This is the case with the method that we needed to print out text. Standard Draw contains a method for printing out text, however it does not allow the user to change the size of it. It simply takes the string and prints it out. However, this had to change for large numbers, because once they passed 2 digit they would be larger than the size of the elevator floor. So what we did was rewrite the method, and modify it so that depending on the size of the number in question it would change the size of the texts so

that it would fit. Several other minor fields were changed, such as the removal of the menu bar and program exiting behavior.

8. Visual.java

This is the class that controls what the user of the program sees. It makes use of the Standard Draw library in order to print out the floors to represent the building and draw the elevator moving in between them. The window size is constant, however the floors are scaled depending on the amount of them in the building. The class allows values between 1 and 101 for the amount of floors. Any range in these values is done by scaling down the size of each floor, person, and elevator. This was done by a scale method that works as follows. depending on the amount of floors it returns a double value. This is used in every method to divide every line size. Since are the lines are divided by the same number they all reduce in exactly the same manner. Once everything is scaled the program operates as follows. The floors are drawn by the drawFloors method which uses a for loop to draw the necessary floors. It draws a series of rectangles with a space in the middle for the elevator to run. People are also generated and added on each floor. The people are drawn as a series of line segments and circles, scaled as needed. The drawElevator method is what controls the movement of the elevator. It takes takes several parameters such as beginning and end floor as well as a boolean to indicate the direction, and the total floors of the building. It also take into account how many people are in the elevator by getting the info from the arraylist that contains all the passengers. This contains other methods that draw people and and erase methods that erase people or floors once the elevator has moved up.

PERFORMANCE ANALYSIS

For each case, the advanced elevator ran more efficiently than the baseline elevator. In every instance we tested, the baseline elevator resulted in a higher average wait time. As we increased the amount of floors and passengers on both of the elevators, waiting time for both elevators increased as expected. The advanced elevator however, managed to bring passengers to their requested floor much faster.

The graphs below show the mean waiting time of both elevators with varying inputs. With smaller inputs for the floors and passengers, mean wait time did converge to a close difference, sometimes in favor of the baseline. However, the advanced Elevator is objectively a more optimized and faster solution for a wider data set, as the graphs and tables below will demonstrate. Our advanced elevator system is the ideal choice in real world applications.

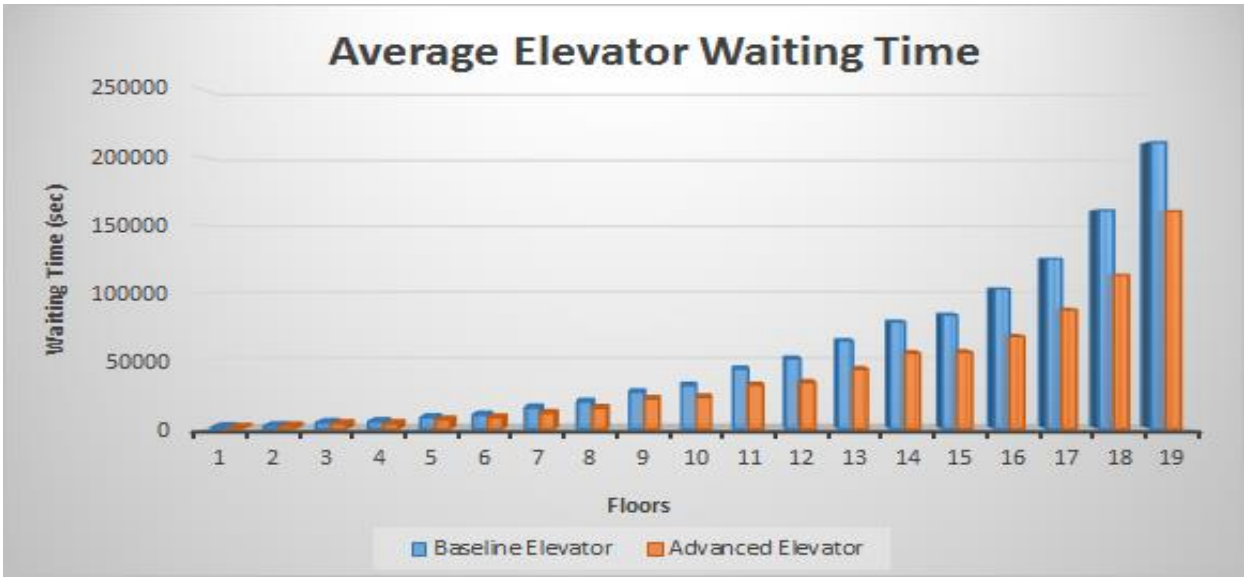
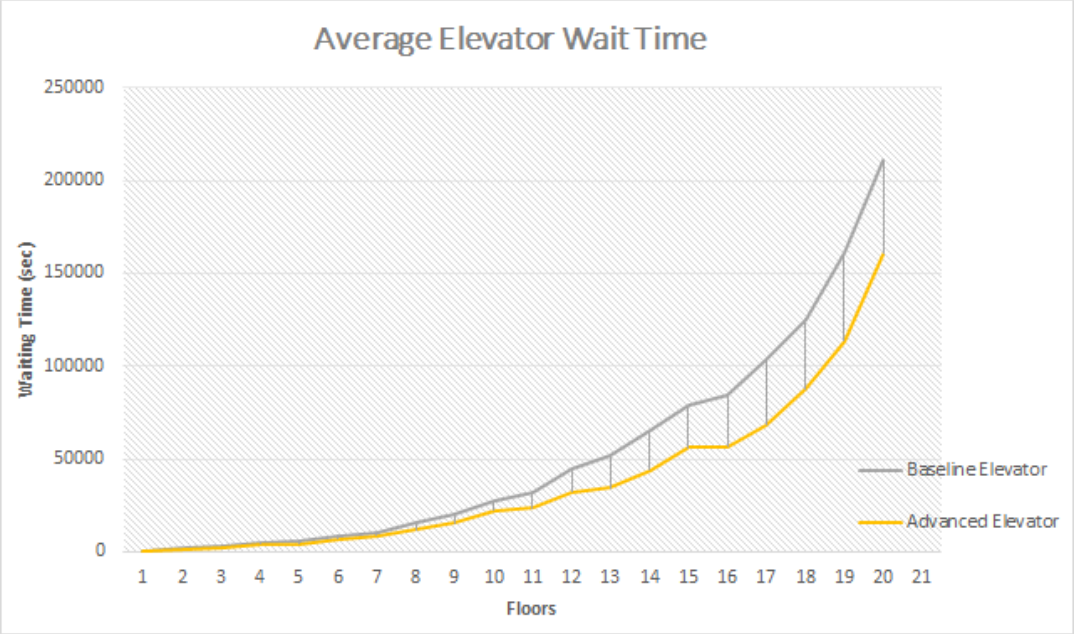


Table 1. Average Elevator Waiting Times

Floors	Passengers	Baseline Elevator Time	Advanced Elevator Time
50	6	716	600
100	12	1936	1406
150	25	2724	2393
200	36	5124	4180
250	45	5718	4165
300	60	8718	6893
350	75	10626	8602
400	95	15949	11882
450	120	20272	15591
500	150	27327	22272
550	180	32340	23487
600	210	44419	32413
650	250	51883	34299
700	280	65002	44116
750	340	78927	56011
800	330	84118	56598
850	370	103171	68235
900	465	125320	87890
950	550	160852	113316
1000	700	210869	160350

In the graph below, the region between the two lines represent the difference in time for each point. Once again you can see that at larger inputs the gap becomes significantly larger and will continue to get larger.



CONCLUSION

The data generated shows the power of algorithms. A more efficient algorithm will outperform a bad one in the long run, and these changes are most felt when numbers become larger. In a world where time is money, having the most efficient algorithm to do something is key. People who live in tall buildings or executives who work in higher floors will be willing pay more for the best algorithm if it ensures that they will get to their destination quickly. All the algorithms that we used were ones that we created ourselves, and they outperformed that standard one that was given. Thus, we believe that our algorithm is the best choice for any building that needs to service a large group of people over a large number of floors.

REFERENCES

For our code we used several libraries.

These libraries were:

- Standard Draw
- Java Utilities
- Swing
- JFrame

The sites we used for reference were

- http://introcs.cs.princeton.edu/java/stdlib/javadoc/StdDraw.html
- http://docs.oracle.com/javase/tutorial/uiswing/
- http://docs.oracle.com/javase/7/docs/api/
- http://stackoverflow.com/a/4422930