Femgineers

Date: December 14, 2017
Course: CSE 40232
Team Members:
Carolyn Bergdolt
Esme Cervantes
Patricia “Happy” Hale
Shelby Lem

Vision Statement

One snowy Christmas Eve Santa Claus was prepping for his long journey around the world delivering presents. While walking to the stables to check in on his reindeer he failed to notice a massive patch of ice. With a loud thump and crunch Santa slipped and fell. Mrs. Claus rushed him to the elf infirmary where Dr. Jingles diagnosed Santa with a broken leg! With the news spreading like wild fire, the entire north pole was on red alert. “Santa Down! Santa Down!” blared over the speakers while the head elves gathered in the Santa Support Center to discuss alternative arrangements with Mrs. Claus. After an emphatic discussion the elves and Mrs. Clause came to a decision: the elves and the reindeer would teleport around the world using the experimental wreath portals to deliver presents. Thus they were off! The sporty elves skiing, the reindeer flying, and the less coordinated elves walking to bring the children around the world their presents.

Our goal in this project was to aid the elves in their mission by creating a simulation of their strategy. In this process we intend to work as a team to practice the AGILE method of development. As team, we intended to create a functioning simulation full of holiday cheer. Through our experiences we hope to grow as a team, developers, and expand our knowledge base.

Team Members + Contributions

Carolyn Bergdolt:

Carolyn had her hands in almost every aspect of the project. Her primary contributions was the development of the roundabout logic, map construction, route generation, and road closures. Carolyn got the ball rolling and lead in the development and construction of this simulation.

Esmeralda Cervantes:

Esmeralda’s contributions were predominantly on the route generation and planning. She tackled the large task of route generation with assistance of the team. She was focused and determined to tackle this difficult task.

Patricia “Happy” Hale:

Happy contributed to the map construction, unit testing, collision avoidance, route generation, and user interface. She contributed a large portion to the thematic aspects of this simulation. She kept the levity among the team and contributed as much as she possibly could.

Shelby Lem:

Shelby’s contributions were vehicle generation, brainstorming, and map layout. Shelby kept the team’s morale high and was very open to ideas. Shelby assisted in any way that she could and proved a great team player.

Process Description

Part of the purpose of this project was to learn about using SCRUM and how to operate in an agile environment. A normal SCRUM methodology consists of daily stand-up meetings where we discuss what we had done the previous day, how it is going, and what our plan is for today. While we were not able to meet every day, we essentially addressed all of these questions whenever we met during class time or outside of class. We tried to keep each other updated on challenges we faced if we were every working separately from each other.

The use of JIRA also helped to see how a normal sprint with user stories, story points, and a backlog. We began our process by putting all of our initial stories into our JIRA backlog. We tried to relate all user stories to one of our epic stories. We also tried to estimate story points based on how many hours we thought it would take to complete that task, and stories with large story point values were split up into subtasks. Here is a brief description of how our sprints went:

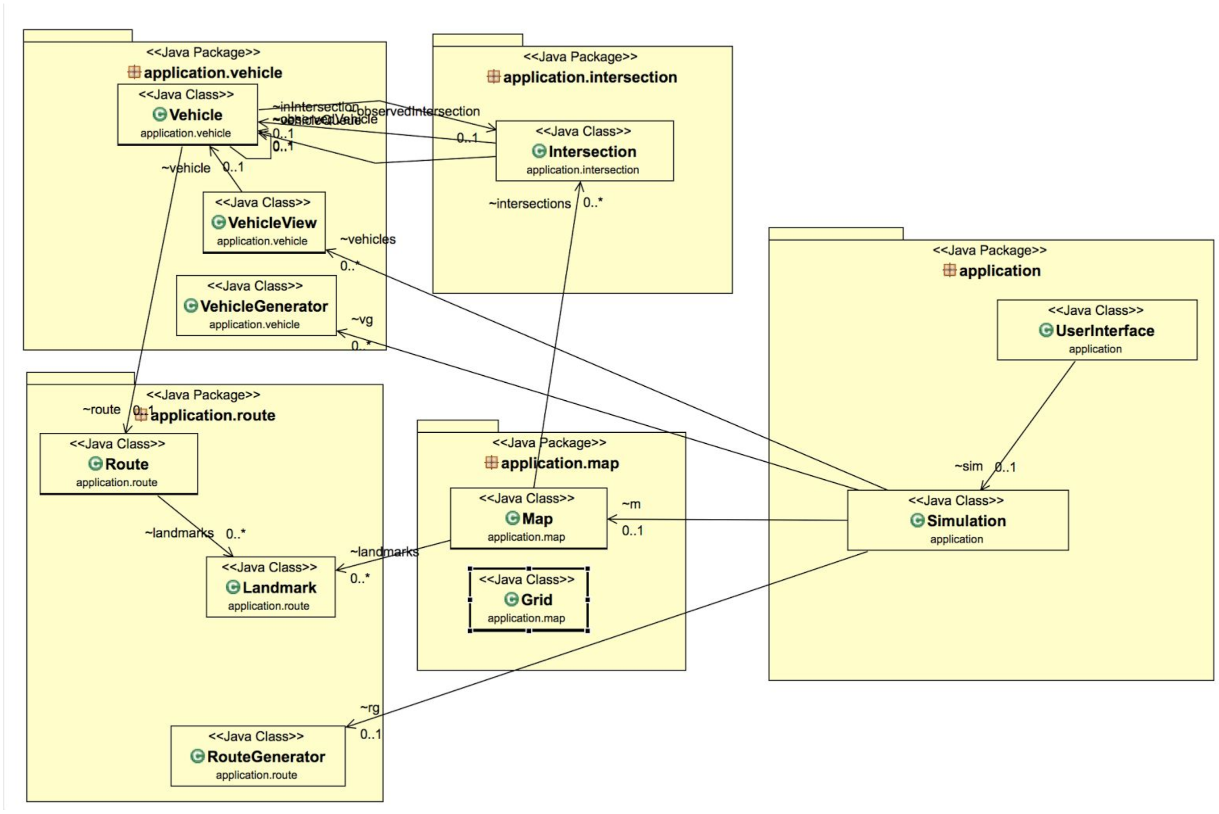
Sprint 1

Sprint 1 focused on the layout of the simulation. Our stories were centered around generating an image. Therefore our stories were meeting the general requirements laid out in the project document so that in sprint 2 most of the logic could be implemented.

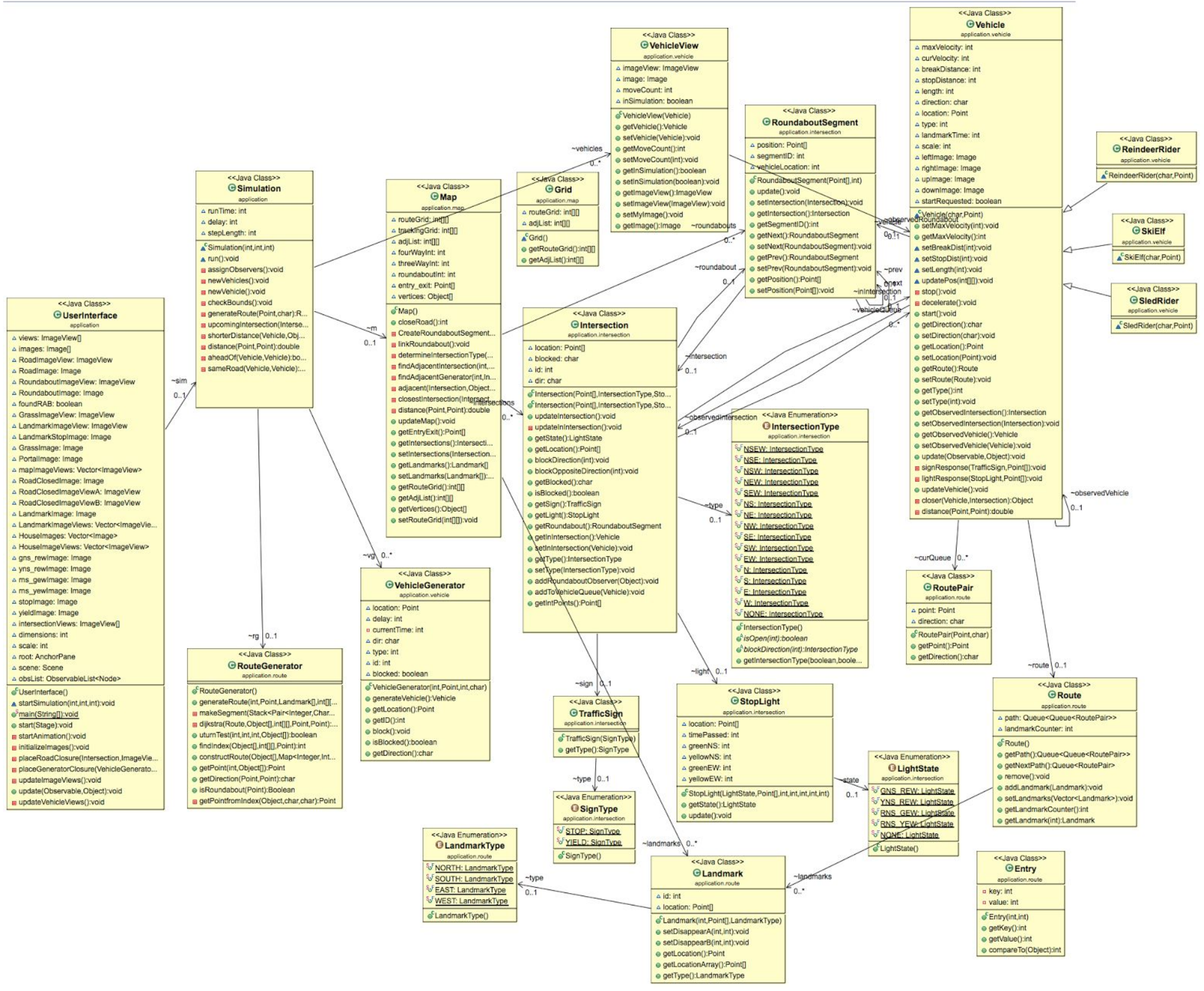
Sprint 2

Sprint 2 focused predominantly the details of the logic for each of the various methods. Our user stories were centered around our epics: Avoid Collisions, Generate Vehicles, Layout Map, Obey Traffic Rules, Plan Route, and Close Roads. Each of these epics covered different aspects of our simulation. The user stories were focused on creating the logic for these various requirements.

Architectural Diagram



UML Class Diagram



The design patterns we used were singleton (simulation), factory (vehicle generators, route generators), and observable (vehicles, intersections). The UserInterface class called a single instance of the Simulation class. The VechicleGenerators were randomly called to create vehicles, which were one of three inherited vehicle types. Each time a vehicle was created, the RouteGenerator created an instance of Route, which was an attribute of the vehicle. To manage traffic, we had the vehicles act as both observers and observables. A vehicle observer the nearest intersection in its direction of travel, as well as the nearest vehicle in its direction of travel. That way it could keep track of the traffic signal, and the speed of the vehicle in front of it so that it could initiate a decelerate method if the vehicle in front of it slowed down.

CLASS Java Docs

```java
package application.intersection;

import java.awt.Point;

/**
 * Represents a stop light and implements switches between each of its four states
 *
 * @author Femgineers
 *
 */

public class StopLight {
	LightState state;
	Point[] location = new Point[4];
	int timePassed; //since last light change
	int greenNS; // time spent in state GNS_REW
	int yellowNS; // time spent in state YNS_REW
	int greenEW; // time spent in state RNS_GEW
	int yellowEW; // time spent in state RNS_YEW

	/**
	 * Constructs a StopLight object
	 * @param initialState a LightState type that specifies the light's initial state
	 * @param loc a point array with 4 points, identical to the location of the light's
	 *     "parent" intersection
	 *            |S|W| >> |0|2|
	 *            |E|N| >> |1|3|
	 * @param gns an integer specifying the duration of the light's GNS_REW state
	 * @param yns an integer specifying the duration of the light's YNS_REW state
	 * @param gew an integer specifying the duration of the light's RNW_GEW state
	 * @param yew an integer specifying the duration of the light's RNW_YEW state
	 *            these durations are measured in the clock tick, which is incremented
	              every time the light is updated
	 *            these durations should also all be positive
	 * @param time: integer specifying the initial time on the clock, so that the light
	              doesn't have to start at the beginning of a cycle
	 *
	 */
	public StopLight(LightState initialState, Point[] loc, int gns, int yns, int gew,
	                                                                  int yew, int time) {
		state = initialState;
		location = loc;
		greenNS = gns;
		yellowNS = yns;
		greenEW = gew;
		yellowEW = yew;
		timePassed = time-1; //account for inclusive duration range, keep updates consistent
	}

	/**
	 * Gets the light's current state
	 * @return a LightState that corresponds to the light's current state
	 */
	public LightState getState() { return state; }

	/**
	 * updates the state of the light based on the internal clock tick and the static values of
		greenNS/EW / yellowNS/EW
	 * each time update is called, the clock is incremented
	 * a check is performed to see if enough clock ticks have passed to change the light state
		based on each state duration
	 */
	public void update() {
		timePassed++; //increment time

		switch(state) {
		case GNS_REW:
			if (timePassed > greenNS) {
				state = LightState.YNS_REW; // turn NS light yellow
				timePassed = 0;
			}
			break;
		case YNS_REW:
			if (timePassed > yellowNS) {
				state = LightState.RNS_GEW; // turn NS light red and EW light green
				timePassed = 0;
			}
			break;
		case RNS_GEW:
			if (timePassed > greenEW) {
				state = LightState.RNS_YEW; // turn EW light yellow
				timePassed = 0;
			}
			break;
		case RNS_YEW:
			if (timePassed > yellowEW) {
				state = LightState.GNS_REW; // turn NS light green and EW light red
				timePassed = 0;
			}
			break;
		default:
			//if the state is something other than these four options, we have an issue...
			System.out.println("update(StopLight): something has gone horribly wrong");
		}
	}
}
```

Class Unit Tests

```
package application.test;

import java.awt.Point;
import java.util.Arrays;
import org.junit.Test;
import application.intersection.StopLight;
import application.intersection.LightState;

/**
 * exhaustively test the functionality and correctness of the StopLight class
 * @author Femgineers
 *
 */
public class StopLightTest {
       /**
        * just a Point array, used as the location for the test StopLights
        */
       Point[] location = {new Point(0, 0), new Point(0, 1), new Point(1, 0), new Point(1, 1)};

       /**
        * tests whether the constructor produces a StopLight object
        */
       @Test
       public void testCreation() {
              StopLight light = new StopLight(LightState.GNS_REW, location, 10, 10, 10, 10, 0);
              assert(light instanceof StopLight);
       }

       /**
        * tests whether the state of the light is assigned properly in the constructor
        */
       @Test
       public void testStateAssignment() {
              StopLight light = new StopLight(LightState.GNS_REW, location, 10, 10, 10, 10, 0);
              assert(light.getState() == LightState.GNS_REW);
       }

       /**
        * tests whether update functionality produces the expected sequence of states
        */
       @Test
       public void testUpdate() {
              LightState[] expectedSequence = {
                            LightState.GNS_REW,
                            LightState.GNS_REW,
                            LightState.GNS_REW,
                        LightState.GNS_REW,
                        LightState.GNS_REW,
                        LightState.YNS_REW,
                        LightState.YNS_REW,
                        LightState.RNS_GEW,
                        LightState.RNS_GEW,
                        LightState.RNS_GEW,
                        LightState.RNS_GEW,
                        LightState.RNS_YEW,
                        LightState.RNS_YEW,
                        LightState.RNS_YEW
                };

                int [] dur = {4, 1, 3, 2}; //durations: [gns, yns, gew, yew]
                StopLight light = new StopLight(LightState.GNS_REW, location,
                                                    dur[0], dur[1], dur[2], dur[3], 0);

                assert(Arrays.equals(expectedSequence, updateSequence(light, dur)));
        }

        /**
         * Generates a sequence of light stages for a given light and its stage durations
         * @param light, a StopLight that will be updated through a single cycle of the given
                durations
         * @param durations, integer array containing gns, yns, gew, yew duration values (that
                should correspond with the durations of light
         * @return array of LightState values, the sequence of light states traversed given the
                light and durations
         */
        private LightState[] updateSequence(StopLight light, int[] durations) {
                int totalSum = 0;
                int sequenceCount = 0;
                for (int d: durations) totalSum += d + 1;   //sum durations, with extra + 1 to
                                    account for inclusive ranges on light state changes
                LightState[] sequence = new LightState[totalSum];

                for (int j = 0; j < durations.length; j++) {
                        for (int i = 0; i <= durations[j]; i++) {
                                light.update();
                                sequence[sequenceCount] = light.getState();
                                System.out.println(sequence[sequenceCount]);
                                sequenceCount += 1;
                        }
                }
                return sequence;
        }
}
```

Acceptance Tests

Test 1:

Given one inputted value in any of the three text boxes
And I am on the user interface page
And I hit the “Start Simulation” button
I should see an error message rendered
Then if I enter three integers in the three text boxes
And I hit the “Start Simulation” button
I should be able to see the simulation run as usual

Test 2:

Given I have hit the “Start Simulation” button
And I have entered the appropriate values in the text fields
I should see the various vehicles follow road rules
And traverse the map on the screen
And there should no collisions between reindeer