V^2 IDE Specification

1. Introduction
This document specifies the system and functional requirements for the V^2 IDE, an Integrated Development Environment specifically designed to support the Values To Variables (V2V) Curriculum in its implementation at the Girls Middle School in Mountain View (GMS). Although our initial scope of deployment and implementation of V2V and V^2 is limited, we speculate that both the curriculum and the IDE will prove valuable to a wider audience.

The purpose of the V^2 IDE is to provide a visually interactive environment in which to model, test, and explore scheduling algorithms in real-world inspired traffic scenarios. The IDE seeks to *integrate* model (testable scenarios), controller (scheduling algorithms), and view (implemented by the IDE), in order to provide students with sufficiently rapid feedback to encourage them to experiment with their implementations.

1.1 Guiding Principles
Relevant enduring objectives of both the V2V curriculum and GMS have been selected as guiding principles for the design of the V^2 IDE. The GMS goals related to V^2 are to encourage girls to:

1. *collaborate*
2. *think critically*
3. *take risks*
4. *celebrate diversity*
5. *empower them to shape a more equitable world Suggested rewording -Gabriel Adauto 3/7/10 11:45 PM*

The V2V enduring objectives related to V^2 are to teach:

1. *skills that students can use to solve problems they see in the world around them*
2. *an understanding of the social impact of engineering - that systems are created by, and have real consequences for, human beings.*

1.2 V2V Curriculum
Although V2V is predicated on a general pedagogical approach to computer science, the initial incarnation addresses a specific topic in the field of computer science: scheduling. Scheduling involves decisions by which limited resources are allocated in order to complete a collection of tasks. Although this concept is central to computer processor design, it has numerous analogs in the human world in queues of all kinds, from the grocery store to the traffic intersection. The latter was chosen as a real-world analogy to guide the majority of the V2V curriculum, due to its status as a subject of interest to middle school students (vis-à-vis driving), as well as its ubiquitous place in society.

1.2.1 V2V Goals
The V2V Curriculum defines a set of goals which guide the manifestation of curricular tasks. The following subset of these goals are those directly related to the V^2 IDE:

Goal	Description

1	Students will implement scheduling algorithms in whole group, small group, and individual settings. The algorithms implemented in group settings will be highly structured, while the individual version will be a free-form activity.
2.2	Students will create models to evaluate the performance of their implemented algorithms. A related objective is to emphasize test-driven development as good engineering practice.
4	Students will develop confidence when faced with technological complexity, in the form of a complex model (e.g. multiple intersections)
5	Students will identify the human element in real-world implementations of algorithms

1.2.2 V2V Activities

The V2V also defines a set of activities designed to meet the aforementioned goals. The following subset of these activities are those directly related to V^2 IDE:

Activity	**[Satisfied Goal] Description**
3	[2.2] Students will convert observational data of local traffic light behavior to a model configuration in the IDE.
4	[1] In a whole group, students and instructors will create an implementation of a Round Robin Scheduling algorithm for the local traffic light model in the IDE. [2.2] In a whole group, students will adjust their models for different scenarios (1 car, 1 car in each direction, 1 car in 3 of 4 directions, lines of cars in each direction, and an ambulance).
6	[2.2] In small groups, students will create models to test algorithms that have been assigned to them.
7	[1] In small groups, students will create an implementation of a priority-based algorithm for the local traffic light model in the IDE. [2.2] In small groups, students will test their algorithms against the scenarios they created in Activity 6, in the IDE.
8	[2.2] As individuals, students will create more complex models (e.g. multiple intersections)
9	[1, 4] As individuals, students will create an algorithm of their choice to address their complex models in the IDE. [GMS:diversity] Students will test their algorithms against models created by other students in the IDE.
10	[5] Students will consider the real-world implications of their algorithms, as played out in the IDE implementation (who was served, who wasn't served, and how considering these issues affected their algorithm design)

2 Approach

In support of these goals and activities, the V^2 IDE serves as an interactive nexus of model configuration, algorithmic development, and visual feedback. Aside from defining the test configuration (i.e. model), the student is faced with the task of implementing the scheduling algorithm for the traffic light. To support the creation of an intelligent algorithm, the traffic light has access to various sensors and its previous state.

This approach offers several advantages over one in which students omnipotently control all aspects of the intersection. For one, the aforementioned approach models more closely real-world behavior, and reduces the complexity of designing the scheduling algorithm. More importantly, reducing the scope to the traffic light itself lends credence to the assertion that values can (and are) encoded in perfunctory yet powerful pieces of technology!

The remaining behavior of the interactive visualization is systematically implemented by the IDE, further focusing the user's scope.

3 System Requirements

The following System Requirements Traceability Matrix is derived from the guiding principles and enduring objectives of GMS and the V2V curriculum:

ID	**Source**	**Description** (*The system will . . .)*
SR.V2V.1	V2V Enduring Objective 1	teach skills for solving relevant problems
SR.V2V.2	V2V Enduring Objective 2	convey the social impact of engineering to students
SR.GMS.1	GMS Mission Statement 1	promote collaboration among students
SR.GMS.2	GMS Mission Statement 2	promote critical thinking in students
SR.GMS.3	GMS Mission Statement 3	encourage risk taking in students
SR.GMS.4	GMS Mission Statement 4	celebrate diversity among students
SR.GMS.5	GMS Mission Statement 5	prepare students to engage in a more equitable world

4 Functional Requirements

The following Functional Requirements Traceability Matrix is derived from the system requirements and the activities of the V2V curriculum:

ID	**Source**	**Description**
FR.IDE.1	SR.V2V.1, Activities 3, 4, 7, & 9	The IDE will consist of three persistently visible and interactive work areas: model editor, control (algorithm) editor, and view (visual simulation pane)
FR.IDE.2	SR.GMS.3	The simulation pane will be equipped with a start and stop button.
FR.IDE.3	Activities 4, 7, & 9	When the start button is pressed, the model and and controller contents will be evaluated. In the absence of syntax errors, the specified simulation will be visually run, and the model and controller panes will be disabled. Otherwise, an appropriate error message will be displayed (with syntax highlighting).
FR.IDE.4	SR.GMS.3, Activity 4	When the stop button is pressed, the visual simulation will be stopped, and the model and controller panes will be enabled. The user may then alter the model or controller, and subsequently commence a new simulation without restarting the IDE.
FR.IDE.5	SR.GMS.1, SR.GMS.4, Activity 9	The IDE will facilitate uploading model and controller code from text files, while the simulation is stopped. If a model or controller is switched, the visual simulation may be (re)started without restarting the IDE.

ID	**Source**	**Description**
FR.VIS.1	SR.V2V.2, SR.GMS.2, Activities 4, 7, & 9	The visualization will be organized as an array of cells formed by parallel grid lines (dimensions TBD).

FR.VIS.2	"	The visualization will be animated as a series of turns on a refresh rate specified in the model.
FR.VIS.3	"	The visualization will support the display of up to 9 four-way intersections, arranged in a fixed 3x3 pattern. The entry, exit, and interconnections of the intersection are two-way roads. Thus, the center of each intersection is a 2x2 grid of cells.
FR.VIS.4	"	Each intersection will be managed by a traffic light, the rules of which will be defined by the contents of the Controller Pane.
FR.VIS.5	"	The visualization will show the state of the traffic light at each intersection. A traffic light may be Green, Red, or Yellow. The length of Yellow is specified in the model, as well as the length of time all sides are simultaneously red.
FR.VIS.6	"	The visualization shall support the display of lines of N cars, which commence at the entry points of the 3x3 intersection grid, and proceed to the corresponding exit points (future versions may support specifying an entry point and an orthogonal exit point, but this will require left/right-hand turn logic)
FR.VIS.7	"	Only one car may occupy a cell at a time. The size of a car will be less than 3/4 the size of an individual cell. The position of a car within a cell will be randomized for variability, except that when waiting on the edge of an intersection, the car will be situated at the fore of the cell with respect to the intersection.
FR.VIS.8	"	Cars trip the car sensor when they occupy a cell on the edge of an intersection.
FR.VIS.9	"	The visualization shall support the the animation of the cars as they follow their path from entry to exit point, as mediated by traffic lights. Cars will move forward one cell per turn if (1) the car is not at the edge of an intersection and the next cell is empty, or (2) the car is on the edge of an intersection, the light is green, and the next cell is empty.
FR.VIS.10	"	The visualization will support the display and animation of individually defined priority vehicles (ambulance, police cars). These will behave similarly to cars, except that (1) priority vehicles move forward 2 cells per turn if the next two cells are empty, (2) priority vehicles compel cars occupying any of the three cells ahead of it to pull to the side, and (3) priority vehicles trip the priority vehicle sensor when they are five or fewer cells away from an intersection.
FR.VIS.11	"	The visualization will support the display and animation of individually defined semi-trucks. These will behave similarly to cars, except that (1) semi-trucks occupy 2 cells at a time, and (2) semi-trucks move forward .5 cells per turn.

FR.VIS.12	"	The visualization will support the display and animation of sets of N pedestrians. 5 pedestrians may occupy the same cell at the same time. Pedestrians move forward .25 cells per turn.
FR.VIS.13	"	The visualization will support the display of a single set of train tracks crossing one set of parallel streets.
FR.VIS.14	"	The visualization will support the display and animation of a train, stopping traffic on one set of parallel streets. The length of the train will be specified in the model. Each car in the train moves 2 cells per turn.
FR.VIS.15	"	The visualization will support the display and animation of a rogue elephant my wife informs me that a cow is more appropriate for an Indian setting. How do you feel about this, Ashley? -Coram Bryant 3/6/10 9:29 PM that sits down in the road, blocking two-way traffic for 20 turns before deciding to move on. The elephant occupies 2 cells at the same time, and moves 1 cell per turn.
FR.VIS.16	"	The visualization will display accidents in the intersection if two cars attempt to occupy the same space. If an accident occurs, the movement of traffic in the two involved directions will cease.
FR.VIS.17	"	In the event of an accident, emergency vehicles will be dispatched to the scene. Once they arrive at the scene, the accident will be cleared in 10 turns.While fun, interesting, and true to life, I'm unclear on the value-add of this feature. Ideally, students are programming their intersections such that accidents do not occur. 90% of the time when they do occur, students will be in the process of writing code and the resulting accident will be extraneous. IMHO. -Gabriel Adauto 3/10/10 8:58 AM
FR.VIS.18	"	The visualization will highlight a vehicle of interest as specified by the model.

ID	**Source**	**Description**
FR.MOD.1	SR.GMS.2, Activities 3, 4, 6, 7, 8, & 9	The Model format will consist of Preconditions and Waves. The preconditions are the number of intersections (up to 9), the turn rate (i.e. refresh rate), Yellow light duration, and all-way red duration.
FR.MOD.2	Activities 3, 4, 6, 7, 8, & 9	A wave consists of a specification of cars, emergency vehicles and impediments. The separation between waves is specified by a number of turns. Waves may be given a repeat value.
FR.MOD.3	"	Cars may be specified individually, or as a list via a repeat value. The model-defined attributes of a car are it's start location (hence it's direction and destination), color, and whether or not it is a vehicle of interest.
FR.MOD.4	"	Emergency vehicles are specified individually. They have no model-defined attributes

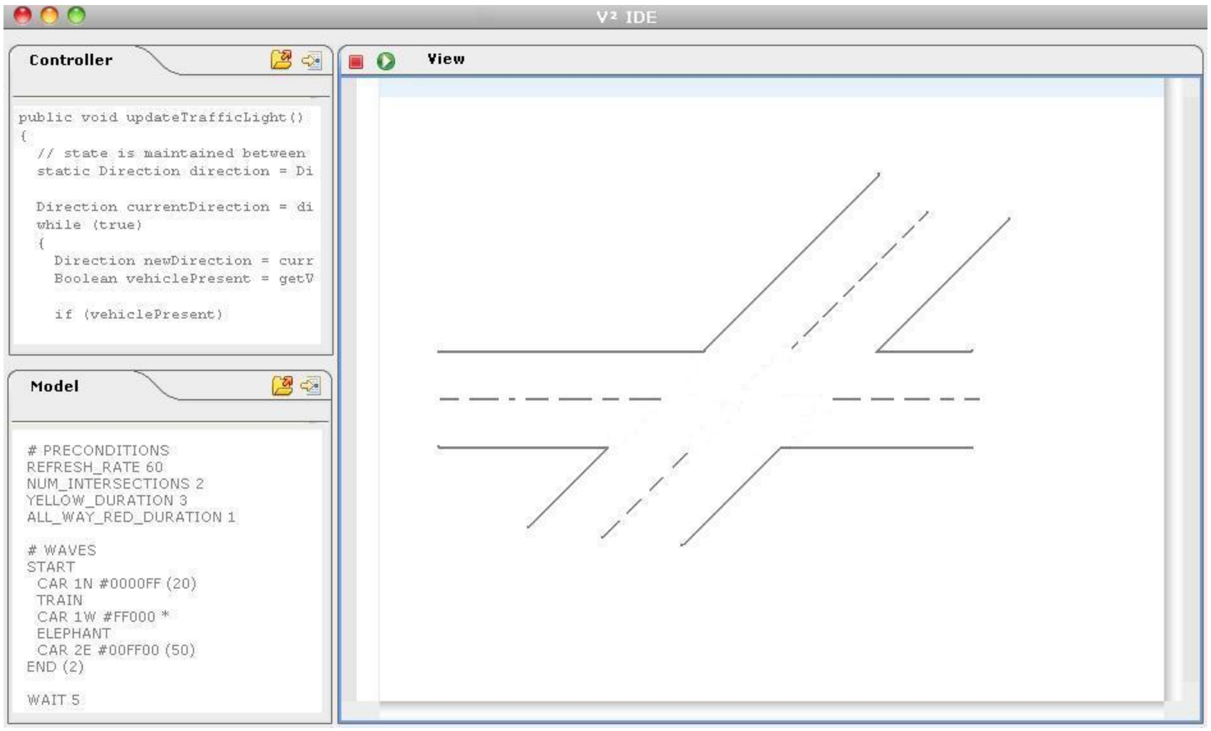
FR.MOD.5	"	Semi-trucks are specified individually. The have no model-defined attributes.
FR.MOD.6	"	Trains are specified individually. They have no model-defined attributes.
FR.MOD.7	"	Elephants are specified individually. They have no model-defined attributes.
FR.MOD.8	"	Pedestrians may be specified individually, or as a list via a repeat value. They have no model-defined attributes.

ID	**Source**	**Description**
FR.ALG.1	SR.V2V.1, Activities 4, 7, & 9	The algorithm code will be written in Java. This will provide all necssary control constructs for defining a scheduling algorithm TBD if this is the correct requirement . . . -Coram
FR.ALG.2	Activities 4, 7, & 9	The algorithm code must implement the following method, which is called once a turn per intersection How to specify which? -Coram (helper methods and global/static variables for maintaining state are encouraged): You could use a parameter or global ***info*** object that indicated the current light -Gabriel void updateTrafficLight()
FR.ALG.3	"	The algorithm code will have access to the following functions: List<[Direction,Boolean]> getEmergencySensorStatus() List<[Direction,Boolean]> getVehicleSensorStatus() List<[Direction,Boolean]> getCurrentLightStatus() List<[Direction,Boolean]> getPedestrianSensorStatus() List<[Direction,integer]> getWaitTurns() [Direction,Boolean] getRailroadSensorStatus() Boolean getVehicleSensorStatus(Direction) void setCurrentLightStatus(List<[Direction,Boolean]>) void setCurrentLightStatus(Direction, Boolean) void setGreenLights(List<Direction>) void setGreenLight(Direction)

FR.ALG.4	"	The algorithm code will make use of the Direction enum, defined as follows:
		
		enum Direction
		{
		    North,
		    East,
		    South,
		    West
		
		   Direction();
		   opposite(); // returns the opposite direction
		   clockwise(); // loops
		   ccwise(); // loops
		}

Additional requirements may be derived from periodic, endo-curricular focus groups in which students evaluate the IDE and suggest improvements. We hypothesize that this approach will further impress upon students the concept that people design technology, while simultaneously building a sense of collective ownership of the curriculum and the IDE.
How to run tests? Is there going to be some way to store models and outputs so that predictive tests can be written? -Gabriel Adauto 3/10/10 9:02 AM

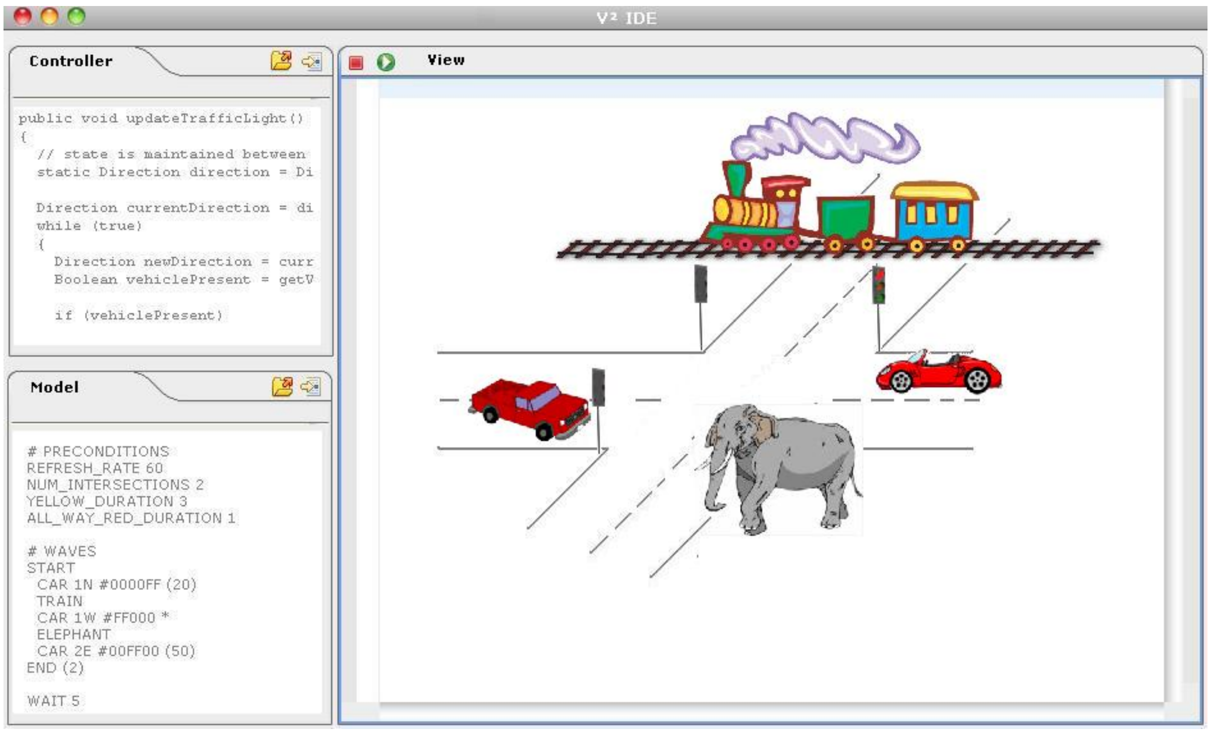
Detail: FR.IDE.*
The following diagram visually demonstrates all required features of the IDE, including the three development panes, start and stop buttons, file upload button, and Controller and Model validation buttons:



```
public void updateTrafficLight()
{
  // state is maintained between
  static Direction direction = Di

  Direction currentDirection = di
  while (true)
  {
    Direction newDirection = curr
    Boolean vehiclePresent = getV

    if (vehiclePresent)
```

```
# PRECONDITIONS
REFRESH_RATE 60
NUM_INTERSECTIONS 2
YELLOW_DURATION 3
ALL_WAY_RED_DURATION 1

# WAVES
START
 CAR 1N #0000FF (20)
 TRAIN
 CAR 1W #FF000 *
 ELEPHANT
 CAR 2E #00FF00 (50)
END (2)

WAIT 5
```

Detail: FR.VIS.*
The following diagram demonstrates a few of the features of the visualization, including an intersection, stop lights, vehicles, and impediments (train and elephant).



```
public void updateTrafficLight()
{
  // state is maintained between
  static Direction direction = Di

  Direction currentDirection = di
  while (true)
  {
    Direction newDirection = curr
    Boolean vehiclePresent = getV

    if (vehiclePresent)
```

```
# PRECONDITIONS
REFRESH_RATE 60
NUM_INTERSECTIONS 2
YELLOW_DURATION 3
ALL_WAY_RED_DURATION 1

# WAVES
START
 CAR 1N #0000FF (20)
 TRAIN
 CAR 1W #FF000 *
 ELEPHANT
 CAR 2E #00FF00 (50)
END (2)

WAIT 5
```

Detail: FR.MOD.*
The model is defined by a series newline separated entities that either describe a single object or the start/end of a group of objects.

Note: brackets indicate optional entities/fields (e.g. [(N)] indicates that (N) is optional)

Preconditions	**Description**
[REFRESH_RATE X]	X: the number of times the scene will be refreshed a second. Range: [1, 60] Default: 1
[INTERSECTIONS X]	X: the number of intersections. Intersections are referred to subsequently as [1, X]. Range: [1, 9] Default: 1
[YELLOW_DURATION X]	X: the number of turns a traffic light remains yellow after the algorithm has instructed the light to change. Default: 2
[ALL_WAY_RED_DURATION X]	X: the number of turns all ways of a traffic light stay red after the algorithm has instructed the light to change, and YELLOW_DURATION turns have expired. Default: 3

Wave Demarcations	**Description**
WAVE	Indicates the start of a wave
END [(N)]	Indicates the end of a wave
[WAIT X]	X: the number of turns to wait after the final action of the preceding stage before commencing the subsequent stage. Default: 2

	Note: entities from the prior stage need not have reached their destinations by the start of the next stage

Wave Entities	**Description**
CAR I D C [* | (N) [S]]	I: the first intersection [1, X] this car is destined to cross D: the direction {N, S, E, W} from which the car will approach the first intersection C: the color of the car as an RGB hex value (e.g. #00FF00) *: Optionally marks this car as an entity of interest (this will be reflected in the visualization) (N): Optionally indicates the number of times a car with these specifications will be repeated in the wave S: Optionally provides a step to apply to the color of each car in a repeated set Note: * and (N [S]) are mutually exclusive
SEMI I D C [* | (N) [S])]	[same as for CAR]
POLICE I D	I: the first intersection [1, X] this police car is destined to cross D: the direction from which the police car will approach the first intersection Range: {N, S, E, W}
AMBULANCE I D	[same as for POLICE]
PED I D [* | (N)]	[same as for CAR, minus the color]
TRAIN	Indicates that a train should cross the scene halfway through the wave
ELEPHANT I	Indicates that a rogue elephant should block one of the four entrance/exit points at intersection I [1, X] halfway through the wave.

Example:

```
# PRECONDITIONS
REFRESH_RATE 60
INTERSECTIONS 2
YELLOW_DURATION 3
ALL_WAY_RED_DURATION 1

# WAVES
WAVE
  CAR 1 N #0000FF (20) 10
  TRAIN
  CAR 1 W #FF000 *
  ELEPHANT 1
  CAR 2 E #00FF00 (50)
END (2)

WAIT 5

WAVE
  CAR 1 S #0000FF (200)
  POLICE 1 N
  SEMI 2 E
  PED 1 W (25)
  CAR 2 W #00FF00 (50)
END
```

Detail: FR.ALG.*
As noted, the scheduling algorithm will be implemented in Java as the function updateTrafficLight. If this function is not defined, the visualization will commence with a random distribution of green lights that will remain unchanged throughout the duration of the simulation.

The following pseudo-code examples demonstrate how to define round-robin and simple priority scheduling algorithms for a V^2 traffic light (Note that [Direction, Boolean] designates a currently undefined object containing those properties):

```java
// Round Robin

public void updateTrafficLight()
{
     // state is maintained between function calls
     static Direction direction = Direction.North;

     Direction currentDirection = direction;

     while (true)
     {
          Direction newDirection = currentDirection.clockwise();

          // Check to see if we've made a full circle without finding any waiting cars
          // if so, break from loop and maintain the original direction
          if (newDirection.equals(direction)
          {
               break;
          }

          // If we have a vehicle waiting at the next direction, switch to it
          Boolean vehiclePresent = getVehicleSensorStatus(newDirection);

          if (vehiclePresent)
          {
               setGreenLight(newDirection);
               // Exercise: what other direction might we set green here?

               // keep track of the current direction for the next function call
               direction = newDirection;
               break;
          }
     }
}
```

```java
// Simple Priority

public void updateTrafficLight()
{
     List<[Direction, Boolean]> EmergencyStatuses = getEmergencySensorStatus();
     for ([Direction, Boolean] status : EmergencyStatuses)
     {
          if (status.on())
          {
               setGreenLight(status.direction());
               return;
          }
     }

     [Direction, Boolean] rrStatus = getRailroadSensorStatus();
     if (rrStatus.on())
     {
         List<Direction> greenLights = new ArrayList<Direction>();
         greenLights.add(rrStatus.direction().clockWise());
         greenLights.add(rrStatus.direction().ccWise());
         setGreenLights(greenLights);
         return;
     }

     // further determinations of priority ...
}
```

Assessment
What values have been encoded in this IDE? explicit versus "null code" - what has been included or left out (consciously or unconsciously)

Design Decisions
1. Platform? Flash/Flex/Processing/Other?
2. compile/interpret algorithm code?