

# Software Requirements Specification

for

## FinalProjectile

Version: 1.3

Prepared by **Caitlin Wheatley**  
**Syler Wagner**

16.35 Real-Time Systems and Software

**Approved By:**

**Signature:**

## Table of Contents

<b>Table of Contents .....</b>	<b>2</b>
<b>Revision History .....</b>	<b>2</b>
<b>1. Introduction.....</b>	<b>3</b>
1.1. Purpose.....	3
1.2. Document overview .....	3
1.3. Intended Audience .....	3
1.4. Project Scope .....	3
1.5. References.....	3
<b>2. Overall Description .....</b>	<b>3</b>
2.1. Product Features .....	3
2.2. User Classes and Characteristics .....	3
2.3. Operating Environment.....	3
2.4. Design and Implementation .....	4
2.5. User Documentation .....	4
2.6. Assumptions and Dependencies .....	4
<b>3. System Features and Requirements .....</b>	<b>4</b>
3.1. GroundVehicle Requirements.....	4
3.2. Projectile Requirements .....	9
3.3. Control Requirements .....	10
3.4. Vehicle Controller Requirements .....	11
3.5. LeadingController Requirements.....	12
3.6. FollowingController Requirements .....	13
3.7. UserController Requirements .....	13
3.8. Simulator Requirements .....	14
3.9. DisplayServer Requirements .....	16
<b>4. External Interface Requirements .....</b>	<b>17</b>
4.1. User Interfaces .....	17
4.2. Hardware Interfaces .....	17
4.3. Software Interfaces .....	17
4.4. Communication Interfaces.....	17
<b>5. Other Nonfunctional Requirements .....</b>	<b>18</b>
5.1. Performance Requirements .....	18
5.2. Safety Requirements .....	18
5.3. Security Requirements .....	18
5.4. Software Quality Attributes .....	18

## Revision History

Name	Date	Reason for Changes	Version
Caitlin Wheatley	2015-05-05 00:24	Initial SRS document	1.1
Syler Wagner	2015-05-05 10:56	Updated formatting and broken references	1.2
Syler Wagner	2015-05-05 21:01	Fixed paragraph formatting and updated run() methods	1.3

# 1. Introduction

## 1.1. Purpose

- 1.1.1. This system shall fulfill the final project requirement for 16.35 by providing an interactive, networked system.

## 1.2. Document overview

- 1.2.1. This document shall outline the system requirements of Final Projectile, an interactive, multi-player game, including individual classes and interface requirements.

## 1.3. Intended Audience

- 1.3.1. This system shall be intended for the use of the Spring 2015 16.35 class and Professor Julie Shah.

## 1.4. Project Scope

- 1.4.1. This document applies to all software necessary to run the FinalProjectile game, which encompasses the classes outlined in section 3.

## 1.5. References

- 1.5.1. This SRS contains no references to websites or other documents.

# 2. Overall Description

## 2.1. Product Features

- 2.1.1. This system includes all classes necessary for the FinalProjectile game. This includes, but is not limited to, vehicle classes, controllers, and projectiles. Refer to section 3 for specific classes, features, and characteristics.

## 2.2. User Classes and Characteristics

- 2.2.1. Refer to section 3 for specific classes, features, and characteristics.

## 2.3. Operating Environment

- 2.3.1. The system shall be designed to operate on a personal computer. Players shall have the capability of logging into the game from a remote server. Refer to section for details on interfaces.

## 2.4. Design and Implementation

2.4.1. Refer to section 3 for specific classes, features, and characteristics.

## 2.5. User Documentation

2.5.1. All requirements and documentation shall be contained in this document.

## 2.6. Assumptions and Dependencies

2.6.1. The design of this system shall assume a pre-existing Display and networking capabilities. The running of this system shall be assumed to occur on a platform with the appropriate version of Java installed and running.

# 3. System Features and Requirements

## 3.1. GroundVehicle Requirements

3.1.1. The GroundVehicle shall be a thread

### 3.1.2. Variables

3.1.2.1. The GroundVehicle shall contain the internal representations of x and y position in two-dimensional space

3.1.2.1.1. x shall be in the interval  $[0, \text{Simulator.SIM\_X}]$

3.1.2.1.2. y shall be in the interval  $[0, \text{Simulator.SIM\_Y}]$

3.1.2.2. The GroundVehicle shall contain the internal representation of a heading angle  $\theta$

3.1.2.3.  $\theta$  will describe the the GroundVehicle's orientation in two-dimensional space

3.1.2.4.  $\theta$  shall be in the interval  $[-\pi, \pi]$

3.1.2.5. The GroundVehicle shall contain internal representations of the minimum and maximum forward velocity, MIN\_VEL and MAX\_VEL

3.1.2.5.1. The values of MIN\_VEL and MAX\_VEL shall be constant

3.1.2.5.2. The minimum forward velocity shall be initialized to MIN\_VEL = 1

3.1.2.5.3. The maximum forward velocity shall be initialized to MAX\_VEL = 10

3.1.2.6. The GroundVehicle shall contain internal representations of the linear velocities  $x'$  and  $y'$

3.1.2.6.1.  $\sqrt{x'^2 + y'^2}$  shall be in the interval  $[\text{MIN\_VEL}, \text{MAX\_VEL}]$

3.1.2.7. The GroundVehicle shall contain the internal representation of an angular velocity  $\theta'$

3.1.2.7.1.  $\theta'$  shall be in the interval  $[-\pi/4, \pi/4]$

3.1.2.8. The GroundVehicle shall contain a reference to the Simulator object it is associated with

3.1.2.9. The GroundVehicle shall contain an integer representing the millisecond duration between vehicle state updates, GV\_MS\_INCREMENT

3.1.2.9.1. The value of GV\_MS\_INCREMENT shall be constant

3.1.2.9.2. The millisecond increment between vehicle state updates shall be initialized to GV\_MS\_INCREMENT = 100

### 3.1.3. Constructor

#### 3.1.3.1. GroundVehicle(double pose[3], double dx, double dy, double dtheta)

##### 3.1.3.1.1. The constructor shall take four arguments

3.1.3.1.1.1. An IllegalArgumentException shall be thrown if the double pose[3] argument is not of length 3

3.1.3.1.1.2. An IllegalArgumentException shall be thrown if any of the arguments' types differ from those specified in 3.1.3.1

##### 3.1.3.1.2. The internal representation of the $x$ , $y$ , $\theta$ pose, and $x'$ , $y'$ , $\theta'$ velocities shall be initialized according to the constructor arguments

3.1.3.1.2.1. The internal representation of the position and heading shall be initialized to the values of the array  $[x, y, \theta] = \text{pose}[3]$

3.1.3.1.2.2. The internal representation of the linear velocity in the  $x$  direction shall be initialized as  $x' = dx$

3.1.3.1.2.3. The internal representation of the linear velocity in the  $y$  direction shall be initialized as  $y' = dy$

3.1.3.1.2.4. The internal representation of the angular velocity shall be initialized as  $\theta' = dtheta$

##### 3.1.3.1.3. The constructor shall initialize the variables to values within the intervals specified in 3.1.2.1.1, 3.1.2.1.2, 3.1.2.4, 3.1.2.6.1 and 3.1.2.7.1

3.1.3.1.3.1. If an element in the array pose[3] falls outside the allowable interval specified in 3.1.2.1.1, 3.1.2.1.2 and 3.1.2.4, the internal representation of that position variable shall be clamped at the nearest limit

3.1.3.1.3.1.1. If the value falls below the specified range, the internal representation of that position variable shall be initialized to the lower limit of the allowable interval

3.1.3.1.3.1.2. If the value falls above the specified range, the internal representation of that position variable shall be initialized to the upper limit of the allowable interval

3.1.3.1.3.2. If a velocity value falls outside the allowable intervals specified in 3.1.2.6.1 and 3.1.2.7.1, the internal representation of that velocity shall be clamped at the nearest limit as specified in 3.1.4.8

3.1.3.1.3.2.1. If the value falls below the specified range, the internal representation of that velocity shall be initialized to the lower limit of the allowable interval

3.1.3.1.3.2.2. If the value falls above the specified range, the internal representation of that velocity shall be initialized to the upper limit of the allowable interval

- 3.1.3.2. GroundVehicle(double pose[3], double s, double omega)
  - 3.1.3.2.1. The constructor shall take three arguments
    - 3.1.3.2.1.1. An IllegalArgumentException shall be thrown if the double pose[3] argument is not of length 3
    - 3.1.3.2.1.2. An IllegalArgumentException shall be thrown if any of the arguments' types differ from those specified in 3.1.3.1
  - 3.1.3.2.2. The internal representation of  $x$ ,  $y$ ,  $\theta$ ,  $x'$ ,  $y'$ ,  $\theta'$  shall be initialized according to the constructor arguments
    - 3.1.3.2.2.1. The internal representation of the position and heading shall be initialized to the values of the array  $[x, y, \theta] = \text{pose}[3]$
    - 3.1.3.2.2.2. The linear velocities shall be calculated based on forward speed  $s$  and heading angle  $\theta$  according to equations 3.1.3.2.2.2.1 and 3.1.3.2.2.2.2
      - 3.1.3.2.2.2.1.  $x' = s \cdot \cos(\theta)$
      - 3.1.3.2.2.2.2.  $y' = s \cdot \sin(\theta)$
    - 3.1.3.2.2.3. The internal representation of angular velocity shall be initialized to the value  $\theta' = \text{omega}$
  - 3.1.3.2.3. The constructor shall initialize the variables to values within the intervals specified in 3.1.2.1.1, 3.1.2.1.2, 3.1.2.4, 3.1.2.6.1 and 3.1.2.7.1
    - 3.1.3.2.3.1. If an element in the array pose[3] falls outside the allowable interval specified in 3.1.2.1.1, 3.1.2.1.2 and 3.1.2.4, the internal representation of that position variable shall be clamped at the nearest limit as described in 3.1.3.1.3.1
    - 3.1.3.2.3.2. If a velocity value falls outside the allowable intervals specified in 3.1.2.6.1 and 3.1.2.7.1, the internal representation of that velocity shall be clamped at the nearest limit as specified in 3.1.3.1.3.2

### 3.1.4. Methods

- 3.1.4.1. double [] getPosition()
  - 3.1.4.1.1. This method shall return an array of 3 doubles, corresponding to the  $x$ ,  $y$ ,  $\theta$  position and orientation of the vehicle
- 3.1.4.2. double [] getVelocity()
  - 3.1.4.2.1. This method shall return an array of 3 doubles, corresponding to the  $x'$ ,  $y'$ ,  $\theta'$  linear and angular velocities of the vehicle
- 3.1.4.3. setPosition(double [3])
  - 3.1.4.3.1. This method shall take one argument, an array of 3 doubles, corresponding to the  $x$ ,  $y$ ,  $\theta$  position of the vehicle.
    - 3.1.4.3.1.1. An IllegalArgumentException shall be thrown if the argument is not an array of length 3
  - 3.1.4.3.2. The setPosition method shall set the internal representation of the vehicle position according to the values contained within the argument array

- 3.1.4.3.3. If the setPosition method attempts to exceed the position constraints in 3.1.2.1.1, 3.1.2.1.2, and 3.1.2.4, the position shall be clamped as described in 3.1.3.1.3.1
- 3.1.4.4. setVelocity(double [3])
  - 3.1.4.4.1. This method shall take one argument, an array of 3 doubles corresponding to the  $x'$ ,  $y'$ ,  $\theta'$  linear and angular velocities of the vehicle
    - 3.1.4.4.1.1. An IllegalArgumentException shall be thrown if the argument is not an array of length 3
  - 3.1.4.4.2. The setVelocity method shall set the internal representation of the vehicle velocities according to the values contained within the argument array
    - 3.1.4.4.2.1. If the setVelocity method attempts to exceed the velocity constraints specified in 3.1.2.6.1 and 3.1.2.7.1, the internal representations of the resulting velocities shall be clamped as described in 3.1.3.1.3.2
- 3.1.4.5. controlVehicle(Control c)
  - 3.1.4.5.1. This method shall modify the internal representations of the  $x'$ ,  $y'$ ,  $\theta'$  velocities according to the specified forward speed and rotational velocity in the Control c argument.
    - 3.1.4.5.1.1.  $x'$  and  $y'$  shall be calculated based on forward speed  $s$  and heading angle  $\theta$  as described in 3.1.3.2.2
    - 3.1.4.5.1.2.  $\theta'$  shall be initialized to the value of rotational velocity  $\omega$
  - 3.1.4.5.2. The internal representations of the velocities that result from applying the control must obey the same limits as setVelocity in 3.1.4.4.2.1
    - 3.1.4.5.2.1. If the controlVehicle method attempts to exceed the velocity constraints in 3.1.2.6.1 and 3.1.2.7.1, the velocity shall be clamped as described in 3.1.3.1.3.2
- 3.1.4.6. updateState(int sec, int msec)
  - 3.1.4.6.1. This method shall take two arguments: the seconds (sec) and milliseconds (msec) comprising  $t$ 
    - 3.1.4.6.1.1. The arguments shall be combined according to the equation 3.1.4.6.1.1.1, where  $t[s] = \text{sec}$  and  $t[\text{ms}] = \text{msec}$ 
      - 3.1.4.6.1.1.1.  $t = t[s] + t[\text{ms}] \cdot 10^{-3}$
      - 3.1.4.6.1.1.2. The resulting  $t$  value will represent the time in seconds
  - 3.1.4.6.2. The updateState method will change the vehicle internal state by computing the appropriate kinematic and dynamic change that would occur after time  $t$ 
    - 3.1.4.6.2.1. The updateState method shall change the internal representation of the  $x$ ,  $y$ ,  $\theta$  pose, and  $x'$ ,  $y'$ ,  $\theta'$  velocities according to the dynamics calculated.
- 3.1.4.7. clampPosition()
  - 3.1.4.7.1. If the internal representations of  $x$ ,  $y$ ,  $\theta$  fall outside the allowable intervals specified in 3.1.2.1.1, 3.1.2.1.2 and 3.1.2.4, the

clampPosition method will clamp the position and heading angle to the nearest limit

3.1.4.7.1.1. If the value falls below the specified range, the internal representation of that position variable shall be initialized to the lower limit of the allowable interval

3.1.4.7.1.2. If the value falls above the specified range, the internal representation of that position variable shall be initialized to the upper limit of the allowable interval

#### 3.1.4.8. clampVelocity()

3.1.4.8.1. If the internal representations of linear velocity fall outside the allowable intervals specified in 3.1.2.6.1 and 3.1.2.7.1, the clampVelocity method will clamp the velocity to values within the allowable intervals

3.1.4.8.1.1. If the forward speed  $s = \sqrt{x'^2 + y'^2}$  falls below the specified range, the internal representation of the both the  $x'$  and  $y'$  velocities shall be clamped according to the equations

$$3.1.4.8.1.1.1. \quad x' = \text{MIN\_VEL} \cdot x' / s$$

$$3.1.4.8.1.1.2. \quad y' = \text{MIN\_VEL} \cdot y' / s$$

3.1.4.8.1.2. If the forward speed  $s = \sqrt{x'^2 + y'^2}$  falls above the specified range, the internal representation of the both the  $x'$  and  $y'$  velocities shall be clamped according to the equations

$$3.1.4.8.1.2.1. \quad x' = \text{MAX\_VEL} \cdot x' / s$$

$$3.1.4.8.1.2.2. \quad y' = \text{MAX\_VEL} \cdot y' / s$$

3.1.4.8.2. If the internal representation of angular velocity  $\theta'$  falls outside the allowable interval specified in 3.1.2.7.1, the clampVelocity method will clamp the angular velocity to a value within the allowable interval

3.1.4.8.2.1. If the value of  $\theta'$  falls below the specified range, the internal representation of  $\theta'$  shall be clamped to the lower limit of the allowable interval

3.1.4.8.2.2. If the value of  $\theta'$  falls above the specified range, the internal representation of  $\theta'$  shall be clamped to the upper limit of the allowable interval

#### 3.1.4.9. setSimulator(Simulator sim)

3.1.4.9.1. This method shall take one argument, a Simulator object

3.1.4.9.2. The setSimulator method shall set the reference to the vehicle's associated Simulator to point to the object present in the sim argument

#### 3.1.4.10. run()

3.1.4.10.1. While the time difference between the current system time and the value of Simulator.STARTUP\_TIME is less than 300 seconds, the run method shall check the current system time in nanoseconds

3.1.4.10.2. The run method shall store the time of the last GroundVehicle state update as a local variable



- 3.1.4.10.2.1. This time value shall be initialized to the system time in nanoseconds at the start of the run method
- 3.1.4.10.2.2. The time value shall be set to the current system time in nanoseconds after every call to the updateState method
- 3.1.4.10.3. If the difference between the current system time and the last vehicle state update is greater than 100 milliseconds, the run method shall update the vehicle state with the updateState method

## 3.2. Projectile Requirements

- 3.2.1. The Projectile shall be a thread

### 3.2.2. Variables

- 3.2.2.1. The Projectile shall contain the internal representations of x and y position in two-dimensional space
  - 3.2.2.1.1. x shall be in the interval  $[0, \text{Simulator.SIM\_X}]$
  - 3.2.2.1.2. y shall be in the interval  $[0, \text{Simulator.SIM\_Y}]$
  - 3.2.2.1.3. If a Projectile moves offscreen and falls outside the allowable ranges, it shall be removed from the simulation when the Simulator iterates over its list of Projectiles as specified in 3.2.3.5.
- 3.2.2.2. The Projectile shall contain the internal representation of a heading angle  $\theta$ 
  - 3.2.2.2.1.  $\theta$  will describe the the projectile's orientation in two-dimensional space
  - 3.2.2.2.2.  $\theta$  shall be in the interval  $[-\pi, \pi)$
- 3.2.2.3. The Projectile shall contain the internal representation of the projectile forward velocity PROJECTILE\_SPEED
  - 3.2.2.3.1. The value of PROJECTILE\_SPEED shall be constant
  - 3.2.2.3.2. The value of the projectile forward velocity shall be initialized to  $\text{PROJECTILE\_SPEED} = 3 \cdot \text{GroundVehicle.MAX\_VEL}$
- 3.2.2.4. The Projectile shall contain internal representations of the projectile's linear velocities  $x'$  and  $y'$ 
  - 3.2.2.4.1.  $x'$  and  $y'$  must satisfy the equation  $\sqrt{x'^2 + y'^2} = \text{PROJECTILE\_SPEED}$
- 3.2.2.5. The Projectile shall contain a reference to the Simulator object it is associated with
- 3.2.2.6. The Projectile shall contain the internal representation of the maximum allowable distance between a projectile and a vehicle that results in the GroundVehicle being shot, HIT\_DISTANCE

### 3.2.3. Constructor

- 3.2.3.1. Projectile(double[3] shooterPosition, Simulator sim)
  - 3.2.3.1.1. The constructor shall take two arguments
    - 3.2.3.1.1.1. An IllegalArgumentException shall be thrown if the double shooterPosition[3] argument is not of length 3
  - 3.2.3.1.2. The internal representation of the x, y,  $\theta$  pose, and  $x'$ ,  $y'$  velocities shall be initialized according to the constructor arguments
    - 3.2.3.1.2.1. The internal representation of the position and heading shall be initialized to the values of the array  $[x, y, \theta] = \text{shooterPosition}[3]$

3.2.3.1.2.2. The linear velocities shall be calculated based on PROJECTILE\_SPEED and heading angle  $\theta$  according to the equations 3.2.3.1.2.2.1 and 3.2.3.1.2.2.2

3.2.3.1.2.2.1.  $x' = \text{PROJECTILE\_SPEED} \cdot \cos(\theta)$

3.2.3.1.2.2.2.  $y' = \text{PROJECTILE\_SPEED} \cdot \sin(\theta)$

### 3.2.4. Methods

3.2.4.1. `double [] getPosition()`

3.2.4.1.1. This method shall return an array of 2 doubles, corresponding to the x and y position of the projectile

3.2.4.2. `shoot(int sec, int msec)`

3.2.4.2.1. This method shall take two arguments: the seconds (sec) and milliseconds (msec) comprising t

3.2.4.2.1.1. The arguments shall be combined according to the equation 3.1.4.6.1.1.1, where  $t[s] = \text{sec}$  and  $t[ms] = \text{msec}$

3.2.4.2.2. The shoot method will change the projectile internal state by computing the appropriate kinematic and dynamic change that would occur after time t

3.2.4.2.2.1. The shoot method shall change the internal representation of the x and y position according to the dynamics in equations 3.2.4.2.2.2 and 3.2.4.2.2.3

3.2.4.2.2.2.  $x = x + x' \cdot t$

3.2.4.2.2.3.  $y = y + y' \cdot t$

3.2.4.3. `run()`

3.2.4.3.1. While the time difference between the current system time and the value of Simulator.STARTUP\_TIME is less than 300 seconds, the run method shall check the current system time in nanoseconds

3.2.4.3.2. The run method shall store the time of the last projectile state update as a local variable

3.2.4.3.2.1. This time value shall be initialized to the system time in nanoseconds at the start of the run method

3.2.4.3.2.2. The time value shall be set to the current system time in nanoseconds after every call to the shoot method

3.2.4.3.3. If the difference between the current system time and the last projectile state update is greater than 100 milliseconds, the run method shall update the projectile state with the shoot method

## 3.3. Control Requirements

### 3.3.1. Variables

3.3.1.1. The Control shall contain the internal representation of forward speed s

3.3.1.1.1. s shall be in the interval  $[\text{GroundVehicle.MIN\_VEL}, \text{GroundVehicle.MAX\_VEL}]$ , where GroundVehicle.MIN\_VEL and GroundVehicle.MAX\_VEL take values specified in 3.1.2.5.2 and 3.1.2.5.3

3.3.1.2. The Control shall contain the internal representation of an angular velocity omega

3.3.1.2.1. omega shall be in the interval  $(-\pi, \pi]$

### 3.3.2. Constructor

3.3.2.1. `Control(double s, double omega)`

- 3.3.2.1.1. The constructor shall take two arguments
- 3.3.2.1.2. The internal representation of *s* and *omega* shall be initialized according to the constructor arguments
- 3.3.2.1.3. The constructor shall initialize the variables to values within the intervals specified in 3.3.1.1.1 and 3.3.1.2.1
  - 3.3.2.1.3.1. If the *s* value falls outside the allowable range specified in 3.3.1.1.1, an `IllegalArgumentException` shall be thrown
  - 3.3.2.1.3.2. If the *omega* value falls outside the allowable range specified in 3.3.1.2.1, an `IllegalArgumentException` shall be thrown

### 3.3.3. Methods

- 3.3.3.1. `double getSpeed()`
  - 3.3.3.1.1. This method shall return a double corresponding to *s*, the internal representation of forward speed
- 3.3.3.2. `double getRotVel()`
  - 3.3.3.2.1. This method shall return a double corresponding to *omega*, the internal representation of angular velocity

## 3.4. Vehicle Controller Requirements

- 3.4.1. The `VehicleController` shall be a thread

### 3.4.2. Variables

- 3.4.2.1. The `VehicleController` shall contain a reference to the `Simulator` object *\_s* with which it is associated
- 3.4.2.2. The `VehicleController` shall contain a reference to the `GroundVehicle` object *\_v* with which it is associated
- 3.4.2.3. The `VehicleController` shall contain an internal representation of the time when the last control was issued
  - 3.4.2.3.1. The internal representation of the last control time shall be comprise of an integer representing the component of time in seconds and an integer representing the component of time in milliseconds

### 3.4.3. Constructor

- 3.4.3.1. `VehicleController(Simulator s, GroundVehicle v)`
  - 3.4.3.1.1. The constructor shall take a `Simulator s` and a `GroundVehicle v` as arguments
    - 3.4.3.1.1.1. In case of invalid arguments, the constructor shall throw an `IllegalArgumentException`
  - 3.4.3.1.2. The internal representations of the `Simulator` and the `GroundVehicle` shall be initialized according to constructor arguments

### 3.4.4. Methods

- 3.4.4.1. `GroundVehicle getGroundVehicle()`
  - 3.4.4.1.1. This method shall return the `GroundVehicle _v`
- 3.4.4.2. `Simulator getSimulator()`
  - 3.4.4.2.1. This method shall return the `VehicleController`'s associated `Simulator` object *\_s*
- 3.4.4.3. `Control getControl(int sec, int msec)`
  - 3.4.4.3.1. This method shall take two arguments: the seconds (*sec*) and milliseconds (*msec*) comprising *t*

- 3.4.4.3.1.1. The arguments shall be combined according to the equation 3.1.4.6.1.1.1, where  $t[s] = \text{sec}$  and  $t[\text{ms}] = \text{msec}$
- 3.4.4.3.2. The shoot method shall return a control for the VehicleController's associated GroundVehicle if a Control should be issued at time  $t$ 
  - 3.4.4.3.2.1. If no control should be issued at time  $t$ , the getControl method shall return null
- 3.4.4.4. run()
  - 3.4.4.4.1. While the time difference between the current system time and the value of Simulator.STARTUP\_TIME is less than 300 seconds, the run method shall check the current system time in nanoseconds
  - 3.4.4.4.2. The run method shall store the time when the last Control was issued as a local variable
    - 3.4.4.4.2.1. This time value shall be initialized to the system time in nanoseconds at the start of the run method
    - 3.4.4.4.2.2. The time value shall be set to the current system time in nanoseconds after every call to the getControl method
  - 3.4.4.4.3. If the difference between the current system time and the time when the last Control was issued is greater than 100 milliseconds, the run method shall call the getControl method
  - 3.4.4.4.4. The run method shall apply the new Control to the GroundVehicle associated with the VehicleController

## 3.5. LeadingController Requirements

3.5.1. The LeadingController shall extend VehicleController

### 3.5.2. Variables

- 3.5.2.1. The LeadingController shall contain \_followerIndexList, a list of integers representing the Simulator GroundVehicle list indexes of the GroundVehicles to evade

### 3.5.3. Constructor

- 3.5.3.1. LeadingController(Simulator  $s$ , GroundVehicle  $v$ )
  - 3.5.3.1.1. The constructor shall take a Simulator and GroundVehicle as arguments
    - 3.5.3.1.1.1. In case of invalid arguments, the constructor shall throw an IllegalArgumentException
  - 3.5.3.1.2. The Simulator and GroundVehicle associated with the LeadingController shall be initialized according to constructor arguments

### 3.5.4. Methods

- 3.5.4.1. addFollower(GroundVehicle  $v$ )
  - 3.5.4.1.1. This method shall add  $v$  to the list of GroundVehicles contained in LeadingController
- 3.5.4.2. Control getControl(int  $\text{sec}$ , int  $\text{msec}$ )
  - 3.5.4.2.1. This method shall generate a Control to be applied to GroundVehicle associated with the LeadingController
  - 3.5.4.2.2. The Control shall be calculated such that the LeadingController moves away from the nearest GroundVehicle found with getClosestVehicle() as in 3.6.4.3
- 3.5.4.3. int getClosestFollower()
  - 3.5.4.3.1. This method shall return an integer representing the Simulator.\_vehicleList index of the GroundVehicle whose spatial location is nearest that of the GroundVehicle associated with the Leading Controller

#### 3.5.4.4. boolean tooCloseToWalls(double[] vehiclePosition)

- 3.5.4.4.1. This method shall return true if the distance between the bounds of the simulation specified in 3.1.2.1.1 and 3.1.2.1.2 and the vehiclePosition argument is 20

### 3.6. FollowingController Requirements

- 3.6.1. The FollowingController shall extend VehicleController.

#### 3.6.2. Variables

- 3.6.2.1. The FollowingController shall contain a reference to the target GroundVehicle object, \_prey

#### 3.6.3. Constructor

- 3.6.3.1. FollowingController(Simulator s, GroundVehicle v, GroundVehicle prey)  
 3.6.3.1.1. The constructor shall take a Simulator s, GroundVehicle v, and GroundVehicle prey as arguments  
 3.6.3.1.1.1. In case of invalid arguments, the constructor shall throw an IllegalArgumentException  
 3.6.3.1.2. The internal representation of \_s, \_v, and \_prey shall initialize according to constructor arguments

#### 3.6.4. Methods

- 3.6.4.1. Control getControl(int sec, int msec)  
 3.6.4.1.1. This method shall return a Control to be applied to FollowingController  
 3.6.4.1.2. The Control shall be such that the FollowingController's GroundVehicle moves towards the targetVehicle

### 3.7. UserController Requirements

- 3.7.1. The UserController shall extend VehicleController

#### 3.7.2. Variables

- 3.7.2.1. The UserController shall contain a reference to the DisplayServer object it is associated with

#### 3.7.3. Constructor

- 3.7.3.1. UserController(Simulator sim, GroundVehicle v, DisplayServer ds)  
 3.7.3.1.1. The constructor shall take three arguments  
 3.7.3.1.2. The UserController shall set its internal Simulator, GroundVehicle, and DisplayServer references to the objects present in the sim, v, and ds arguments

#### 3.7.4. Methods

- 3.7.4.1. GroundVehicle getUserVehicle()  
 3.7.4.1.1. This method shall return the GroundVehicle associated with the UserController  
 3.7.4.2. Control getControl(int sec, int msec)  
 3.7.4.2.1. The getControl shall get the next forward velocity and next rotational velocity values for the user vehicle from the UserController's associated DisplayServer as outlined in 3.9.2.3 and 3.9.2.4  
 3.7.4.2.2. If any of the resulting velocities exceed the velocity constraints in 3.1.2.6.1 and 3.1.2.7.1, those velocity values shall be clamped as described in 3.1.3.1.3.2

## 3.8. Simulator Requirements

3.8.1. The Simulator shall be a thread

### 3.8.2. Variables

3.8.2.1. The Simulator shall contain a reference to the UserController \_uc object with which it is associated

3.8.2.2. The Simulator shall contain a list \_vehicleList of the GroundVehicles in the simulation

3.8.2.3. The Simulator shall contain a list \_projectileList of the Projectiles in the simulation

3.8.2.4. The Simulator shall contain a list \_followerList of FollowingControllers

3.8.2.5. The Simulator shall contain a list \_leadingList of LeadingControllers

3.8.2.6. The Simulator shall contain a reference to the DisplayClient \_dc with which it is associated

3.8.2.7. The Simulator shall contain internal representations of the x and y dimensions of the simulation size, SIM\_X and SIM\_Y

3.8.2.7.1. The values of SIM\_X and SIM\_Y shall be constant

3.8.2.7.2. The values of SIM\_X and SIM\_Y shall be initialized according to the display dimensions set in the DisplayServer

3.8.2.7.2.1. The x dimension of the simulation shall be initialized to  $SIM\_X = DisplayServer.DISPLAY\_X / 5$

3.8.2.7.2.2. The y dimension of the simulation shall be initialized to  $SIM\_Y = DisplayServer.DISPLAY\_Y / 5$

3.8.2.8. The Simulator shall contain a long integer representing the value of the system time in nanoseconds when the simulation starts running, STARTUP\_TIME

### 3.8.3. Constructor

3.8.3.1. Simulator(DisplayClient dc)

3.8.3.1.1. The constructor shall take a DisplayClient dc as an argument

3.8.3.1.2. The internal representation of the DisplayClient shall be initialized according to constructor argument

3.8.3.1.3. The constructor shall initialize \_vehicleList as an empty list of GroundVehicles

3.8.3.1.4. The constructor shall initialize \_leadingList as an empty list of LeadingControllers

3.8.3.1.5. The constructor shall initialize \_followerList as an empty list of FollowingControllers

3.8.3.1.6. The constructor shall initialize \_projectileList as an empty list of Projectiles

### 3.8.4. Methods

3.8.4.1. GroundVehicle getVehicle(int i)

3.8.4.1.1. This method shall return the GroundVehicle at the index i in \_vehicleList

3.8.4.1.1.1. If the value of i is equal to or greater than the size of \_vehicleList, an IndexOutOfBoundsException shall be thrown

3.8.4.2. DisplayClient getDisplayClient()

3.8.4.2.1. This method shall return the the DisplayClient \_dc associated with the Simulator

3.8.4.3. addVehicle(GroundVehicle v)

3.8.4.3.1. This method shall add the GroundVehicle v to the internal list of GroundVehicles \_vehicleList.

3.8.4.3.1.1. In case of invalid arguments, this method shall throw an IllegalArgumentException.

3.8.4.4. addUserController(UserController uc)

- 3.8.4.4.1. This method shall initialize the UserController associated with the Simulator to uc
  - 3.8.4.4.1.1. In case of invalid arguments, this method shall throw an IllegalArgumentException.
- 3.8.4.5. generateProjectile()
  - 3.8.4.5.1. This method shall create a new Projectile object
  - 3.8.4.5.2. The generateProjectile method shall add the projectile p to the internal \_projectileList
- 3.8.4.6. checkIfShot(GroundVehicle gv)
  - 3.8.4.6.1. This method shall iterate over \_projectileList and determine if the distance between any projectile and the GroundVehicle gv is less than or equal to Projectile.HIT\_DISTANCE
    - 3.8.4.6.1.1. If the controller associated with the shot gv is a LeadingController or FollowingController, the checkIfShot method shall remove the projectile object which shot the vehicle from \_projectileList
      - 3.8.4.6.1.1.1. If the controller associated with the shot gv is a LeadingController, the checkIfShot method shall change gv's associated controller from LeadingController to a FollowingController
      - 3.8.4.6.1.1.2. If the controller associated with the shot gv is a FollowingController, the checkIfShot method shall remove gv from the \_vehicleList
    - 3.8.4.6.1.2. If the controller associated with the shot gv is a UserController, the checkIfShot method shall do nothing
- 3.8.4.7. removeOffscreenProjectiles()
  - 3.8.4.7.1. This method shall iterate over the Simulator's list of Projectiles and remove any projectiles that have moved outside the bounds of the simulation specified in 3.8.2.7.2.1 and 3.8.2.7.2.2
- 3.8.4.8. run()
  - 3.8.4.8.1. The run method shall initialize the value of STARTUP\_TIME to the current system time in nanoseconds
  - 3.8.4.8.2. The run method shall use the DisplayClient to clear the display of previous trajectories.
  - 3.8.4.8.3. The run method shall use the DisplayClient to enable the display mode showing complete trajectories, not just current positions.
  - 3.8.4.8.4. While the time difference between the current system time and the value of STARTUP\_TIME is less than 300 seconds, the run method shall check the current system time
  - 3.8.4.8.5. If the difference between the current system time and the last display update is greater than 100 milliseconds, the run method shall use the DisplayClient to update the display
    - 3.8.4.8.5.1. The run method shall check if any of the GroundVehicles have been shot as specified in 3.8.4.6 and remove shot GroundVehicles from \_vehicleList
    - 3.8.4.8.5.2. The x, y,  $\theta$  position and orientation of all GroundVehicles in \_vehicleList shall be sent to the display



- 3.8.4.8.5.3. Any offscreen projectiles shall be removed from `_projectileList` as specified in 3.8.4.7
- 3.8.4.8.5.4. The x and y positions of all Projectiles in `_projectileList` shall be sent to the display
- 3.8.4.8.6. After 300 seconds of time have passed, the run method shall use the `DisplayClient` to clear the display

## 3.9. DisplayServer Requirements

### 3.9.1. Variables

- 3.9.1.1. The DisplayServer shall contain an internal representation of the maximum absolute value of rotational velocity for the user vehicle, `MAX_OMEGA`
  - 3.9.1.1.1. The maximum absolute value of rotational velocity shall be initialized to  $MAX\_OMEGA = \pi/4$
- 3.9.1.2. The DisplayServer shall contain an internal representation of the forward velocity increment for the user vehicle, `SPEED_INCREMENT`
  - 3.9.1.2.1. The forward velocity increment shall be initialized to  $SPEED\_INCREMENT = 0.5$
- 3.9.1.3. The DisplayServer shall contain internal representations of the next forward velocity and rotational for the user vehicle, `nextSpeed` and `nextOmega`
  - 3.9.1.3.1. The forward velocity starting value shall be initialized to  $nextSpeed = 1$
  - 3.9.1.3.2. The rotational velocity starting value shall be initialized to  $nextOmega = 0$
- 3.9.1.4. The DisplayServer shall contain an integer representing the number of Projectiles which will be drawn on the display, `numProjectiles`
  - 3.9.1.4.1. The number of Projectiles shall be initialized to  $numProjectiles = 0$
- 3.9.1.5. The DisplayServer shall contain double arrays of the internal representations of x and y position for all Projectiles which will be drawn on the display
- 3.9.1.6. The DisplayServer shall contain internal representations of the x and y pixel dimensions of the display size, `DISPLAY_X` and `DISPLAY_Y`
- 3.9.1.7. The values of `DISPLAY_X` and `DISPLAY_Y` shall be constant
  - 3.9.1.7.1. The x dimension of the display shall be initialized to  $DISPLAY\_X = 800$
  - 3.9.1.7.2. The y dimension of the display shall be initialized to  $DISPLAY\_Y = 600$

### 3.9.2. Methods

- 3.9.2.1. `startGraphics()`
  - 3.9.2.1.1. The `startGraphics()` method shall set the dimensions of the display window to the x and y values specified in 3.9.1.7.1 and 3.9.1.7.2
- 3.9.2.2. `keyPressed(KeyEvent e)`
  - 3.9.2.2.1. The `keyPressed` method shall listen to keyboard input from the user
  - 3.9.2.2.2. If the right arrow key is pressed, the value of `nextOmega` shall be set to  $nextOmega = MAX\_OMEGA$
  - 3.9.2.2.3. If the left arrow key is pressed, the value of `nextOmega` shall be set to  $nextOmega = -MAX\_OMEGA$
  - 3.9.2.2.4. If the up arrow key is pressed, the value of `nextSpeed` shall be set to  $nextSpeed = nextSpeed + SPEED\_INCREMENT$



- 3.9.2.2.5. If the down arrow key is pressed, the value of nextSpeed shall be set to nextSpeed = nextSpeed - SPEED\_INCREMENT
- 3.9.2.2.6. If the space bar is pressed, a projectile shall be generated in Simulator as specified in 3.2.3.5.
- 3.9.2.3. double getUserSpeed()
  - 3.9.2.3.1. This method shall return a double corresponding to the next forward velocity of the user vehicle, nextSpeed
- 3.9.2.4. double getUserOmega()
  - 3.9.2.4.1. This method shall return a double corresponding to the next rotational velocity of the user vehicle, nextOmega
- 3.9.2.5. drawProjectiles(Graphics g)
  - 3.9.2.5.1. This method shall iterate over the lists of Projectile x and y locations and draws a circle of radius 1 at each Projectile location
- 3.9.2.6. drawCircle(Graphics g, int Xc, int Yc, int R)
  - 3.9.2.6.1. This method shall take four arguments: a Graphics object g, the x location of the circle center Xc, the y location of the circle center Yc, and the radius of the circle R
  - 3.9.2.6.2. This method shall draw a circle of radius R centered at (Xc, Yc) on the display

## 4. External Interface Requirements

### 4.1. User Interfaces

- 4.1.1. The main user interface shall be encompassed by the visual DisplayServer and the user-controlled UserController.

### 4.2. Hardware Interfaces

- 4.2.1. There shall be no hardware interfaces beyond the standard keyboard on the user's personal computer.

### 4.3. Software Interfaces

- 4.3.1. All software interfaces shall occur between the classes specified in section 3 and within the Java framework.

### 4.4. Communication Interfaces

- 4.4.1. Remote access shall be possible to allow multiple users to compete on the same game.

## 5. Other Nonfunctional Requirements

### 5.1. Performance Requirements

5.1.1. This system shall perform to the specifications laid forth in this document.

### 5.2. Safety Requirements

5.2.1. There are no safety requirements associated with this system.

### 5.3. Security Requirements

5.3.1. There are no security requirements associated with this system.

### 5.4. Software Quality Attributes

5.4.1. This system shall have methods that have been rigorously tested using a unit test matrix.