# Software Requirements Specification

for

# FinalProjectile

**Version: 2.1**

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

16.35 Real-Time Systems and Software

**Approved By:**

**Signature:**

# Table of Contents

# 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 |
| Syler Wagner | 2015-05-11 18:41 | Added content | 1.3.1 |
| Syler Wagner | 2015-05-12 09:19 | Refactoring velocity bounds | 1.3.3 |
| Caitlin Wheatley | 2015-05-14 12:03 | Added additional methods | 1.3.4 |
| Syler Wagner | 2015-05-15 16:30 | Major requirement updates | 1.3.5 |
| Syler Wagner | 2015-05-15 23:17 | Even more major requirement updates all across the board, still working on modifying for multiple user | 1.3.6 |
| Syler Wagner | 2015-05-16 00:43 | DisplayServer requirements | 1.3.7 |
| Caitlin Wheatley | 2015-05-16 00:52 | Finished remaining methods and formatting | 1.3.8 |
| Syler Wagner | 2015-05-16 02:45 | Fixed broken references | 2.0 |
| Syler Wagner | 2015-05-16 20:26 | Modified some requirements while writing tests | 2.1 |

# 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. Constants**

3.1.2.1. The GroundVehicle shall contain internal representations of the minimum and maximum forward velocity, MIN_VEL and MAX_VEL

3.1.2.1.1. The values of MIN_VEL  and MAX_VEL shall be constant

3.1.2.1.2. The minimum forward velocity shall be initialized to MIN_VEL = 1

3.1.2.1.3. The maximum forward velocity shall be initialized to MAX_VEL = 20

3.1.2.2. The GroundVehicle shall contain an internal representation of the maximum absolute value of rotational velocity for the vehicle, MAX_OMEGA

3.1.2.2.1. The values of MAX_OMEGA shall be constant

3.1.2.2.2. The maximum absolute value of rotational velocity shall be initiaized to MAX_OMEGA = $\pi/2$

3.1.2.3. The GroundVehicle shall contain an integer representing the millisecond duration between vehicle state updates, UPDATE_MS

3.1.2.3.1. The value of UPDATE_MS shall be constant

3.1.2.3.2. The millisecond increment between vehicle state updates shall be initialized to UPDATE_MS = FinalProjectile.VEHICLE_MS

**3.1.3. Variables**

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

3.1.3.1.1. x shall be in the interval [0, Simulator.SIM_X]

3.1.3.1.2. y shall be in the interval [0, Simulator.SIM_Y]

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

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

3.1.3.2.2. $\theta$ shall be in the interval $[-\pi, \pi)$

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

3.1.3.3.1. $\sqrt{x'^2+y'^2}$ shall be in the interval [MIN_VEL, MAX_VEL]

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

3.1.3.4.1. $\theta'$ shall be in the interval [–MAX_OMEGA, MAX_OMEGA]

3.1.3.5. The GroundVehicle shall contain a reference to the Simulator object it is associated with, _sim

3.1.3.6. The GroundVehicle shall contain a reference to the VehicleController object it is associated with, _vc

3.1.3.7. The Projectile shall contain an integer representing the color of the vehicle, _color

### 3.1.4. Constructors

3.1.4.1. GroundVehicle(double pose[3], double s, double omega)

3.1.4.1.1. The constructor shall take three arguments

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

3.1.4.1.2. The internal representation of x, y, $\theta$, x', y', $\theta'$ shall be initialized according to the constructor arguments

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

3.1.4.1.2.2. The linear velocities shall be calculated based on forward speed s and heading angle $\theta$ according to equations 3.1.4.1.2.2.1 and 3.1.4.1.2.2.2

3.1.4.1.2.2.1. $x' = s \cdot \cos(\theta)$

3.1.4.1.2.2.2. $y' = s \cdot \sin(\theta)$

3.1.4.1.2.3. The internal representation of angular velocity shall be initialized to the value $\theta'$ = omega

3.1.4.1.3. The constructor shall initialize the variables to values within the intervals specified in 3.1.3.1.1, 3.1.3.1.2, 3.1.3.2.2, 3.1.3.3.1 and 3.1.3.4.1

3.1.4.1.3.1. If an element in the array pose[3] falls outside the allowable interval specified in in 3.1.3.1.1, 3.1.3.1.2 and 3.1.3.2.2, the internal representation of that position variable shall be clamped at the nearest limit as described in 3.1.5.8

3.1.4.1.3.2. If a velocity value falls outside the allowable intervals specified in 3.1.3.3.1 and 3.1.3.4.1, the internal representation of that velocity shall be clamped at the nearest limit as specified in 3.1.5.9

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

3.1.4.2.1. The constructor shall take four arguments

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

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

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

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

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

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

3.1.4.2.3. The constructor shall initialize the variables to values within the intervals specified in 3.1.3.1.1, 3.1.3.1.2, 3.1.3.2.2, 3.1.3.3.1 and 3.1.3.4.1

3.1.4.2.3.1. If an element in the array pose[3] falls outside the allowable interval specified in in 3.1.3.1.1, 3.1.3.1.2 and 3.1.3.2.2, the internal representation of that position variable shall be clamped at the nearest limit

3.1.4.2.3.1.1. 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

3.1.4.2.3.1.2. 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

3.1.4.2.3.2. If a velocity value falls outside the allowable intervals specified in 3.1.3.3.1 and 3.1.3.4.1, the internal representation of that velocity shall be clamped at the nearest limit as specified in 3.1.5.9

3.1.4.2.3.2.1. 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

3.1.4.2.3.2.2. 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

### 3.1.5. Methods

3.1.5.1. double[3] getPosition()

3.1.5.1.1. This method shall return an array of 3 doubles, corresponding to the x, y, $\theta$ position and orientation of the vehicle

3.1.5.2. double[4] getDisplayData()

3.1.5.2.1. This method shall return an array of 4 doubles, corresponding to the x, y, $\theta$ position and orientation of the vehicle, and _color, the vehicle color

3.1.5.3. double[3] getVelocity()

3.1.5.3.1. This method shall return an array of 3 doubles, corresponding to the $x'$, $y'$, $\theta'$ linear and angular velocities of the vehicle

3.1.5.4. setPosition(double[3] newPos)

3.1.5.4.1. This method shall take one argument, an array of 3 doubles, corresponding to the $x$, $y$, $\theta$ position of the vehicle.

3.1.5.4.1.1. An IllegalArgumentException shall be thrown if the argument is not an array of length 3

3.1.5.4.2. The setPosition method shall set the internal representation of the vehicle position according to the values contained within the argument array

3.1.5.4.3. If the setPosition method attempts to exceed the position constraints in 3.1.3.1.1, 3.1.3.1.2, and 3.1.3.2.2, the position shall be clamped as described in 3.1.5.8

3.1.5.5. setVelocity(double[3] newVel)

3.1.5.5.1. This method shall take one argument, an array of 3 doubles corresponding to the $x'$, $y'$, $\theta'$ linear and angular velocities of the vehicle

3.1.5.5.1.1. An IllegalArgumentException shall be thrown if the argument is not an array of length 3

3.1.5.5.2. The setVelocity method shall set the internal representation of the vehicle velocities according to the values contained within the argument array

3.1.5.5.2.1. If the setVelocity method attempts to exceed the velocity constraints specified in 3.1.3.3.1 and 3.1.3.4.1, the internal representations of the resulting velocities shall be clamped as described in 3.1.5.9

3.1.5.6. controlVehicle(Control c)

3.1.5.6.1. This method shall take a Control c as an argument

3.1.5.6.2. 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

3.1.5.6.2.1. If c is null, this method shall do nothing

3.1.5.6.2.2. $x'$ and $y'$ shall be calculated based on forward speed $s$ and heading angle $\theta$ as described in 3.1.4.1.2

3.1.5.6.2.3. $\theta'$ shall be initialized to the value of rotational velocity omega

3.1.5.6.3. The internal representations of the velocities that result from applying the control must obey the same limits as setVelocity in 3.1.5.5

3.1.5.6.3.1. If the controlVehicle method attempts to exceed the velocity constraints in 3.1.3.3.1 and 3.1.3.4.1, the velocity shall be clamped as described in 3.1.5.9

3.1.5.7. updateState(int sec, int msec)

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

3.1.5.7.1.1. The arguments shall be combined according to the equation 3.1.5.7.1.1.1, where t[s] = sec and t[ms] = msec

3.1.5.7.1.1.1. $t = t[s] + t[ms] \cdot 10–3$

3.1.5.7.1.1.2. The resulting t value will represent the time in seconds

3.1.5.7.2. The updateState method will change the vehicle internal state by computing the appropriate kinematic and dynamic change that would occur after time t

3.1.5.7.2.1. The updateState method shall change the internal representation of the x, y, θ pose, and x', y', θ' velocities according to the dynamics calculated

3.1.5.8. clampPosition()

3.1.5.8.1. If the internal representations of x, y, θ fall outside the allowable intervals specified in 3.1.3.1.1, 3.1.3.1.2 and 3.1.3.2.2, the clampPosition method will clamp the position and heading angle to the nearest limit

3.1.5.8.1.1. 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

3.1.5.8.1.2. 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

3.1.5.9. clampVelocity()

3.1.5.9.1. If the internal representations of linear velocity fall outside the allowable intervals specified in 3.1.3.3.1 and 3.1.3.4.1, the clampVelocity method will clamp the velocity to values within the allowable intervals

3.1.5.9.1.1. 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

3.1.5.9.1.1.1. $x' = MIN\_VEL \cdot x' / s$

3.1.5.9.1.1.2. $y' = MIN\_VEL \cdot y' / s$

3.1.5.9.1.2. 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

3.1.5.9.1.2.1. $x' = MAX\_VEL \cdot x' / s$

3.1.5.9.1.2.2. $y' = MAX\_VEL \cdot y' / s$

3.1.5.9.2. If the internal representation of angular velocity θ' falls outside the allowable interval specified in 3.1.3.4.1, the clampVelocity method shall clamp the angular velocity to a value within the allowable interval

3.1.5.9.2.1. If the value of θ' falls below the specified range, the internal representation of θ' shall be clamped to the lower limit of the allowable interval

3.1.5.9.2.2. If the value of θ' falls above the specified range, the internal representation of θ' shall be clamped to the upper limit of the allowable interval

3.1.5.10. setSimulator(Simulator sim)

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

3.1.5.10.2. The setSimulator method shall set the reference to the vehicle's associated Simulator to _sim = sim

3.1.5.11. run()

3.1.5.11.1. The run method shall store the system time in nanoseconds when the GroundVehicle starts running as a local variable, startupTime

3.1.5.11.2. While the time difference between the current system time in nanoseconds and the value of startupTime is less than FinalProjectile.GAME_TIME seconds, the run method shall check the current system time in nanoseconds

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

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

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

3.1.5.11.4. If the difference between the current system time and the last vehicle state update is greater than UPDATE_MS 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. Constants

3.2.2.1. The Projectile shall contain the internal representation of the projectile forward velocity PROJECTILE_SPEED

3.2.2.1.1. The value of PROJECTILE_SPEED shall be constant

3.2.2.1.2. The value of the projectile forward velocity shall be initialized to PROJECTILE_SPEED = 6 · GroundVehicle.MAX_VEL

3.2.2.2. 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

3.2.2.2.1. The value of HIT_DISTANCE shall be constant

3.2.2.2.2. The millisecond increment between projectile state updates shall be initialized to HIT_DISTANCE = 4

3.2.2.3. The Projectile shall contain an integer representing the millisecond duration between projectile state updates, UPDATE_MS

3.2.2.3.1. The value of UPDATE_MS shall be constant

3.2.2.3.2. The millisecond increment between projectile state updates shall be initialized to UPDATE_MS = FinalProjectile.PROJECTILE_MS

### 3.2.3. Variables

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

3.2.3.1.1. x shall be in the interval [0, Simulator.SIM_X]

3.2.3.1.2. y shall be in the interval [0, Simulator.SIM_Y]

3.2.3.1.3. 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.

3.2.3.2. The Projectile shall contain the internal representation of a heading angle θ

3.2.3.2.1. θ will describe the the projectile's orientation in two-dimensional space

3.2.3.2.2. θ shall be in the interval $[-\pi, \pi)$

3.2.3.3. The Projectile shall contain internal representations of the projectile's linear velocities x' and y '

3.2.3.3.1. x' and y' must satisfy the equation $\sqrt{x'^2+y'^2}$ = PROJECTILE_SPEED

3.2.3.4. The Projectile shall contain a reference to the Simulator object it is associated with, _sim

3.2.3.5. The Projectile shall contain a reference to the UserController object it is associated with, _uc

3.2.3.6. The Projectile shall contain an integer representing the color of the projectile, _color

3.2.3.6.1. The integer shall represent the index of the color for the user vehicle which fired the projectile in the DisplayServer.COLORS array

3.2.3.7. The Projectile shall contain an integer ID representing the ID of the UserController object associated with the projectile

### 3.2.4. Constructor

3.2.4.1. Projectile(double[3] shooterPosition, Simulator sim, UserController uc)

3.2.4.1.1. The constructor shall take three arguments

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

3.2.4.1.1.2. If the sim argument is null, the constructor shall throw an IllegalArumentException

3.2.4.1.1.3. If the uc argument is null, the constructor shall throw an IllegalArumentException

3.2.4.1.2. The internal representation of the x, y, θ pose, and x', y' velocities shall be initialized according to the first constructor argument

3.2.4.1.2.1. The internal representation of the position and heading shall be initialized to the values of the array [x, y, θ] = shooterPosition[3]

3.2.4.1.2.2. The linear velocities shall be calculated based on PROJECTILE_SPEED and heading angle θ according to the equations 3.2.4.1.2.2.1 and 3.2.4.1.2.2.2

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

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

3.2.4.1.3. The reference to the associated Simulator object shall be initialized to _sim = sim

3.2.4.1.4. The reference to the associated VehicleController shall be initialized according to _uc = uc

3.2.4.1.5. The projectile _color shall be initialized to the color of the GroundVehicle associated with the uc UserController argument

### 3.2.5. Methods

3.2.5.1. double[3] getPosition()

3.2.5.1.1. This method shall return an array of 3 doubles, corresponding to the x, y, $\theta$ position and orientation of the projectile

3.2.5.2. double[3] getDisplayData()

3.2.5.2.1. This method shall return an array of 3 doubles, corresponding to the x, y position of the projectile, and _color, the projectile's color representation

3.2.5.3. shoot(int sec, int msec)

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

3.2.5.3.1.1. The arguments shall be combined according to the equation 3.1.5.7.1.1.1, where t[s] = sec and t[ms] = msec

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

3.2.5.3.2.1. The shoot method shall change the internal representation of the x and y position according to the dynamics in equations 3.2.5.3.2.2 and 3.2.5.3.2.3

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

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

3.2.5.4. run()

3.2.5.4.1. While the time difference between the current system time and the value of _sim.STARTUP_TIME is less than FinalProjectile.GAME_TIME seconds, the run method shall check the current system time in nanoseconds

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

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

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

3.2.5.4.3. If the difference between the current system time and the last projectile state update is greater than UPDATE_MS 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 [GroundVehicle.MIN_VEL, GroundVehicle.MAX_VEL], where GroundVehicle.MIN_VEL and GroundVehicle.MAX_VEL take values specified in 3.1.2.1.2 and 3.1.2.1.3

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 (−π, π]

### 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 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. VehicleController Requirements

3.4.1. The VehicleController shall be a thread

### 3.4.2. Constants

3.4.2.1. The VehicleController shall contain an integer representing the millisecond duration between successive control updates, UPDATE_MS

3.4.2.1.1. The value of UPDATE_MS shall be constant

3.4.2.1.2. The millisecond increment between successive control updates shall be initialized to UPDATE_MS = FinalProjectile.CONTROLLER_MS

### 3.4.3. Variables

3.4.3.1. The VehicleController shall contain a reference to the Simulator object _s with which it is associated

3.4.3.2. The VehicleController shall contain a reference to the GroundVehicle object _v with which it is associated

### 3.4.4. Constructor

3.4.4.1. VehicleController(Simulator s, GroundVehicle v)

3.4.4.1.1. The constructor shall take a Simulator s and a GroundVehicle v as arguments

3.4.4.1.1.1. If the sim argument is null, the constructor shall throw an IllegalArumentException

3.4.4.1.1.2. If the v argument is null, the constructor shall throw an IllegalArumentException

3.4.4.1.2. The reference to the associated Simulator object shall be initialized to _sim = sim

3.4.4.1.3. The reference to the associated GroundVehicle object shall be initialized to _v = v

### 3.4.5. Methods

3.4.5.1. setGroundVehicle(GroundVehicle v)

3.4.5.1.1. This method shall take a GroundVehicle object v as an argument

3.4.5.1.2. This method shall set the reference to the controller's associated GroundVehicle object to v

3.4.5.2. GroundVehicle getGroundVehicle()

3.4.5.2.1. This method shall return the GroundVehicle _v associated with the controller

3.4.5.3. removeGroundVehicle()

3.4.5.3.1. This method shall set the reference to the associated GroundVehicle object _v to null

3.4.5.4. Simulator getSimulator()

3.4.5.4.1. This method shall return the VehicleController's associated Simulator object _s

3.4.5.5. double normalizeAngle(double theta)

3.4.5.5.1. This method shall take a double representing an angle as an argument

3.4.5.5.1.1. While theta is less than –π, an increment of 2π shall be added to theta

3.4.5.5.1.2. While theta is greater than π, an increment of 2π shall be subtracted from theta

3.4.5.5.2. This method shall return the angle theta normalized to the interval [-π, π]

3.4.5.6. Control clampControl(double s, double omega)

3.4.5.6.1. This method shall take two doubles, speed s and rotational velocity omega, as arguments

3.4.5.6.2. If the speed s is greater than GroundVehicle.MAX_VEL, the value of s shall be set to s = GroundVehicle.MAX_VEL

3.4.5.6.3. If the speed s is less than GroundVehicle.MIN_VEL, the value of s shall be set to s = GroundVehicle.MIN_VEL

3.4.5.6.4. If rotational velocity is greater than GroundVehicle.MAX_OMEGA, the value of omega shall be set to omega = GroundVehicle.MAX_OMEGA

3.4.5.6.5. If rotational velocity is less than −GroundVehicle.MAX_OMEGA, the value of omega shall be set to omega = –GroundVehicle.MAX_OMEGA

3.4.5.6.6. This method shall return a Control with speed s and rotational velocity omega

3.4.5.7. Control getControl(int sec, int msec)

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

3.4.5.7.1.1. The arguments shall be combined according to the equation 3.1.5.7.1.1.1, where t[s] = sec and t[ms] = msec

3.4.5.7.2. The getControl method shall return a control for the VehicleController's associated GroundVehicle if a Control should be issued at time t

3.4.5.7.2.1. If no control should be issued at time t, the getControl method shall return null

3.4.5.8. run()

3.4.5.8.1. The run method shall store the system time in nanoseconds when the VehicleController starts running as a local variable, startupTime

3.4.5.8.2. While the time difference between the current system time in nanoseconds and the value of startupTime is less than FinalProjectile.GAME_TIME seconds, the run method shall check the current system time in nanoseconds

3.4.5.8.3. The run method shall store the time when the last Control was issued as a local variable

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

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

3.4.5.8.4. If the difference between the current system time and the time when the last Control was issued is greater than UPDATE_MS milliseconds, the run method shall call the getControl method

3.4.5.8.5. 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. Constants

3.5.2.1. The LeadingController shall contain internal representations of the minimum and maximum forward velocity for the controller's associated GroundVehicle, LEADING_MIN_VEL and LEADING_MAX_VEL

3.5.2.1.1. The values of LEADING_MIN_VEL  and MAX_MAX_VEL shall be constant

3.5.2.1.2. The minimum forward velocity shall be initialized to LEADING_MIN_VEL = 5 · GroundVehicle.MIN_VEL

3.5.2.1.3. The maximum forward velocity shall be initialized to LEADING_MAX_VEL = GroundVehicle.MAX_VEL

3.5.2.2. The LeadingController shall contain the variable DANGER_ZONE specifying the threshold distance at which a vehicle is too close to the simulation boundar

3.5.2.2.1. The theshold distance shall be initialized to DANGER_ZONE = 20

### 3.5.3. Constructor

3.5.3.1. LeadingController(Simulator s, GroundVehicle v)

3.5.3.1.1. The constructor shall take a Simulator s and a GroundVehicle v as arguments

3.5.3.1.1.1. If the sim argument is null, the constructor shall throw an IllegalArumentException

3.5.3.1.1.2. If the v argument is null, the constructor shall throw an IllegalArumentException

3.5.3.1.2. The reference to the associated Simulator object shall be initialized to _sim = sim

3.5.3.1.3. The reference to the associated GroundVehicle object shall be initialized to _v = v

3.5.3.1.4. The constructor shall set the _color of the associated GroundVehicle object to Simulator.LEADING_COLOR

### 3.5.4. Methods

3.5.4.1. Control getControl(int sec, int msec)

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

3.5.4.1.1.1. The arguments shall be combined according to the equation 3.1.5.7.1.1.1, where t[s] = sec and t[ms] = msec

3.5.4.1.2. The getControl method shall return a control for the LeadingController's associated GroundVehicle if a Control should be issued at time t

3.5.4.1.3. The Control shall be calculated such that the LeadingController moves away from the nearest GroundVehicle found with getClosestVehicle() as in 3.6.4.3

3.5.4.2. GroundVehicle getClosestFollower()

3.5.4.2.1. This method shall store the shortest distance between the leading vehicle _v and and the closest other vehicle in the simulation as a local variable shortestDistance

3.5.4.2.2. This method shall store the reference to the closest vehicle in a local variable, closestVehicle

3.5.4.2.2.1. The getClosestFollower() method shall iterate over the Simulator's _vehicleList array of GroundVehicles excluding itself and compute their linear distance as specified in 3.8.6.5

3.5.4.2.2.2. If the distance to another vehicle calculated is less than the previous shortestDistance, the shortestDistance shall be set to that distance and closestVehicle shall be set to that vehicle

3.5.4.2.3. The getClosestFollower method shall return closestVehicle, the GroundVehicle whose location is nearest that of the GroundVehicle associated with the LeadingController

3.5.4.3. boolean tooCloseToWalls(double[3] vehiclePosition)

3.5.4.3.1. This method shall take a double array of length 3 as an argument, corresponding to a vehicle's position and orientation [x, y, θ] = vehiclePosition

3.5.4.3.2. This method shall evaluate if the distance between the vehicle and the bounds of the simulation is less than DANGER_ZONE

3.5.4.3.2.1. If the distance along x between the bounds of the simulation specified in 3.1.3.1.1 and the vehicle's x position is less than DANGER_ZONE, this method shall return true

3.5.4.3.2.2. If the distance along y between the bounds of the simulation specified in 3.1.3.1.2 and the vehicle's y position is less than DANGER_ZONE, this method shall return true

3.5.4.3.2.3. This method shall return false otherwise

## 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, a GroundVehicle v, and a GroundVehicle prey as arguments

3.6.3.1.1.1. If the sim argument is null, the constructor shall throw an IllegalArumentException

3.6.3.1.1.2. If the v argument is null, the constructor shall throw an IllegalArumentException

3.6.3.1.1.3. If the prey argument is null, the constructor shall throw an IllegalArumentException

3.6.3.1.2. The reference to the associated Simulator object shall be initialized to _sim = sim

3.6.3.1.3. The reference to the associated GroundVehicle object shall be initialized to _v = v

3.6.3.1.4. The reference to the associated target GroundVehicle object shall be initialized to _prey = prey

3.6.3.1.5. The constructor shall set the _color of the associated GroundVehicle object to Simulator.FOLLOWING_COLOR

**3.6.4. Methods**

3.6.4.1. Control getControl(int sec, int msec)

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

3.6.4.1.1.1. The arguments shall be combined according to the equation 3.1.5.7.1.1.1, where t[s] = sec and t[ms] = msec

3.6.4.1.2. The getControl method shall return a control for the FollowingController's associated GroundVehicle if a Control should be issued at time t

3.6.4.1.3. The Control shall be such that the FollowingController’s GroundVehicle moves towards the target vehicle object _prey

## 3.7. UserController Requirements

3.7.1. The UserController shall extend VehicleController

**3.7.2. Constants**

3.7.2.1. The UserController shall contain an integer representing the millisecond duration between successive projectiles being fired, REACTION_TIME

3.7.2.1.1. The value of REACTION_TIME shall be constant

3.7.2.1.2. The millisecond increment between successive projectiles being fired shall be initialized to REACTION_TIME = 500

**3.7.3. Variables**

3.7.3.1. The UserController shall contain a reference to the DisplayServer object it is associated with, _ds

3.7.3.2. The UserController shall contain an integer representation of the number of projectiles fired by the user, _shots

3.7.3.3. The UserController shall contain an integer representation of the number of leading and following vehicles shot by the user, _hits

3.7.3.4. The UserController shall contain an integer representation of the number of following vehicles shot by the user, _kills

3.7.3.5. The UserController shall contain an integer representation of the total number of following vehicles shot by all users, TOTAL_KILLS

3.7.3.6. The UserController shall contain an integer representation of the total number of UserControllers created, userControllerCount

3.7.3.7. The UserController shall contain a unique integer _userID

### 3.7.4. Constructor

3.7.4.1. UserController(Simulator sim, GroundVehicle v, DisplayServer ds)

3.7.4.1.1. The constructor shall take three arguments, a Simulator object sim, a GroundVehicle object v, and a DisplayServer ds

3.7.4.1.1.1. If the sim argument is null, the constructor shall throw an IllegalArumentException

3.7.4.1.1.2. If the v argument is null, the constructor shall throw an IllegalArumentException

3.7.4.1.1.3. If the ds argument is null, the constructor shall throw an IllegalArumentException

3.7.4.1.2. The reference to the associated Simulator object shall be initialized to _sim = sim

3.7.4.1.3. The reference to the associated GroundVehicle object shall be initialized to _v = v

3.7.4.1.4. The reference to the associated DisplayServer object shall be initialized to _ds = ds

3.7.4.1.5. The value of _userID shall be initialized to _userID = userControllerCount

3.7.4.1.6. The constructer shall increment the value of userControllerCount by 1

3.7.4.1.7. The constructor shall set the _color of the associated GroundVehicle object to _v._color = userControllerCount

### 3.7.5. Methods

3.7.5.1. GroundVehicle getUserVehicle()

3.7.5.1.1. This method shall return the GroundVehicle _v associated with the UserController

3.7.5.2. Control getControl(int sec, int msec)

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

3.7.5.2.1.1. The arguments shall be combined according to the equation 3.1.5.7.1.1.1, where t[s] = sec and t[ms] = msec

3.7.5.2.2. The getControl method shall get the next forward velocity and next rotational velocity values for the user vehicle from the UserController's associated DisplayServer as outlined in 3.10.3.10 and 3.10.3.11

3.7.5.2.2.1. The getControl method shall create the next Control to be issued with the next forward velocity and next rotational velocity as arguments

3.7.5.2.3. The getControl method shall use the getProjectileGenerated() method with this UserController's _userID as an argument to determine whether the user is currently shooting

3.7.5.2.3.1. If the user is shooting, the getControl method shall generate a projectile associated with this UserController in Simulator as specified in 3.8.6.7

3.7.5.2.4. The getControl method shall return the next control for the UserController's associated GroundVehicle, specified in 3.7.5.2.2.1

## 3.8. Simulator Requirements

3.8.1. The Simulator shall be a thread

**3.8.2. Constants**

3.8.2.1. The Simulator shall contain internal representations of the x and y dimensions of the simulation size, SIM_X and SIM_Y

3.8.2.1.1. The values of SIM_X and SIM_Y shall be constant

3.8.2.1.2. The values of SIM_X and SIM_Y shall be initialized according to the display dimensions set in the DisplayServer

3.8.2.1.2.1. The x dimension of the simulation shall be initialized to SIM_X = DisplayServer.DISPLAY_X / 5

3.8.2.1.2.2. The y dimension of the simulation shall be initialized to SIM_Y = DisplayServer.DISPLAY_Y / 5

3.8.2.2. The Simulator shall contain an integer representing the millisecond duration between sending successive updates to the display, UPDATE_MS

3.8.2.2.1. The value of UPDATE_MS shall be constant

3.8.2.3. The millisecond increment between successive display updates shall be initialized to UPDATE_MS = FinalProjectile.SIMULATOR_MS

3.8.3. The Simulator shall contain integers representing the the indexes in the DisplayServer.COLORS array for different vehicle and projectile types, USER1_COLOR, USER2_COLOR, LEADING_COLOR, and FOLLOWING_COLOR

3.8.3.1. The values of USER1_COLOR, USER2_COLOR, LEADING_COLOR, and FOLLOWING_COLOR shall be constant

3.8.3.2. The integer color representation for the first user-controlled vehicle and its associated projectiles shall be initialized to USER1_COLOR = 1

3.8.3.3. The integer color representation for the second user-controlled vehicle and its associated projectiles shall be initialized to USER2_COLOR = 2

3.8.3.4. The integer color representation for vehicles associated with LeadingControllers shall be initialized to LEADING_COLOR = 3

3.8.3.5. The integer color representation for vehicles associated with FollowingControllers shall be initialized to FOLLOWING_COLOR = 4

**3.8.4. Variables**

3.8.4.1. The Simulator shall contain a long integer representing the value of the system time in nanoseconds when the simulation starts running, STARTUP_TIME

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

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

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

3.8.4.5. The Simulator shall contain references to two UserController objects with which it is associated

3.8.4.6. The Simulator shall contain _lastProjectileTime, an array of long integers of length 2 specifying the time when last projectile was fired for each usercontroller

**3.8.5. Constructor**

3.8.5.1. Simulator(DisplayClient dc)

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

3.8.5.1.2. The internal representation of the DisplayClient shall be initialized according to constructor argument as _dc = dc

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

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

**3.8.6. Methods**

3.8.6.1. GroundVehicle getVehicle(int i)

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

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

3.8.6.2. DisplayClient getDisplayClient()

3.8.6.2.1. This method shall return the the DisplayClient _dc associated with the Simulator

3.8.6.3. addVehicle(GroundVehicle v)

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

3.8.6.4. addUserController(UserController uc)

3.8.6.4.1. This method shall add a UserController object reference to theSimulator

3.8.6.4.2. If the Simulator has no associated UserControllers, addUserController shall initialize the first UserController associated with the Simulator to _uc1 = uc

3.8.6.4.3. If the Simulator has one associated UserController, addUserController shall initialize the second UserController associated with the Simulator to _uc2 = uc

3.8.6.4.4. If the Simulator already has two associated UserControllers, this method shall throw an IllegalStateException

3.8.6.5. double distance(double[3] obj1pos, double[3] obj2pos)

3.8.6.5.1. This method shall take two double arrays of length 3 as arguments, corresponding to the x, y, θ position of the two objects to be compared

3.8.6.5.1.1. An IllegalArgumentException shall be thrown if either of the two arguments is not an array of length 3

3.8.6.5.2. The distance method shall return the linear distance d between the points ($x_1$, $y_1$) and ($x_2$, $y_2$) where $x_1$ = obj1pos[0], $y_1$ = obj1pos[1], $x_2$ = obj2pos[0], $y_2$ = obj1pos[1]

3.8.6.5.2.1. The distance d shall be calculated according to the equation in 3.8.6.5.2.2

3.8.6.5.2.2. $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

3.8.6.6. boolean checkWithinDistance(double[3] obj1pos, double[3] obj2pos, double thresholdDistance)

3.8.6.6.1. This method shall take three arguments: two double arrays of length 3, corresponding to the x, y, θ position of the two objects to be compared, and a double representation of the threshold distance between the objects

3.8.6.6.1.1. An IllegalArgumentException shall be thrown if obj1pos is not an array of length 3

3.8.6.6.1.2. An IllegalArgumentException shall be thrown if obj2pos is not an array of length 3

3.8.6.6.1.3. An IllegalArgumentException shall be thrown if the thresholdDistance argument is less than or equal to zero

3.8.6.6.2. The checkWithinDistance method shall calculate the linear distance d between the objects as specified in 3.8.6.5

3.8.6.6.3. If the linear distance between the objects is less than the threshold distance, checkWithinDistance shall return true

3.8.6.6.4. If the linear distance is equal to or greater than the threshold distance, checkWithinDistance shall return false

3.8.6.7. generateProjectile()

3.8.6.7.1. This method shall calculate the time since the last projectile was fired, timeSinceLastProjectile

3.8.6.7.2. If the time since the last projectile was fired is greater than UserController.REACTION_TIME this method shall create a new Projectile object p

3.8.6.7.2.1. The generateProjectile() method shall add the projectile p to the internal _projectileList

3.8.6.7.2.2. This method shall start the projectile thread

3.8.6.7.2.3. This methods shall increase the counter for shots fired by the user associated with the projectile

3.8.6.7.2.4. This method shall reset the last time a projectile was fired, in the array _lastProjectileTime at the corresponding user's index, to the current system time in nanoseconds

3.8.6.8. boolean projectileOffscreen(double[3] projectilePos)

3.8.6.8.1. This method shall take an array of length 3, corresponding to the x, y, θ position of a projectile, as an argument

3.8.6.8.1.1. An IllegalArgumentException shall be thrown if the argument is not an array of length 3

3.8.6.8.2. If the projectile position is outside the bounds of the simulation specified in 3.8.2.1.2.1 and 3.8.2.1.2.2, projectileIsOffscreen shall return true

3.8.6.8.3. If the projectile position is within the bounds of the simulation, projectileOffscreen shall return false

3.8.6.9. removeOffscreenProjectiles()

3.8.6.9.1. This method shall iterate over the Simulator's list of projectiles and check if the projectiles have gone offscreen by calling the projectileIsOffscreen method on each projectile

3.8.6.9.2. The removeOffscreenProjectiles method shall remove any projectiles that have moved outside the bounds of the simulation from _projectileList

3.8.6.10. boolean projectileShotVehicle(double[] projectilePos, double[] vPos)

3.8.6.10.1. This method shall return true if the linear distance between the projecile position projectilePos and the vehicle position vPos, calculated as specified in 3.8.6.5 is less than HIT_DISTANCE

3.8.6.11. switchVehicleControllers(VehicleController oldController, GroundVehicle targetUserVehicle)

3.8.6.11.1. This method shall take a VehicleController oldController and a GroundVehicle targetUserVehicle as arguments

3.8.6.11.1.1. If the oldController does not have an associated GroundVehicle, an IllegalArgumentException shall be thrown

3.8.6.11.2. The switchVehicleControllers method shall get the oldController's associated GroundVehicle object v

3.8.6.11.3. This method shall change the oldController's GroundVehicle reference to null using the removeGroundVehicle() method as specified in 3.4.5.3

3.8.6.11.4. This method shall create newController, a new FollowingController object associated with this Simulator, the GroundVehicle v, and the GroundVehicle targetUserVehicle

3.8.6.11.5. This method shall start the newController thread

3.8.6.12. changeShotVehicles()

3.8.6.12.1. This method shall iterate over both _vehicleList and _projectileList and use the projectileShotVehicle() method to compare the positions of all possible pairs of a vehicle v and a projectile p

3.8.6.12.2. If the vehicle was shot by the projectile, this method shall check the vehicle color to determine the type of vehicle

3.8.6.12.3. If the vehicle v is a following vehicle, v shall be removed from _vehicleList

3.8.6.12.3.1. This method shall increment the user hit counter for the UserController associated with the projectile p

3.8.6.12.3.2. This method shall increment both the user kill counter for the UserController associated with the projectile p, and the value of UserController.TOTAL_KILLS

3.8.6.12.4. If the vehicle v is a leading vehicle, the vehicle's controller shall be changed to a FollowingController as specified in 3.8.6.11

3.8.6.12.4.1. This method shall increment the user hit counter for the UserController associated with the projectile p

3.8.6.13. run()

3.8.6.13.1. The run method shall initialize the value of STARTUP_TIME to the current system time in nanoseconds

3.8.6.13.2. The run method shall use the DisplayClient to clear the display of previous trajectories.

3.8.6.13.3. The run method shall use the DisplayClient to enable the display mode showing complete trajectories, not just current positions.

3.8.6.13.4. While the time difference between the current system time and the value of STARTUP_TIME is less than FinalProjectile.GAME_TIME seconds, the run method shall check the current system time

3.8.6.13.5. If the difference between the current system time and the last display update is greater than UPDATE_MS milliseconds, the run method shall use the DisplayClient to update the display

3.8.6.13.5.1. The run method shall check if any of the GroundVehicles have been shot as specified in 3.8.6.7 and remove shot GroundVehicles from _vehicleList

3.8.6.13.5.2. The x, y, θ position and orientation of all GroundVehicles in _vehicleList shall be sent to the display

3.8.6.13.5.3. The number of shots, hits, and kills for each user shall be sent to the display

3.8.6.13.5.4. Any offscreen projectiles shall be removed from _projectileList as specified in 3.8.6.11

3.8.6.13.5.5. The x and y positions of all Projectiles in _projectileList shall be sent to the display

3.8.6.13.6. After FinalProjectile.GAME_TIME seconds of time have passed, the run method shall send the final number of shots, hits, and kills for each user to the display and signal to the DisplayServer that the game is over

3.8.6.13.7. The run() method shall exit the entire game application after sleeping for FinalProjectile.GAME_OVER_TIMEOUT seconds

## 3.9. FinalProjectile Requirements

### 3.9.1. Constants

3.9.1.1. FinalProjectile shall contain an integer representation of the game run time in seconds, GAME_TIME

3.9.1.1.1. The value of GAME_TIME shall be constant

3.9.1.1.2. The value of the game duration in seconds shall be initialized to GAME_TIME = 200

3.9.1.2. Final Projectile shall contain an integer representation of the number of seconds the display remains after the game is over, GAME_OVER_TIMEOUT

3.9.1.2.1. The value of GAME_OVER_TIMEOUT shall be constant

3.9.1.2.2. The value of the time the display remains after the game is over shall be initialized to GAME_OVER_TIMEOUT = 10

3.9.1.3. Final Projectile shall contain integer representations of the millisecond duration between successive updates for for the Simulator, GroundVehicle, Controller, and Projectile threads

3.9.1.3.1. The millisecond durations between thread updates shall be constant for each thread

3.9.1.3.2. The time between successive Simulator thread updates shall be initialized to SIMULATOR_MS = 50

3.9.1.3.3. The time between successive GroundVehicle thread updates shall be initialized to VEHICLE_MS = 50

3.9.1.3.4. The time between successive VehicleController thread updates shall be initialized to CONTROLLER_MS = 100

3.9.1.3.5. The time between successive Projectile thread updates PROJECTILE_MS = 20

### 3.9.2.

### 3.9.3. Variables

3.9.3.1. FinalProjectile shall contain a boolean representing whether multi-player mode is enabled, MULTIPLAYER

3.9.3.2. FinalProjectile shall contain an integer representation of the number of non-user vehicles, NUM_VEHICLES

### 3.9.4. Methods

3.9.4.1. main(String[] args)

3.9.4.1.1. The main method shall parse the command line arguments

3.9.4.1.1.1. The first command line argument shall initialize the value of MULTIPLAYER

3.9.4.1.1.1.1. If the first argument is 1, the value shall be initalized to MULTIPLAYER = false

3.9.4.1.1.1.2. If the first argument is 2, the value shall be initialized to MULTIPLAYER = true

3.9.4.1.1.2. The second command argument shall specify the number of non-user vehicles to be created

3.9.4.1.1.2.1. If the value of the argument is less than 1, the game application shall exit

3.9.4.1.1.2.2. If no second command line argument is present, the value shall be set to NUM_VEHICLES = 10

3.9.4.1.2. The main method shall create a DisplayServer, DisplayClient, and Simulator associated with the DisplayClient

3.9.4.1.3. The main method shall create two UserControllers and associated vehicles if multiplayer mode is enabled, and one UserController and associated vehicle otherwise

3.9.4.1.4. The main method shall create NUM_VEHICLES LeadingControllers and associated GroundVehicles as specified by the command arguments

3.9.4.1.5. The main method shall start all the Simulator, VehicleController, and GroundVehicle threads

## 3.10. DisplayServer Requirements

### 3.10.1. Constants

3.10.1.1. The DisplayServer shall contain internal representations of the x and y pixel dimensions of the display size, DISPLAY_X and DISPLAY _Y

3.10.1.1.1. The values of DISPLAY_X and DISPLAY_Y shall be constant

3.10.1.1.2. The x dimension of the display shall be initialized to DISPLAY_X = 1250

3.10.1.1.3. The y dimension of the display shall be initialized to DISPLAY_Y = 800

3.10.1.2. The DisplayServer shall contain an internal representation of the forward velocity increment for the user vehicle, SPEED_INCREMENT

3.10.1.2.1. The value of SPEED_INCREMENT shall be constant

3.10.1.2.2. The forward velocity increment shall be initalized to SPEED_INCREMENT = 1

3.10.1.3. The DisplayServer shall contain an array of color pairs for specifying the colors of different projectile and vehicle types, COLORS

3.10.1.3.1. The COLORS array shall contain color pairs consisting of a color for each vehicle type and a darker shade of the same color

3.10.1.3.2. The COLORS array shall contain the color pairs USER1_COLOR, USER2_COLOR, LEADING_COLOR, and FOLLOWING_COLOR

### 3.10.2. Variables

3.10.2.1. The DisplayServer shall contain internal representations of the next forward velocity and rotational for two user vehicle, userSpeed[2] and userOmega[2]

3.10.2.1.1. The foward velocity starting values for both users shall be initialized to 5 · GroundVehicle.MIN_VEL

3.10.2.1.2. The rotational velocity starting values for both users shall be initialized to 0

3.10.2.2. The DisplayServer shall contain a boolean array of length two with an internal representation of whether or not a projectile is being fired by either of the two users, projectileGenerated[2]

3.10.2.2.1. The values in the array for both users shall be initalized to false

3.10.2.3. The DisplayServer shall contain an integer representing the number of projectiles which will be drawn on the display, numProjectiles

3.10.2.3.1. The number of projectiles shall be initialized to numProjectiles = 0

3.10.2.4. The DisplayServer shall contain two double arrays pX[] and pY[] with internal representations of x and y position for all projectiles which will be drawn on the display

3.10.2.5. The DisplayServer shall contain a double array pC[] with color indexes in the COLORS array for all projectiles which will be drawn on the display

3.10.2.6. The DisplayServer shall contain a double array gvC[] with color indexes in the COLORS array for all vehicles which will be drawn on the display

3.10.2.7. The DisplayServer shall contain a boolean OVER representing whether the game is over

3.10.2.8. The DisplayServer shall contain a boolean HELP representing whether the help menu is displayed

**3.10.3.Methods**

3.10.3.1. startGraphics()

3.10.3.1.1. The startGraphics() method shall set the dimensions of the display window to the x and y values specified in 3.10.1.1.2 and 3.10.1.1.3

3.10.3.2. increaseSpeed(int UserID)

3.10.3.2.1. This method shall set the value of user forward velocity for the user vehicle specified by UserID to userSpeed = userSpeed + SPEED_INCREMENT

3.10.3.2.2. If the resulting value of userSpeed exceeds the limits specified in 3.1.2.1.3, the value of userSpeed shall be set to userSpeed = GroundVehicle.MAX_VEL

3.10.3.3. decreaseSpeed(int UserID)

3.10.3.3.1. This method shall set the value of user forward velocity for the user vehicle specified by UserID to userSpeed = userSpeed – SPEED_INCREMENT

3.10.3.3.2. If the resulting value of userSpeed exceeds the limits specified in3.1.2.1.2, the value of userSpeed shall be set to userSpeed = GroundVehicle.MIN_VEL

3.10.3.4. turnLeft(int UserID)

3.10.3.4.1. This method shall set the value of user rotational velocity for the user vehicle specified by UserID to userOmega = – GroundVehicle.MAX_OMEGA

3.10.3.5. turnRight(int UserID)

3.10.3.5.1. This method shall set the value of user rotational velocity for the user vehicle specified by UserID to userOmega = GroundVehicle.MAX_OMEGA

3.10.3.6. stopTurning(int UserID)

3.10.3.6.1. This method shall set the value of user rotational velocity for the user vehicle specified by UserID to userOmega = 0

3.10.3.7. toggleProjectile(boolean generated, int UserID)

3.10.3.7.1. This method shall take two arguments, an integer specifying the user ID and a boolean representing whether or not the user vehicle specified by UserID is currently shooting

3.10.3.7.2. This method shall set the value of whether or not a projectile is being fired by that user in the projectileGenerated[] array to the value of the generated argument

3.10.3.8. keyPressed(KeyEvent e)

3.10.3.8.1. The keyPressed method shall listen to keyboard input from the user

3.10.3.8.2. If the H key is pressed, the value of HELP shall be switched between true and false

3.10.3.8.3. The first user shall use the arrow keys and space bar key as inputs

3.10.3.8.3.1. If the right arrow key is pressed, the keyPressed method shall call the turnRight method for user 1

3.10.3.8.3.2. If the left arrow key is pressed, the keyPressed method shall call the turnLeft method for user 1

3.10.3.8.3.3. If the up arrow key is pressed, the keyPressed method shall increase the user forward velocity with the increaseSpeed method for user 1

3.10.3.8.3.4. If the down arrow key is pressed, the keyPressed method shall decrease the user forward velocity with the decreaseSpeed method for user 1

3.10.3.8.3.5. If the space bar key is pressed, the keyPressed method shall use the toggleProjectile() method for user 1 and set the value to true

3.10.3.8.4. The second user shall use the A, S, D, W keys and the shift key as inputs

3.10.3.8.4.1. If the D key is pressed, the keyPressed method shall call the turnRight method for user 2

3.10.3.8.4.2. If the A key is pressed, the keyPressed method shall call the turnLeft method for user 2

3.10.3.8.4.3. If the W key is pressed, the keyPressed method shall increase the user forward velocity with the increaseSpeed method for user 2

3.10.3.8.4.4. If the S key is pressed, the keyPressed method shall decrease the user forward velocity with the decreaseSpeed method for user 2

3.10.3.8.4.5. If the shift key is pressed, the keyPressed method shall use the toggleProjectile() method for user 2 and set the value to true

3.10.3.9. keyReleased(KeyEvent e)

3.10.3.9.1. The keyReleased method shall listen to keyboard input from the user

3.10.3.9.2. The first user shall use the left and right arrow keys and space bar key as inputs

3.10.3.9.2.1. If the right arrow key is released after being pressed, the keyPressed method shall set the user rotational velocity to zero by calling the stopTurning method for user 1

3.10.3.9.2.2. If the left arrow key is released after being pressed, the keyPressed method shall set the user rotational velocity to zero by calling the stopTurning method for user 1

3.10.3.9.2.3. If the space bar key is released after being pressed, the keyPressed method shall use the toggleProjectile() method for user 1 and set the value to false

3.10.3.9.3. The second user shall use the A, D and shift key as inputs

3.10.3.9.4. If the D arrow key is released after being pressed, the keyPressed method shall set the user rotational velocity to zero by calling the stopTurning method for user 2

3.10.3.9.5. If the A key is released after being pressed, the keyPressed method shall set the user rotational velocity to zero by calling the stopTurning method for user 2

3.10.3.9.6. If the shift key is released after being pressed, the keyPressed method shall use the toggleProjectile() method for user 2 and set the value to false

3.10.3.10. double getUserSpeed(int UserID)

3.10.3.10.1. This method shall return a double corresponding to the next forward velocity of the user vehicle specified by UserID

3.10.3.11. double getUserOmega(int UserID)

3.10.3.11.1. This method shall return a double corresponding to the next rotational velocity of the user vehicle specified by UserID

3.10.3.12. boolean getProjectileGenerated(int UserID)

3.10.3.12.1. This method shall return a boolean representing whether or not the user vehicle specified by UserID is currently shooting

3.10.3.13. drawCircle(Graphics g, int Xc, int Yc, int R)

3.10.3.13.1. 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

3.10.3.13.2. This method shall draw a circle of radius R centered at (Xc, Yc) on the display

3.10.3.14. drawProjectiles(Graphics g)

3.10.3.14.1. This method shall iterate over the lists of Projectile x and y locations pX[] and pY[] and draws a circle of radius 1 pixel at each Projectile location

3.10.3.14.2. Each projectile drawn shall be colored according to the color index in the pC[] array

3.10.3.15. drawScores(Graphics g)

3.10.3.15.1. This method shall display each user's number of shots, hits, and kills on the display

3.10.3.15.1.1. If the game is in single-player mode, scores for one user shall be displayed

3.10.3.15.1.2. If the game is in multi-player mode, scores for both users shall be displayed

3.10.3.15.2. The drawScores() method shall display each user's percentage accuracy in shooting, measured as accuracy = 100 · (hits / shots)

3.10.3.16. drawHelp(Graphics g)

3.10.3.16.1. If the value of HELP is set to true, this method shall display a help menu explaining the different key controls for the user

3.10.3.16.1.1. The drawHelp() method shall display the key control menu for user 1 if the value of FinalProjectile.MULTIPLAYER is false

3.10.3.16.1.2. The drawHelp() method shall display the key control menus for both user 1 and user 2 if the value of FinalProjectile.MULTIPLAYER is true

3.10.3.17. gameOver(Graphics g)

3.10.3.17.1. This method shall display the final score and the text "GAME OVER" on the screen

# 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.

# 6. Other Nonfunctional Requirements

## 6.1. Performance Requirements

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

## 6.2. Safety Requirements

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

## 6.3. Security Requirements

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

## 6.4. Software Quality Attributes

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