1. Control Requirements
    1.1. Variables
        1.1.1. The Control shall contain the internal representation of forward speed s
            1.1.1.1. s shall be in the interval [GroundVehicle.MIN_VEL, GroundVehicle.MAX_VEL], where GroundVehicle.MIN_VEL and GroundVehicle.MAX_VEL take values specified in 3.1.3.2 and 3.1.3.3
        1.1.2. The Control shall contain the internal representation of an angular velocity omega
            1.1.2.1. omega shall be in the interval $(-\pi, \pi]$
    1.2. Constructor
        1.2.1. Control(double s, double omega)
            1.2.1.1. The constructor shall take two arguments
            1.2.1.2. The internal representation of s and omega shall be initialized according to the constructor arguments
            1.2.1.3. The constructor shall initialize the variables to values within the intervals specified in 1.1.1.1 and 1.1.2.1
                1.2.1.3.1. If the s value falls outside the allowable range specified in 1.1.1.1, an IllegalArgumentException shall be thrown
                1.2.1.3.2. If the omega value falls outside the allowable range specified 1.1.2.1, an IllegalArgumentException shall be thrown
    1.3. Methods
        1.3.1. double getSpeed()
            1.3.1.1. This method shall return a double corresponding to s, the internal representation of forward speed
        1.3.2. double getRotVel()
            1.3.2.1. This method shall return a double corresponding to omega, the internal representation of angular velocity

2. Simulator Requirements
    2.1. Variables
        2.1.1. The Simulator shall contain internal representations of the x and y dimensions of the simulation size, SIM_X and SIM_Y
        2.1.2. The values of SIM_X and SIM_Y shall be constant
        2.1.3. The values of SIM_X and SIM_Y shall be initialized according to the display dimensions set in the DisplayServer
            2.1.3.1. The x dimension of the simulation shall be initialized to SIM_X = DisplayServer.DISPLAY_X / 5
            2.1.3.2. The y dimension of the simulation shall be initialized to SIM_Y = DisplayServer.DISPLAY_Y / 5
        2.1.4. The Simulator shall contain a reference to the UserController object it is associated with
        2.1.5. The Simulator shall contain an integer representing the millisecond duration between sending updates to the DisplayClient, SIM_MS_INCREMENT
            2.1.5.1. The value of SIM_MS_INCREMENT shall be constant
            2.1.5.2. The millisecond increment between updates shall be initialized to SIM_MS_INCREMENT = 50
    2.2. Constructor

2.3. Methods
- 2.3.1. addUserController(UserController uc)
  - 2.3.1.1. This method shall take one argument, a UserController object
  - 2.3.1.2. The addUserController method shall set the reference to the Simulator's UserController to point to the object present in the uc argument
- 2.3.2. generateProjectile()
  - 2.3.2.1. This method shall create a new Projectile object p
    - 2.3.2.1.1. The Projectile shall be created with the double array representing the position of the user vehicle, and the Simulator, passed in the constructor arguments
  - 2.3.2.2. The generateProjectile method shall add p to the Simulator's list of Projectiles
- 2.3.3. removeOffscreenProjectiles()
  - 2.3.3.1. 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 2.1.3.1 and 2.1.3.2

3. GroundVehicle Requirements

3.1. Variables
- 3.1.1. The GroundVehicle shall contain the internal representations of x and y position in two-dimensional space
  - 3.1.1.1. x shall be in the interval [0, Simulator.SIM_X]
  - 3.1.1.2. y shall be in the interval [0, Simulator.SIM_Y]
- 3.1.2. The GroundVehicle shall contain the internal representation of a heading angle θ
  - 3.1.2.1. θ will describe the the GroundVehicle's orientation in two-dimensional space
  - 3.1.2.2. θ shall be in the interval $[-\pi, \pi)$
- 3.1.3. The GroundVehicle shall contain internal representations of the minimum and maximum forward velocity, MIN_VEL and MAX_VEL
  - 3.1.3.1. The values of MIN_VEL and MAX_VEL shall be constant
  - 3.1.3.2. The minimum forward velocity shall be initialized to MIN_VEL = 1
  - 3.1.3.3. The maximum forward velocity shall be initialized to MAX_VEL = 10
- 3.1.4. The GroundVehicle shall contain internal representations of the linear velocities x' and y '
  - 3.1.4.1. $\sqrt{x'^2 + y'^2}$ shall be in the interval [MIN_VEL, MAX_VEL]
- 3.1.5. The GroundVehicle shall contain the internal representation of an angular velocity θ'
  - 3.1.5.1. θ' shall be in the interval $[-\pi/4, \pi/4]$
- 3.1.6. The GroundVehicle shall contain a reference to the Simulator object it is associated with
- 3.1.7. The GroundVehicle shall contain an integer representing the millisecond duration between vehicle state updates, GV_MS_INCREMENT
  - 3.1.7.1. The value of GV_MS_INCREMENT shall be constant
  - 3.1.7.2. The millisecond increment between vehicle state updates shall be initialized to GV_MS_INCREMENT = 100

3.2. Constructor
- 3.2.1. GroundVehicle(double pose[3], double dx, double dy, double dtheta)
  - 3.2.1.1. The constructor shall take four arguments

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

3.2.1.1.2. An IllegalArgumentException shall be thrown if any of the arguments' types differ from those specified in 3.2.1

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

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

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

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

3.2.1.2.4. The internal representation of the angular velocity shall be initialized as θ' = dtheta

3.2.1.3. The constructor shall initialize the variables to values within the intervals specified in 3.1.1.1, 3.1.1.2, 3.1.2.2, 3.1.4.1 and 3.1.5.1

3.2.1.3.1. If an element in the array pose[3] falls outside the allowable interval specified in in 3.1.1.1, 3.1.1.2 and 3.1.2.2, the internal representation of that position variable shall be clamped at the nearest limit

3.2.1.3.1.1. 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

3.2.1.3.1.2. 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

3.2.1.3.2. If a velocity value falls outside the allowable intervals specified in 3.1.4.1 and 3.1.5.1, the internal representation of that velocity shall be clamped at the nearest limit as specified in 3.3.8

3.2.1.3.2.1. 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

3.2.1.3.2.2. 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

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

3.2.2.1. The constructor shall take three arguments

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

3.2.2.1.2. An IllegalArgumentException shall be thrown if any of the arguments' types differ from those specified in 3.2.1

3.2.2.2. The internal representation of x, y, θ, x', y', θ' shall be initialized according to the constructor arguments

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

3.2.2.2.2. The linear velocities shall be calculated based on forward speed s and heading angle θ according to equations 3.2.2.2.2.1 and 3.2.2.2.2.2

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

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

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

3.2.2.3. The constructor shall initialize the variables to values within the intervals specified in 3.1.1.1, 3.1.1.2, 3.1.2.2, 3.1.4.1 and 3.1.5.1

3.2.2.3.1. If an element in the array pose[3] falls outside the allowable interval specified in in 3.1.1.1, 3.1.1.2 and 3.1.2.2, the internal representation of that position variable shall be clamped at the nearest limit as described in 3.2.1.3.1

3.2.2.3.2. If a velocity value falls outside the allowable intervals specified in 3.1.4.1 and 3.1.5.1, the internal representation of that velocity shall be clamped at the nearest limit as specified in 3.2.1.3.2

3.3. Methods

3.3.1. double [] getPosition()

3.3.1.1. This method shall return an array of 3 doubles, corresponding to the x, y, θ position and orientation of the vehicle

3.3.2. double [] getVelocity()

3.3.2.1. This method shall return an array of 3 doubles, corresponding to the x', y', θ' linear and angular velocities of the vehicle

3.3.3. setPosition(double [3])

3.3.3.1. This method shall take one argument, an array of 3 doubles, corresponding to the x, y, θ position of the vehicle.

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

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

3.3.3.3. If the setPosition method attempts to exceed the position constraints in 3.1.1.1, 3.1.1.2, and 3.1.2.2, the position shall be clamped as described in 3.2.1.3.1

3.3.4. setVelocity(double [3])

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

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

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

3.3.4.2.1. If the setVelocity method attempts to exceed the velocity constraints specified in 3.1.4.1 and 3.1.5.1, the internal representations of the resulting velocities shall be clamped as described in 3.2.1.3.2

3.3.5. controlVehicle(Control c)

3.3.5.1. This method shall modify the internal representations of the x', y', θ' velocities according to the specified forward speed and rotational velocity in the Control c argument.

3.3.5.1.1. x' and y' shall be calculated based on forward speed s and heading angle θ as described in 3.2.2.2

3.3.5.1.2. θ' shall be initialized to the value of rotational velocity omega

3.3.5.2. The internal representations of the velocities that result from applying the control must obey the same limits as setVelocity in 3.3.4.2.1

3.3.5.2.1. If the controlVehicle method attempts to exceed the velocity constraints in 3.1.4.1 and 3.1.5.1, the velocity shall be clamped as described in 3.2.1.3.2

3.3.6. updateState(int sec, int usec)

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

3.3.6.1.1. The arguments shall be combined according to the equation 3.3.6.1.1.1, where t[s] = sec and t[μs] = usec

3.3.6.1.1.1. $t = t[s] + t[\mu s] \cdot 10{-}3$

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

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

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

3.3.7. clampPosition()

3.3.7.1. If the internal representations of x, y, θ fall outside the allowable intervals specified in 3.1.1.1, 3.1.1.2 and 3.1.2.2, the clampPosition method will clamp the position and heading angle to the nearest limit

3.3.7.1.1. 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

3.3.7.1.2. 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

3.3.8. clampVelocity()

3.3.8.1. If the internal representations of linear velocity fall outside the allowable intervals specified in 3.1.4.1 and 3.1.5.1, the clampVelocity method will clamp the velocity to values within the allowable intervals

3.3.8.1.1. 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

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

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

3.3.8.1.2. 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

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

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

3.3.8.2. If the internal representation of angular velocity θ' falls outside the allowable interval specified in 3.1.5.1, the clampVelocity method will clamp the angular velocity to a value within the allowable interval

3.3.8.2.1. 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

3.3.8.2.2. 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

3.3.9. setSimulator(Simulator sim)

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

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

4. VehicleController Requirements

4.1. Variables

4.1.1. The VehicleController shall contain a reference to the Simulator object it is associated with

4.1.2. The VehicleController shall contain a reference to the GroundVehicle object it is associated with

4.1.3. The VehicleController shall contain an internal representation of the time when the last control was issued

4.1.4. The VehicleController shall contain an integer representing the millisecond duration between successive controls being issued, VC_MS_INCREMENT

4.1.4.1. The value of VC_MS_INCREMENT shall be constant

4.1.4.2. The millisecond increment between controls shall be initialized to VC_MS_INCREMENT = 100

4.2. Constructor

4.2.1. VehicleController(Simulator s, GroundVehicle v)

4.3. Methods

4.3.1. run()

4.3.2. double normalizeAngle(double theta)

4.3.2.1. This method shall take one argument, a double representing an angle

5. UserController Requirements

5.1. Variables

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

5.2. Constructor

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

5.2.1.1. The constructor shall take three arguments

5.2.1.2. The UserController shall set its internal Simulator, GroundVehicle, and DisplayServer references to the objects present in the sim, v, and ds arguments

5.3. Methods

5.3.1. GroundVehicle getUserVehicle()

5.3.1.1. This method shall return the GroundVehicle associated with the UserController

5.3.2. Control getControl(int sec, int msec)

5.3.2.1. 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 7.2.3 and 7.2.4

5.3.2.2. If any of the resulting velocities exceed the velocity constraints in 3.1.4.1 and 3.1.5.1, those velocity values shall be clamped as described in 3.2.1.3.2

6. Projectile Requirements

6.1. Variables

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

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

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

6.1.1.3. 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 2.3.3

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

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

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

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

6.1.3.1. The value of PROJECTILE_SPEED shall be constant

6.1.3.2. The value of the projectile forward velocity shall be initialized to PROJECTILE_SPEED = 3 · GroundVehicle.MAX_VEL

6.1.4. The Projectile shall contain internal representations of the linear velocities x' and y '

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

6.1.5. The Projectile shall contain a reference to the Simulator object it is associated with

6.2. Constructor

6.2.1. Projectile(double[3] shooterPosition, Simulator sim)

6.2.1.1. The constructor shall take two arguments

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

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

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

6.2.1.2.2. The linear velocities shall be calculated based on projectile forward velocity PROJECTILE_SPEED and the heading angle θ according to equations 6.2.1.2.2.1 and 6.2.1.2.2.2

6.2.1.2.2.1. x' = PROJECTILE_SPEED · cos(θ)

6.2.1.2.2.2. y' = PROJECTILE_SPEED · sin(θ)

6.3. Methods

6.3.1. double [] getPosition()

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

6.3.2. shoot(int sec, int msec)

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

6.3.2.1.1. The arguments shall be combined according to the equation 6.3.2.1.1.1, where t[s] = sec and t[μs] = usec

6.3.2.1.1.1. $t = t[s] + t[\mu s] \cdot 10{-}3$

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

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

6.3.2.2.1. The shoot method shall change the internal representation of the x and y position according to the dynamics in equations 6.3.2.2.2 and 6.3.2.2.3

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

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

6.3.3. run()

7. DisplayServer Requirements

7.1. Variables

7.1.1. The DisplayServer shall contain an internal representation of the maximum absolute value of rotational velocity for the user vehicle, MAX_OMEGA

7.1.1.1. The maximum absolute value of rotational velocity shall be initiaized to MAX_OMEGA = π/4

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

7.1.2.1. The forward velocity increment shall be initalized to SPEED_INCREMENT = 0.5

7.1.3. The DisplayServer shall contain internal representations of the next forward velocity and rotational for the user vehicle, nextSpeed and nextOmega

7.1.3.1. The foward velocity starting value shall be initialized to nextSpeed = 1

7.1.3.2. The rotational velocity starting value shall be initialized to nextOmega = 0

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

7.1.4.1. The number of Projectiles shall be initialized to numProjectiles = 0

7.1.5. 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

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

7.1.7. The values of DISPLAY_X and DISPLAY_Y shall be constant

7.1.7.1. The x dimension of the display shall be initialized to DISPLAY_X = 800

7.1.7.2. The y dimension of the display shall be initialized to DISPLAY_Y = 600

7.2. Methods

7.2.1. startGraphics()

7.2.1.1. The startGraphics() method shall set the dimensions of the display window to the x and y values specified in 7.1.7.1 and 7.1.7.2

7.2.2. keyPressed(KeyEvent e)

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

7.2.2.2. If the right arrow key is pressed, the value of nextOmega shall be set to nextOmega = MAX_OMEGA

7.2.2.3. If the left arrow key is pressed, the value of nextOmega shall be set to nextOmega = - MAX_OMEGA

7.2.2.4. If the up arrow key is pressed, the value of nextSpeed shall be set to nextSpeed = nextSpeed + SPEED_INCREMENT

7.2.2.5. If the down arrow key is pressed, the value of nextSpeed shall be set to nextSpeed = nextSpeed - SPEED_INCREMENT

7.2.2.6. If the space bar is pressed, a projectile shall be generated in Simulator as specified in 2.3.2

7.2.3. double getUserSpeed()

7.2.3.1. This method shall return a double corresponding to the next forward velocity of the user vehicle, nextSpeed

7.2.4. double getUserOmega()

7.2.4.1. This method shall return a double corresponding to the next rotational velocity of the user vehicle, nextOmega

7.2.5. drawProjectiles(Graphics g)

7.2.5.1. This method shall iterate over the lists of Projectile x and y locations and draws a circle of radius 1 at each Projectile location

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

7.2.6.1. 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

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