

# Artillery Design

## Class Diagrams

---

### Angle

Everything we need to know about an angle.

Angle
- radians : Double
+ Angle() + assign(rhs : Angle) + setRadians(rhs : Double) + setDegrees(rhs : Double) + setDxDy(dx, dy) + setDown() + setRight() + setLeft() + reverse() + add(delta : Double) + getDy() : Double + getRadians() : Double + getDegrees() : Double + getDx() : Double + getDy() : Double + isLeft() : Bool + isRight() : Bool - normalize(rhs) : Double - convertToRadians(rad) - convertToDegrees(deg)

### Acceleration

Everything we need to know about acceleration.

Acceleration
- ddx - ddy
+ Acceleration() + Acceleration(ddx, ddy) + getDDX() + getDDY() + setDDX(ddx) + setDDY(ddy) + addDDX(ddx) + addDDY(ddy) + add(rhs : Acceleration)

## Velocity

Everything we need to know about speed.

Velocity
- dx - dy
+ Velocity() + getDX() : Double + getDY() : Double + getSpeed() : Double + getAngle() : Angle + setDX(dx : Double) + setDY(dy : double) + setDxDy(dx, dy) + setAngle(Angle) + setSpeed(velocity) + addDX(dx) + addDY(dy) + addV(velocity : Velocity) + reverse()

## Position

Everything we need to know about the position.

Position
- x : Double - y : Double - <u>metersFromPixels</u>
+ getMetersX() : Double + getMetersY() : Double + setMetersX(x : Double) + setMetersY(y : Double) + setPixelsX(x : Double) + setPixelsY(y : Double) + addMetersX(x : Double) + addMetersY(y : Double) + setZoom(ratio) + add(accel, vel, time)

## Projectile

A projectile, including how it flies and where it is located.

Projectile
<ul style="list-style-type: none"> <li>- mass : Double</li> <li>- radius : Double</li> <li>- flightPath : [ ]</li> </ul>
<ul style="list-style-type: none"> <li>+ Projectile()</li> <li>+ reset()</li> <li>+ fire(pos, time, angle, vel)</li> <li>+ advance(time)</li> <li>+ draw(gout)</li> <li>+ flying() : Bool</li> <li>+ getAltitude() : Double</li> <li>+ getPostion() : Position</li> <li>+ getFlightTime() : Double</li> <li>+ getFlightDistance() : D</li> <li>+ getSpeed() : Double</li> <li>+ getCurrentTime() : D</li> <li>+ setMass(mass)</li> <li>+ setRadius(radius)</li> </ul>

## Howitzer

The gun, including where it is located and where it is pointed.

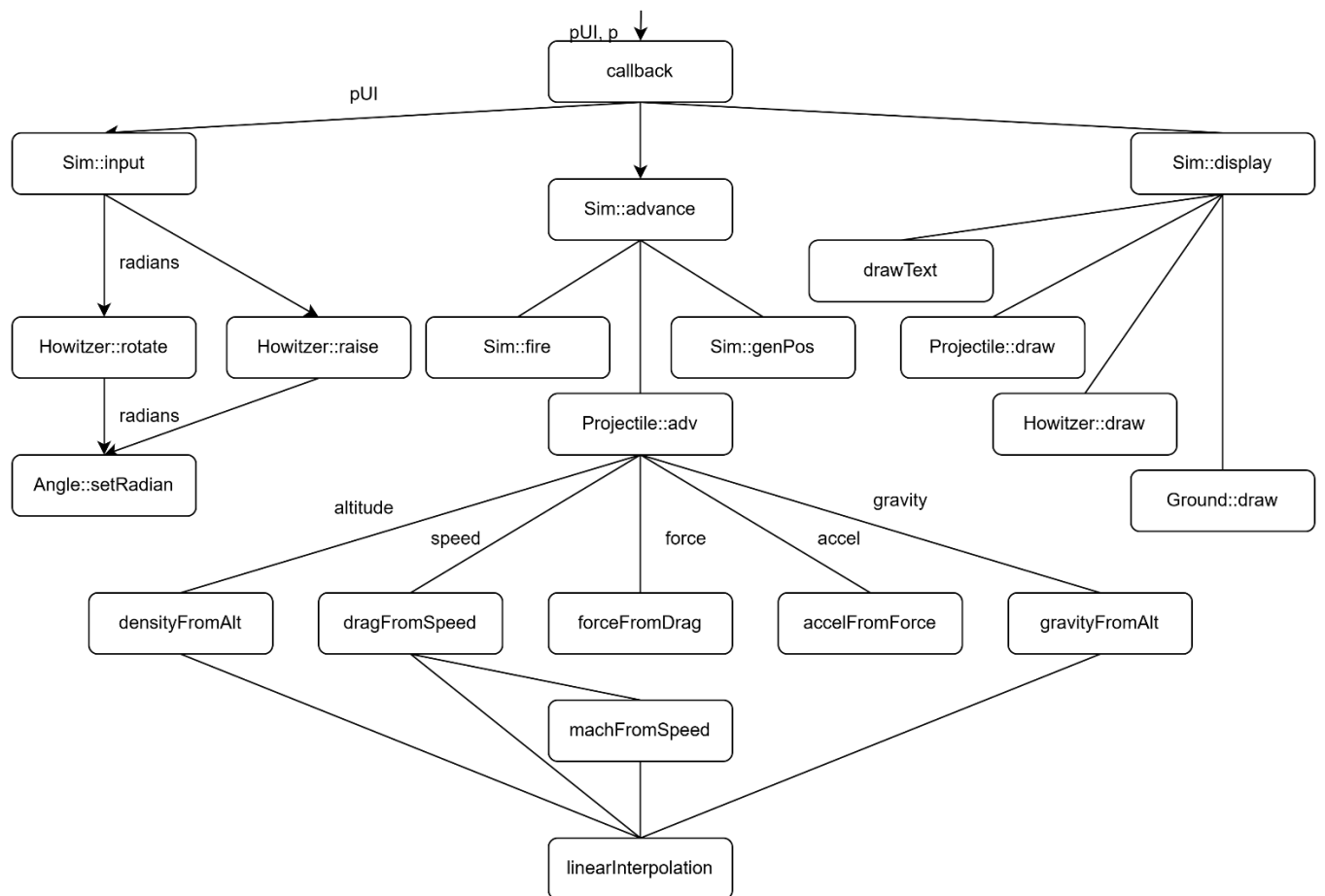
Howitzer
<ul style="list-style-type: none"> <li>- position : Pos</li> <li>- muzzleVelocity : Double</li> <li>- elevation : Direction</li> </ul>
<ul style="list-style-type: none"> <li>+ Howitzer()</li> <li>+ draw(gout, flightTime)</li> <li>+ getPostion() : Position</li> <li>+ generatePosition(size)</li> <li>+ getMuzzleVelocity() : vel</li> <li>+ setMuzzleVelocity(vel)</li> <li>+ rotate(radians)</li> <li>+ raise(radians)</li> </ul>

## Simulator

This connects all the elements together. It does not know how the projectile flies or anything about the gun, but it does control the relationship between those elements.

Simulator
<ul style="list-style-type: none"> <li>- interval : Double</li> <li>- howitzer : Howitzer</li> <li>- projectile : Projectile</li> <li>- status : status</li> <li>- simTime : Double</li> <li>- ground : Ground</li> </ul>
<ul style="list-style-type: none"> <li>+ Simulation(...)</li> <li>+ reset()</li> <li>+ fire()</li> <li>+ display()</li> <li>+ advance()</li> <li>+ input(Interface)</li> <li>+ setInterval(interval)</li> <li>+ setMuzzleVelocity(vel)</li> <li>+ setDiameter(diameter)</li> <li>- hitTarget()</li> <li>- getHeightMeters()</li> </ul>

## Structure Chart



## Pseudocode

The main function to handle all the physics is called in `Projectile::advance()`. This moves the projectile along the path taking into account all the forces which act on it.

```
Projectile.advance()
  Pvt ← flightpath.back()
  Speed ← pvt.v.getSpeed()
  Altitude ← pvt.pt.getMetersY()

  MODIFY VELOCITY TO HANDLE WIND RESISTANCE
  Density ← densityFromAltitude(altitude)
  dragCoefficient ← dragFromSpeed(speed, altitude)
  windResistance ← forceFromDrag(density, dragCoefficient, radius, speed)
  accelerationDrag ← accelerationFromForce(windResistance, mass)
  velocityWind ← velocityFromAcceleration(accelerationDrag, interval),
    pvt.v.getDirection()
  velocityWind.reverse()
  pvt.v += velocityWind

  MODIFY VELOCITY TO HANDLE GRAVITY
  accelerationGravity ← gravityFromAltitude(altitude)
  velocityGravity ← velocityFromAcceleration(accelerationGravity, interval)
    Angle.setDown()
  Pvt.v += velocityGravity

  INERTIA
  Pvt.pt.addMetersX(velocityFromAcceleration(pvt.v.getDX(), interval)
  Pvt.pt.addMetersY(velocityFromAcceleration(pvt.v.getDY(), interval)

  ADD IT TO THE BACK OF THE FLIGHT PATH
  Flightpath.push_back(pvt);
```

The callback function does three things: handle input, performs processing, and handles output.

```
Callback(UI, sim)
  Sim.input(UI)
  Sim.advance()
  Sim.display()
```