Lab 08 Artillery Design

# Class Diagrams:

Simulator:

The class has a fidelity of complete because it completely matches and represents the needs of the design concern.

Robustness level is fragile because it is currently not tested only reviewed by two engineers.

Convenience is seamless because the client needs only call one function Simulator :: Run to use the class.

Abstraction is complete because no implementation details are leaked to the client, everything is wrapped in simulator :: run.

Simulator

+Simulator(ptUpperRight: Position)

+Run(pUI: interface, gout: ogstream)

-doInput(pUI: interface)

-update()

-draw(gout: ogstream)

-getKeys() : [bool]

-applyKeys(keys : [bool])

-getDrag() : double

-getGravity() : double

-checkCollisions()

-drawText(gout: ogstream)

-ptUpperRight: Position

-howitzer: Howitzer

-projectile: Projectile

-ground: Ground

-physics: Physics

- time: double

-keysIndex: enum

Ground:

The class has a fidelity of complete because it completely matches and represents the needs of the design concern.

Robustness level is fragile because it is currently not tested only reviewed by two engineers.

Convenience is seamless because all the needs of the ground class are handled and data is returned in a valid and easy to use state to the client.

Abstraction is complete because there are no implementation details revealed to the client as we provide all necessary methods the client needs.

Ground

-ground: [double]

-posUpperRight: Position

-posTarget: Position

+Ground(posUpperRight: Position)

+reset(posHowitzer: Position)

+draw(gout: ogstream)

+hitGround(posProjectile: position) : Bool

+hitTarget(posProjectile: position) : Bool

+getElevationMeters(pos: position) : Double

-getTarget() : Position

## Physics

Fidelity: Complete. The LookUps represent all values that this game could possibly have.

Robustness: Fragile. No testing has been done.

Convenience: Easy. The arguments for getGravity and getDragForce need to be acquired, but they are easy to get from the Projectile.

Abstraction: Complete. It is not possible to know how the class is implemented from the methods.

Physics

-dragTable : DragLookUp

-gravityTable : GravityLookUp

-airDensityTable : AirDensityLookUp

-machTable : MachLookUp

+Physics()

+getGravity(altitude : double) : double

+getDragForce(altitude : double, speed : double, area : double) : double

## Look Ups:

The class has a fidelity of complete because it completely matches and represents the needs of the design concern and nothing extra is done.

Robustness level is fragile because it is currently not tested only reviewed by two engineers.

Convenience is seamless because all the extra work to interpolate and is done by the class and returned conveniently to the client.

Abstraction is complete because the client does not know anything about how the values are stored or found. They only know that the interface returns a tuple of long doubles.

Look Up

+searchTable() : (long double)

-interpolate(x0: long double, y0: long double, x1: long double, y1: long double, current\_x: long double) : long double

-*table : [(double, double)]*

Gravity Look Up

-*table : [(double, double)]*

Air Density Look Up

-*table : [(double, double)]*

Mach Look Up

-*table : [(double, double)]*

Drag Look Up

-*table : [(double, double)]*

## Angle:

Fidelity: Complete. The choice of double for degrees allows for precise calculations. The value will be limited to the range -180 to +180 so that it is easy to maintain good state.

Robustness: Fragile. No testing has been done.

Convenience: Easy. All desired adding, setting, and getting is implemented.

Abstraction: Complete. The only possible way for the user to know how this class is implemented would be to do a performance test.

Angle

-degrees : double

+Angle()

+Angle(angle : Angle)

+Angle(x : double, y : double)

+setDegrees(degrees : double)

+setRadians(radians : double)

+addDegrees(amount : double)

+addRadians(amount : double)

+getDegrees() : double

+getRadians() : double

-radiansFromDegrees(radians : double) : double

-degreesFromRadians(degrees : double) : double

-degreesFromXY(x : double, y : double) : double

## Vector2D

This is a base class that Acceleration and Velocity will inherit from

Fidelity: Complete. X and Y being doubles give good precision, and also covers all possible values.

Robustness: Fragile. No testing has been done.

Convenience: Seamless. All the main actions a Vector2D could take are represented here.

Abstraction: Complete. It is not possible to know how this class is implemented from the methods.

Vector2D

-x : double

-y : double

+Vector2D()

+Vector2D(vect: Vector2D)

+Vector2D(x : double, y : double)

+Vector2D(angle : Angle, magnitude : double)

+getAngle() : Angle

+getMagnitude() : double

+getX() : double

+getY() : double

+addVector2D(otherVect: Vector2D)

## Velocity

Velocity privately inherits from Vector2D.

Fidelity: Complete. This class is a Vector2D, so it has an x and y as a double that exactly cover the values this class needs to have.

Robustness: Fragile. No testing has been done.

Convenience: Seamless. All the main actions Velocity needs to take are represented here.

Abstraction: Complete. It is not possible to know how this class is implemented from the methods.

Velocity

None

+Velocity()

+Velocity(velocity : Velocity)

+Velocity(dx : double, dy : double)

+Velocity(angle : Angle, speed : double)

+getDirection() : Angle

+getSpeed() : double

+getDX() : double

+getDY() : double

+addAcceleration(acceleration : Acceleration, time : double)

## Acceleration

Acceleration privately inherits from Vector2D.

Fidelity: Complete. This class is a Vector2D, so it has an x and y as a double that exactly cover the values this class needs to have.

Robustness: Fragile. No testing has been done.

Convenience: Seamless. All the main actions Acceleration needs to take are represented here.

Abstraction: Complete. It is not possible to know how this class is implemented from the methods.

Acceleration

None

+Acceleration()

+Acceleration(acceleration : Acceleration)

+Acceleration(ddx : double, ddy : double)

+Acceleration(angle : Angle, magnitude : double)

+getDDX() : double

+getDDY() : double

+addAcceleration(acceleration : Acceleration)

## Projectile

Fidelity: Complete. The Position object will handle all possible locations the projectile could be in, Velocity will handle all possible velocities, the age of the projectile is a float because it does not need to be very accurate but it does need to handle decimal values, the weight is stored as a double for precision and is constant because it won't change (same for the shell's area), and the status is limited to the only four states the projectile could be in.

Robustness: Fragile. No testing has been done.

Convenience: Seamless. All methods take no parameters besides exactly what the Simulator has to offer, and they all do exactly what the Simulator needs.

Abstraction: Complete. It isn't difficult to tell what some datatypes being used are, but it would not be difficult for me to change what datatypes are being used and the user wouldn't know.

Projectile

-shadows : [Position]

-position : Position

-velocity : Velocity

-age : float

-WEIGHT : double

-AREA : double

-status : enum{LOADED, FLYING, ON\_TARGET, OFF\_TARGET}

+Projectile() : Projectile

+Projectile(point : Position) : Position

+move(dragForce : double, gravityAcceleration : double, time : double) : void

+fire(velocity : Velocity) : void

+hitTarget() : void

+missedTarget() : void

+getPosition() : Position

+getLastPosition() : Position

+getSpeed() : double

+getVelocity() : Velocity

+getAngle() : Angle

+getArea() : double

+getAge() : float

+isLoaded() : bool

+isFlying() : bool

+isImpacted() : bool

+isOnTarget() : bool

+draw(gout : OGStream) : void

-shiftShadows() : void

## Howitzer

Fidelity: Complete. Position covers all places Howitzer could be, and the constants are as precise as they need to be, though Angle does allow for more angles than the Howitzer needs. Some limiting will need done in the changeAngle function to make sure it is not in an extraneous state.

Robustness: Fragile. No testing has been done.

Convenience: Easy. Passing age to the draw function takes a tiny bit of effort, but it would be more difficult to pass in the elapsed time between each draw and have the Howitzer keep track of its own age.

Abstraction: Complete. It is not possible to know how this class is implemented from the methods.

Howitzer

-angle : Angle

-position : Position

-MUZZLE\_VELOCITY : double

-SMALL\_ANGLE\_CHANGE : double

-LARGE\_ANGLE\_CHANGE : double

-FLASH\_TIME : int

+Howitzer() : Howitzer

+Howitzer(position Position) : Howitzer

+fire() : Projectile

+draw(gout : OGStream, age) : void

+getAngle() : Angle

+setPosition(position : Position) : void

+smallIncrement() : void

+smallDecrement() : void

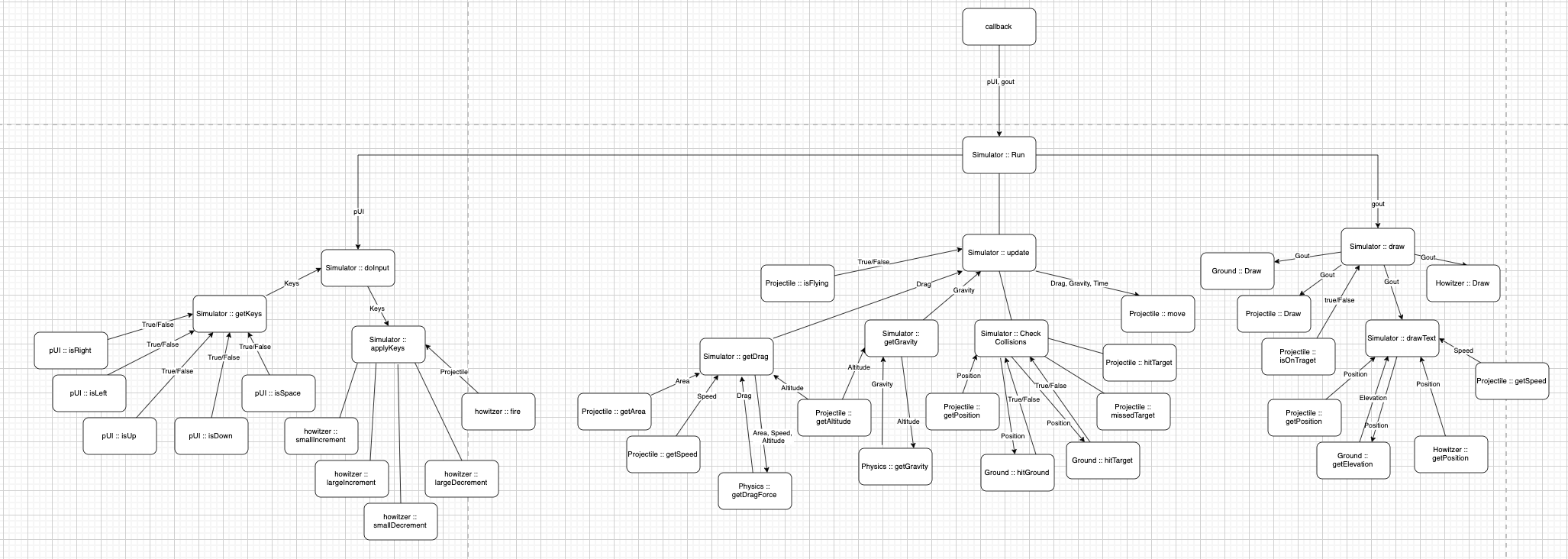
+largeIncrement() : void

+largeDecrement() : void

-changeAngle(radians : double) : void

# Structure Chart:

The whole structure chart

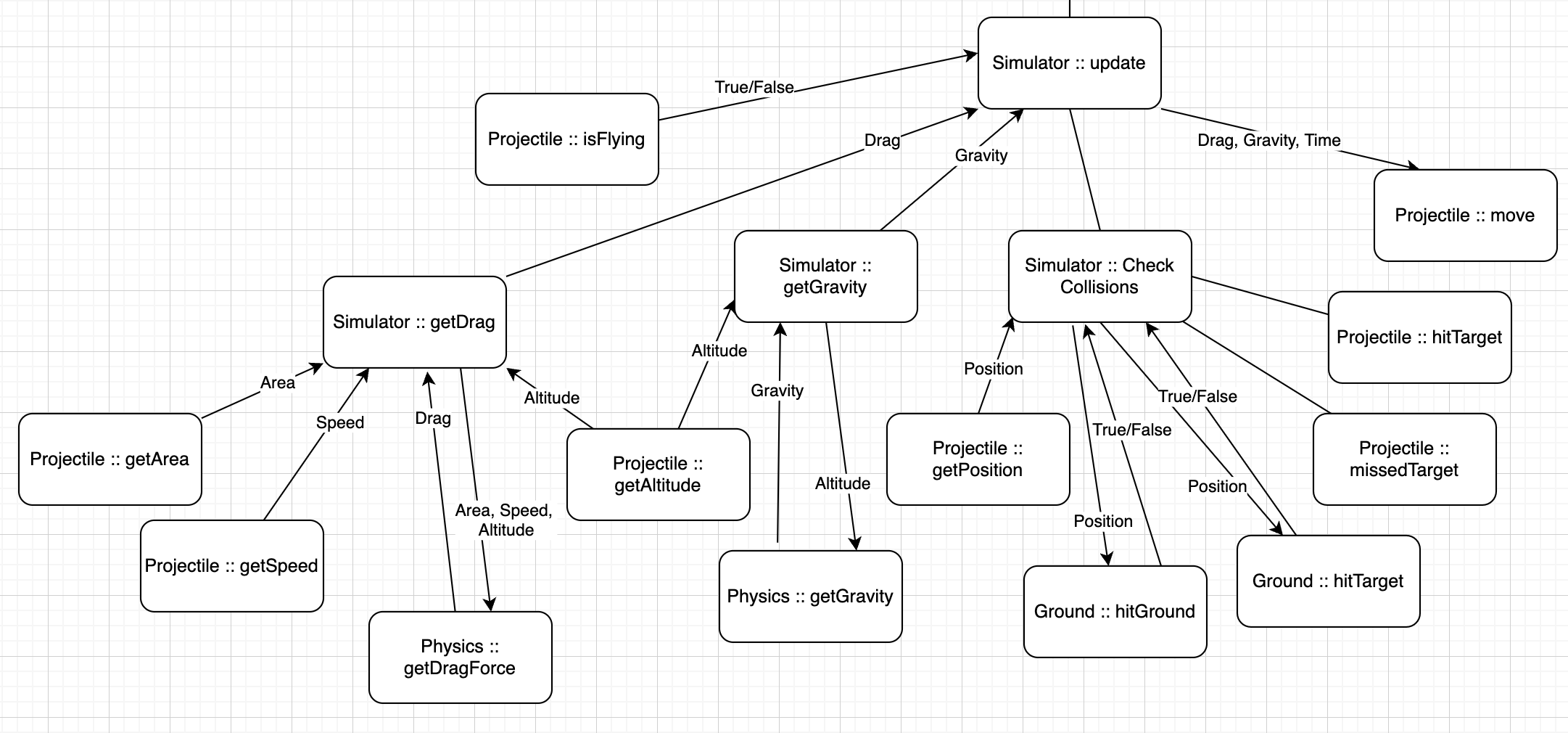


Simulator :: doInput

Diagram

Description automatically generated

Simulator :: update



Simulator :: draw

Diagram

Description automatically generated

# Pseudocode:

LookUp::SearchTable(notContainedKey, lookUpTable)

iterator ß lookUpTable [0]

high ß lookUpTable [0][0]

low ß lookUpTable [0][0]

searching ß true

WHILE iterator != lookUpTable[-1] and searching = true

IF iterator[0] < notContainedKey

IF iterator[0] > low

low ß iterator

ELSE

high ß iterator[0]

searching ß false

iterator++

keys ß (low, high)

return keys

Projectile::move(dragForce, gravityAcceleration, time)

shiftShadows()

oppositeAngle <- Angle(velocity.getAngle())

oppositeAngle.addDegrees(180)

dragAcceleration <- Acceleration(oppositeAngle, dragForce / WEIGHT)

totalAcceleration <- Acceleration(0, -gravityAcceleration)

totalAcceleration.addAcceleration(dragAcceleration)

position.addVelocityAcceleration(velocity, totalAcceleration, time)

velocity.addAcceleration(totalAcceleration, time)

# Test Cases

## Test cases for Projectile::move()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Precondition | Inputs | Outputs | Rationale |
| Stationary | position = (0, 0), velocity = (0, 0), WEIGHT = 1 | df = 0  ga = 0  t = 1 | position = (0, 0)  velocity = (0, 0) | If the projectile is not in motion and no forces act on it, it should stay at rest. |
| Inertia Only | position = (1, 1), velocity = (3, 4), WEIGHT = 1 | df = 0 ga = 0  t = 1 | position = (4, 5)  velocity = (3, 4) | If the projectile has a velocity and no outside forces act on it, it should maintain that velocity. |
| Gravity Only | position = (1.7, 2.5), velocity = (0, 0), WEIGHT = 1 | df = 0  ga = 3  t = 0.5 | position = (1.7, 2.125)  velocity = (0, -1.5) | If only gravity acts on the object, then only the y-values should decrease. |
| Inertia and Drag | position = (-3, -2), velocity = (1, 1), WEIGHT = 1 | df = 1  ga = 0  t = 1 | position = (-2.354, -1.354)  velocity = (0.2929, 0.2929) | The drag force should directly oppose the velocity. |
| Inertia and Gravity | position = (10, 20), velocity = (-1, -2), WEIGHT = 1 | df = 0  ga = 2.5  t = 2 | position = (8, 11)  velocity = (-1, -7) | The acceleration due to gravity should only affect the y positions, posX can still be affected by velocityX. |
| Base Case | position = (15, 7), velocity = (50, 42), WEIGHT = 46.7 | df = 25  ga = 9.8  t = 0.5 | position = (39.957, 29.723)  velocity = (49.828, 31.790) | All values used, and all updated. |

## Test cases for Physics::getDragForce()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Precondition | Inputs | Outputs | Rationale |
| Zero Velocity | None | altitude = 0  speed = 0  area = 1 | 0.0 | If you have zero velocity, then there's nothing to resist and drag will have no effect. |
| Simple Case | None | altitude = 0  speed = 340  area = 1 | 30,148.769 | With these inputs, there is no interpolation necessary and the equation will be 1/2 \* .4258 \* 1.225 \* 340 \* 340 \* 1 = 30,148.769 |
| Interpolate | None | altitude = 500  speed = 135.2  area = 1 | 1755.715 | With these inputs, mach, drag coefficient, and density all need linearly interpolated and the equation will be 1/2 \* .1644 \* 1.1685 \* 135.2 \* 135.2 \* 1 |
| Max Table Values | None | altitude = 40000  speed = 1620  area = 1 | 1392.687 | With these inputs, the maximum altitude still on the two charts and the maximum speed are used and the equation will be 1/2 \* .2656 \* .003996 \* 1620 \* 1620 |
| Off The Charts | None | altitude = 100000  speed = 5000  area = 1 | Error | Inputs are not allowed to be outside the range of table values. |
| Standard | None | altitude = 0  speed = 827  area = .018843 | 2048.854 | With the values for a just-fired shell, the equation will be 1/2 \* .25956 \* 1.225 \* 827 \* 827 \* .018843 |