Austin Hilderbrand, Elijah Harrison
CSE 230 Lab 8
Artillery Sim Design Document

UML Class Diagrams

***functions in bold: pseudocode below**
**** highlighted classes: chosen for test cases**

Velocity
- dx - dy
+ initialize(dx_init: double, dy_init: double) + getDx() : double + getDy() : double + getSpeed() : double + == (v:Velocity) : bool operator + != (v:Velocity) : bool operator + setDx(newDx:double) + setDy(newDy:double) + set(v:Velocity) + set(dx:double, dy:double) + addDx(ddx:double) + addDy(ddy:double) + add(v:Velocity) + add(ddx:double, ddy:double) + addMagnitude(angleRadians:double, magnitude:double)

Fidelity: complete - only the necessary states are represented, and they are represented completely.
Robustness: fragile - no testing done yet
Convenience: straightforward - may require some simple data manipulation (converting to the correct angle type, the proper units for velocity, acceleration, etc.)
Abstraction: critical - the client would greatly benefit from understanding implementation details, including angle type, how the velocity/acceleration is stored and with what units, and the layout of the map.

Howitzer
- point : Point - screenTopRight : Point - angle : double - age : double
+ initialize(x_init:double) + update() **+ handleInput()** + draw() + fire() : Artillery

Fidelity: complete (possibly extraneous) - only the necessary states are represented, and they are represented completely. However, we are unsure at this stage if we will need screenTopRight.
Robustness: fragile - no testing done yet
Convenience: easy - no extra work necessary, but there are ways it could be improved to be even easier to use
Abstraction: porous - the client would benefit from knowing the implementation details, including how "age" works, the angle type required by the class, and how the angle translates into the orientation of the barrel.

Artillery
- point : Point - screenTopRight : Point - velocity : Velocity - time : double - artilleryV0 : const double - artilleryMass : const double - artilleryDiameterMm : const double
+ initialize() **+ update()** + draw() + getAltitude() : double + getSpeed() : double + getDistance() : double + getHangTime() : double

Fidelity: complete(possibly extraneous) - only the necessary states are represented, and they are represented completely. However, we are unsure at this stage if we will need screenTopRight

Robustness: fragile - no testing done yet

Convenience: straightforward - may require some simple data manipulation, such as making sure the correct parameter data types are passed

Abstraction: porous - the client would benefit from knowing implementation details, such as the format for "point" and "velocity", and the limitations of each

Grid
- screenTopRight : Point
+ initialize(screenTopRightInit:Point) **+ update()** + draw()

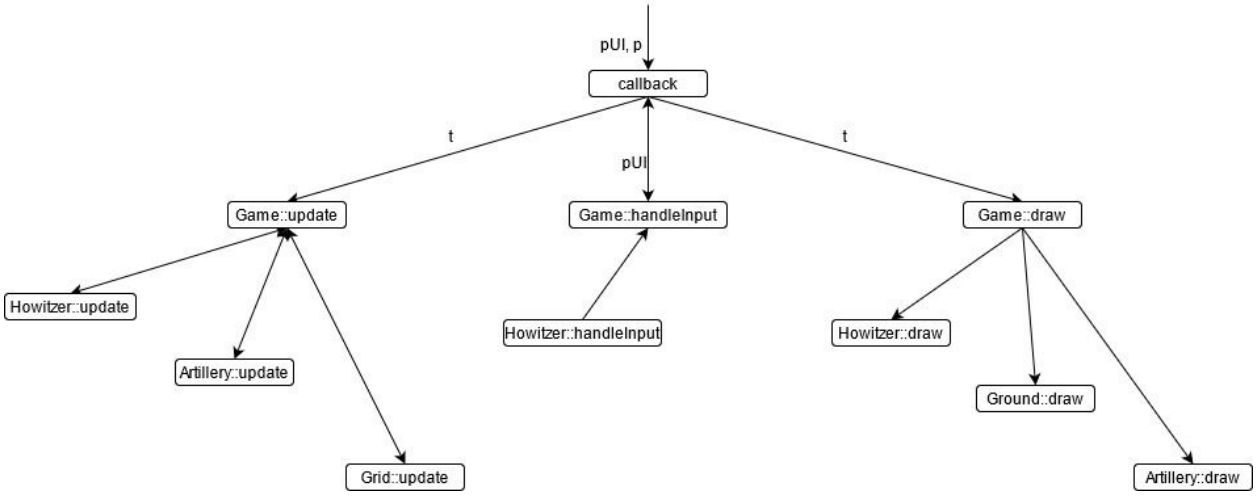
Fidelity: complete - only the necessary states are represented, and they are represented completely

Robustness: fragile - no testing done yet

Convenience: seamless - no data manipulation necessary and well-aligned with the needs of the program

Abstraction: complete - opaque, the only implementation detail which could be revealed to the client, or which could be useful for the client to know, is the limitations of the screenTopRight point object. This point needs to be within a certain range of values to be valid.

Structure Chart



Pseudocode

Howitzer::handleInput()

```
artillery.add(howitzer.fire())
```

Artillery::update()

```
// increment time by program-defined deltaTime
t                += dt

// update angle/speed/velocity
angle          = cartesianToAngle(dx, dy)
speed           = mag(dx, dy)
dx                = horizontalComponent(speed, angle)
dy                = verticalComponent(speed, angle)

// update gravity
g                 = -altitudeToGravity(y)

// update drag
c                 = machToDragCoefficient(speed)
p                 = altitudeToDensity(y)
artilleryRadius   = getArtilleryDiameter() * .5
a                 = circleArea(artilleryRadius)
dragF             = dragForce(c, p, speed, a)

// update x
ddx               = getAccelerationX(dragF, angle)
dx               += ddx
x                += dx

// update y
ddy               = getAccelerationY(g, dragF, angle)
dy               += ddy
y                += dy
```

Grid::update(artillery:Artillery)

```
this.altitude  = artillery.getAltitude()
this.speed     = artillery.getSpeed()
this.distance  = artillery.getDistance()
this.hangTime  = artillery.getHangTime()
```

Test Cases

Class under test: Velocity	Input parameters	Output
test_setDx_invalid	(‘string’)	ERROR: invalid
test_setDx_valid	(-100.00)	dx = -100.00
test_setDy_invalid	(‘string’)	ERROR: invalid
test_setDy_valid	(-100.00)	dy = -100.00
test_set_invalid	(‘string’, 100)	ERROR: invalid
test_set_valid	(100, 100)	dx = 100, dy = 100
test_addDx_invalid	(‘string’)	ERROR: invalid
test_addDx_valid	(40)	dx += 40
test_addDy_invalid	(‘string’)	Error: invalid
test_addDy_valid	(-40)	dy += -40
test_add_invalid	(‘string’, 40)	ERROR: invalid
test_add_valid	(40, 40)	dx += 40, dy += 40
test_addMagnitude_invalid	(2, -100)	ERROR: invalid
test_addMagnitude_valid	(2, 100)	dx += 100cos(2), dy += 100sin(2)

Class under test: Artillery	Input parameters	Output
test_initialize_invalidPoint	point = (-1, 1)	ERROR: invalid point
test_initialize_invalidVel	velocity = (1, 1)	ERROR: starting velocity should be 0
test_initialize_invalidTime	time = 10	ERROR: starting hang time should be 0
test_initialize_valid	((150, 200), (800, 800), (0 , 0), 0)	point = (150, 200) ScreenTopRight = (800, 800) velocity = (0, 0)

		time = 0
test_getAltitude_invalidTime	at t = -1	point.gety = 0
test_getAltitude_validTime	at t = 10	point.gety >= 0
test_getSpeed_invalidTime	at t = -1	velocity.getSpeed = 0
test_getSpeed_validTime	at t = 10	velocity.getSpeed >= 0
test_getDistance_invalidTime	at t = -1	point.getx = 0
test_getDistance_validTime	at t = 10	point.getx >= 0
test_getHangTime_invalidTime	at t = -1	time = 0
test_getHangTime_validTime	at t = 10	time >= 0