VGP126 Project (due Wed Sept 19)

W. Dobson 9-5-2018

Modify the **`fighter / jet`** class from assignment #6 to use vectors to store munitions and to track launched munitions until they hit the ground. This new method will require many changes to the classes. To do this we will create a common **`munition`** class that can be set to be either a bomb, rocket or missile at time of instantiation. Note that for this project **`x`** and **`y`** are horizontal coordinates and **`z`** is the vertical (altitude) coordinate.

Common `munition` class:

This class will have the same members as the previous weapons classes but will now include an enumerated variable for the munition type called **`mtype`** that is defined globally as:

```
enum MunitionType {BOMB, ROCKET, MISSILE};
```

This should be included in the class header file and if needed in the cpp files that use this enum type. Note that we will use a new method to handle these weapon objects so the **`released`** boolean variable should be removed from the class.

Rewrite the constructor for this class to use this **`MunitionType`** as input then have it set the default parameters for bombs, rockets, or missiles based on this input. Be sure to set **`mtype`** to this value for each instance. Note: the new method for handling weapons will not require default values for **`x, y, z, vx, vy,`** `and` **`vz`** to be set in the constructor. These can be set to 0 if desired.

Each class should have the following variables stored for each weapon object:

1. Enumerated munition type: **`mtype`**
2. Integer location coordinates: **`x, y, z`**
3. Floating point* velocity vector components: **`vx, vy, vz`**
4. Floating point mass in kg of object: **`mass`**
5. Floating point thrust of the object: **`thrust`**
6. Floating point drag of the object: **`drag`**
7. Integer value for weapon effectiveness: **`hitpower`**

* Note new type for velocity parameters.

Default munitions parameters.

Weapon Type	`mass`	`drag`	`thrust`	`hitpower`
`bomb`	550.0 kg	0.08	0	1000
`rocket`	78.0 kg	0.02	1500.0 N	150
`missile`	245.0 kg	0.05	5200.0 N	560

The `jet` Class:

The **`jet`** class should have a constructor that has default values as well an an option to initialize all its parameters upon initialization. Note that this constructor will no longer need to initialize weapons location and velocity parameters as before. The new default `jet` parameters should be:

	x	y	z	vx	vy	vz	mass	drag	thrust
jet	1500	6800	9800	120.0	80.0	0.0	14000	0.28	200,000

This class will now include 4 STL vectors that store objects of class **`munition`** . Three will store munitions of each type as payload for the aircraft and one vector will store released munitions that will be updated along with the aircraft position by a new **`update()`** class function in this project.

The vector declarations will need the library: **`#include <vector>`**

```
vector<munition> bombs, rockets, missiles; // declare weapons storage
vectors
vector<munition> releasedweapons;          // vector to store weapons
released
```

`jet` Class Functions:

The jet class should include the class functions:

1. A function to check the range of another jet object. Same as before this returns a 1 if the jets are within the weapon range otherwise it returns a 0.

```
int inrange((jet *enemyjet, float weaponrange){  }
```

2. A function to load weapons using the enumerated munition type and a quantity as parameters.

```
void load(MunitionType mtype, int qty) {}
```

This function will allow up to 4 bomb, 4 missile, or 8 rocket objects to be inserted into the vectors defined above. Be sure to check the vector **`size()`** so that the function does not add more than the limits above for weapons. When called this function should increase the **`mass`** and **`drag`** of the jet by the amount of each rocket, missile, or bomb loaded.

3. A function to fire a number of selected weapons using the enumerated munition type and a quantity as parameters. Note that **`qty`** will be limited to max number available.

```
void fire(MunitionType mtype, int qty) {}
```

This function will use a for loop to:

- pull each munition object out of the appropriate storage vector
- set the object variables **`x, y, z, vx, vy,`** and **`vz`** to those of the jet object.
- subtract **`mass`** and **`drag`** values for each object fired from the jet variables.
- insert the object into the **`releasedweapons`** vector.

4. Write a function to update the position of the jet and any released weapons.

`void update() {}`

This function will use update **`x, y,`** and **`z`** using **`vx, vy,`** and **`vz`** for the jet and each released weapon using the technique of the in class space ship example: **`x += vx; y += vy; z += vz;`**

For released weapons we should account for gravity using: **`zy += -9.80;`** which is the acceleration due to gravity.
There are a number of possible additions you can make to the physics but we will omit them for this simple project.

When an object reaches a **`z <= 0`** it should be removed from the **`releasedweapons`** vector and display a message stating "weapon detonation" and print its **`x`** and **`y`** location.

5. Write a function for the `jet` class called: **`void dispstatus()`** that prints out the jet's 9 parameter values to the screen along with the numbers of stored weapons with each value labeled.

6. Write a function to display the status of the released weapons: **`void disprelweapons()`** which will include location and velocity parameters.

7. In the **`main()`** program body create 2 instances of the jet with jet1 at the default values and jet2 using these parameters.

	x	y	z	vx	vy	vz	mass	drag	thrust
jet2	2700	7600	9700	-120.0	-80.0	0.0	18000	0.48	200,000

Notice that the velocities and locations have the 2 jets approaching each other.

Then write some code to exercise the functions in the classes to make sure the status is reported correctly and that the jet and released weapons track correctly when updated.

Be sure to document your code with a comment header with the course number, assignment name, your name, date, and a basic description of the program. There should also be a comment header for each function with a description along with comments within the code body and particularly for the *if / else if* and *while* control statements.

Grading rubric: Documentation 20%, Code compiles without error 40%, Code functionality 40%

Submit your C++ source code to the Final Project Drop Box in the Brightspace - Learning Path / Week 11 module.