Final Project Description

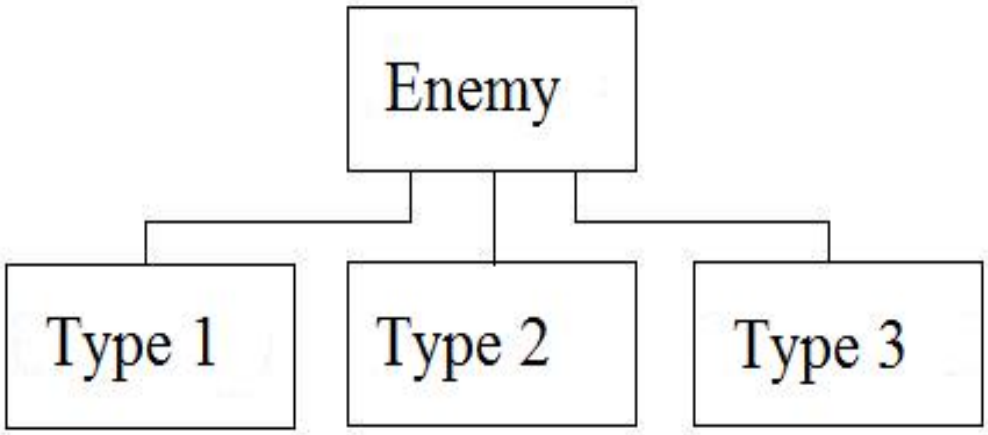
Overview:

Final Project is an application of the Object-Oriented approach using C++ and all the concepts that were covered in this course.

Description:

Create the classes and test all the functions as described below. Each class should have its own .h (specification) and .cpp (implementation) files, then test all the classes in the main.cpp file.

The chart below shows the classes relationships:



The assigned Types are as follow:

Type 1	Type 2	Type 3
PitBull	Jet	Soldier
Lion	Helicopter	Robot
Tiger	Drone	BadCop

The class Enemy:

The parent class Enemy is the super class to 3 sub classes Type1, Type 2 and Type 3.

- The class declare the variables: x_position, y_position, width, height and status, all as private.
- The class has getters and setters for all the variables
- The class will contain the methods: move_position, fire_weapon and update_status: These three methods are public and pure virtual

Main

The main method should be in its own .cpp file; It will create Enemy objects and store them in an array. The methods of an object define what the object DOES, the actions it will perform. Main method will loop and ask the Enemy objects to perform actions like move_position, fire_weapon and update_status. The basic layout will be:

```
int main()
{
    const int max_enemy = 20;
    Enemy* enemy_ptr[max_enemy];
    int num_enemy;

        // create Enemy objects, place in array
        // set value of num_enemy

    while ( true ) {

        // every Enemy object should move_position

        // Pick a random Enemy to fire_weapon

        // Pick a random Enemy to update_status

        cout << endl;
    }
    return 0;
}
```

You can google 'C++ random number' for help on generating a random number, then use the modulus '%' operator: **rand() % num_enemy** This will select which Enemy in the array should perform an action.
Keep main() simple! The objects will do the work, and print messages reporting what they have done. The ONLY enemy object methods you are allowed to call are move_position, fire_weapon and update_status

Type 1 - Type 2 - Type 3 Classes

Create the assigned classes and have each one print a simple message for each action it takes: These should be simple classes, a variable to track the name of the object and simple cout statements to report each action. Make sure move_position does NOT output an end of line, if there are 10 objects we DO NOT want it to print 10 lines of output. Do custom messages to match the Enemy, a pitbull will bite and a soldier will shoot a rifle when they fire a weapon. A soldier will say ouch and a Jet will make a ping sound when update status records a hit.
Make sure you can get the simple version of your program working. Test it with different numbers and different combinations of enemy objects. Each time through the main loop it should output messages like the example below:

```
PitBull moves position   Jet moves position   Soldier moves position
Jet fire weapon: missile
PitBull update status: I have been hit (bark)
```

Better Enemy Classes!

When you have the simple version working, improve your child classes. Make them behave like they would in a video game. Any change to main() or the Enemy parent class should be very minor (if at all), the focus is on the child classes. Make them move, and fire weapons and record hits when update status is called:

move_position

Each object exists in a 2D space, their X position can range from 0 to 800, y position 0 to 600. Position 0,0 is the top left corner, 800,600 is the bottom right. The ground is at 500 so a person or animal will be at y position 500. A low flying object at y position 300, a high flying one a 100. Objects move on the X axis, have walking objects move 3, running 6 each time move position is called. Flying objects move 15 to 30 with each call depending on their speed. Do not make them all start in the same position, do not make them move at the same speed and do not make them all move in the same direction. When an objects status is zero, it is DEAD, it should no longer move.

fire_weapon

Make sure each object fires an appropriate weapon. In general type 1 objects bite or slash. Type 2 objects have missiles or bombs. Type 3 have guns. Keep track of ammo, if a jet has 4 missiles, have fire weapon report out of ammo on the 5th call. Also check status, when dead, fire weapon should say NO WEAPON FIRED. Since animals do not use ammo, have them vary the attack, e.g. bite leg, slash chest, bite neck.

update_status

Update status means the Enemy object has been HIT. It is to lose status points and DIE if status reaches zero. Type 1 objects should take only 1 or 2 hits before they die, type 3 objects 4 to 5 hits and type 2 should take 7 to 8 hits. As always, make it match the enemy, a Robot will take more hits before death than a Car Jacker. Always report current status points when called, output an extra special statement when death occurs. A soldier might say ouch for a non-lethal hit, but

ARRRGH for a lethal one. The simple way to do this is to start each object with a status that matches the number of hits to kill it and subtract one every time.
Once you have your improved child classes, the output should update each time through the loop, the example below shows the program with 3 loops:

```
PitBull move to 710,500   Jet move to 320,100   Soldier move to 518,500
Jet fire weapon: missile (2 left)
PitBull update status: hit by bullet, status points  0 (dead)
PitBull move to 710,500   Jet move to 360,100   Soldier move to 514,500
PitBull fire weapon: dead!!!!!
Soldier update status: hit by bullet, status points  3 (ouch)
PitBull move to 710,500   Jet move to 400,100   Soldier move to 510,500
Soldier fire weapon: rifle (12 bullets left)
Jet update status: hit by bullet, status points  7 (ping)
```

Submission: Overall you should have 9 files to submit, submit all the files as a zip file (no rar files) to the Drop Box by the Due Date.