CS162 - Programming Assignment 6 - Object Pointer Practice

The purpose of this assignment is to give you some more practice using object pointers and dynamic memory allocation.

Overview

This program is part five of many that will implement an Asteroids-like game. Note: Asteroids is a registered trademark of Atari, Inc.

This assignment will build upon the previous assignment.

- First, you'll create a Photon class (photons are what the ship shoots at the asteroids, feel free to name it something else)
- Then you'll add code to enable the ship to fire photons
- And lastly you'll detect collisions between the photons and asteroids and split (optional) or destroy the asteroid.

When the assignment's complete, your ship will be able to shoot at and destroy the asteroids! Take that asteroids!

Part 1 - Bring on the Photons!

Add source files to your project to define and implement the Photon class. The class definition should look like this:

```
class Photon : public SpaceObject {
private:
      int TimeToLive; //counts down to 0
public:
      Photon();
      Photon(Vector loc, Vector vel, float ang);
      bool timeToDie();
      void draw(sf::RenderWindow& win);
};
```

Notice that in addition to the default constructor, we have one that accepts arguments; when photons are created, we need to provide this basic information (which is going to come from the ship).

The functions should do the following:

Photon()	Initialize everything to 0. We'll never really use this, but just in case….
Photon(…)	initialize base class variables as follows: ● radius should be something small (maybe .5 or 1) ● location should be set to given location (loc) ● velocity should be set to the given velocity (which will be the ships velocity) plus an additional (much greater) amount computed using the given angle (which is the direction the photon is fired). This will be computed much like the ships thrust was. Initialize derived class varialbes as follows ● timeToLive should be around 150 (this will be tweaked later based on experimentation). This will be used to make the photon disappear after awhile.
timeToDie	returns false if TimeToLive is <= 0.
draw	Subtracts 1 from TimeToLive. Draws photon on screen as a polygon of the given radius (it'll be very small)

In the file that contains your main() routine:

- Define a constant integer constant named MAX_PHOTONS and set it equal to 10;
- Near the top of main()
  - declare an array of pointers to Photons and initialize them all to NULL (0);
- In the event handler section
  - When the space bar is pressed (not when it's down, when it's pressed; this is an event, which is NOT how we're handling the arrow keys!)
    ```
    if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Space )
    ```
    Find the first NULL entry in the photons array (might want a function for that, as photons will come and go), add a pointer to a new Photon, which when created will be passed the ships position, velocity, and angle, to the photon pointer array.
    ```
    photon[x] = new Photon(ship.getLocation(), ship.getVelocity(), ship.getAngle());
    ```
- In main where you draw the ship and asteroids
  - Add code that goes through the photon array and calls draw() for each non-null entry.
  - You'll also probably need to call update-location so they move.

Your ship should now be able to fire photons, and have MAX_PHOTONS on the screen at a time. Test it out and debug it as necessary.

Part 2 - Blast-em!

Now we just need to detect when photons have hit an asteroid, and make the asteroid disappear (or split). Detecting the collisions is easy because we already have a function to detect when two SpaceObjects are intersecting. And to start, we'll just have the asteroids disappear when they're hit.

At this point, the "game loop" is getting pretty big. It should have the sections listed below, which I'd recommend commenting. Also, if you are doing multiple things (updating location, drawing, handling collisions) in the same area/loop I'd recommend splitting it up. Doing one thing at a time will result in easier to read and debug code. Especially since we're now deleting things from the arrays.

- handle events
- update object locations
- check for collisions (asteroid/ship, photon/asteroid)
- if ship is gone, redraw in middle (optional)
- Draw new frame (asterids, photons, ship; in that order…)

Add code to check each photon against each asteroid to see if they intersect. If an intersection is found, delete both the asteroid and photon and set their array entries to NULL. This takes less than a dozen lines of code. (two nested for loops, a couple of if's, two deletes and null assignments). Make sure you skip NULL array entries!

And now that our Asteroid array will contain null pointers you need to go back through your code and everywhere you process the Asteroid array skip over the null entries!!!

Part 3 - Making it better (optional)

When a big asteroid is hit, you could split it into two or four smaller asteroids (delete the original and add new ones to the array). And instead of just disappearing, the asteroid could explode.

main() is getting rather long, and combines both UI (User interface) code with the game logic. We really should have a Game class, which contains the ship, asteroid, and photon arrays as members, public functions that are called when certain keys are pressed (arrows, space bar, etc..), as well as functions that moved objects, detected collisions, and drew a new frame. main() would then create an instance of this object, and implement the loop that used SFML to detect events/keys and call the functions when those events/keypresses occurred. This would simplify the addition of new features, such as different levels, players, a splash screen, etc..

Very Important Stuff

All programs should follow the class's coding conventions.
Submit the following:

- A zip file containing all the source files you created and your executable