Name: Luna McBride
Title: Graphics Test
Project Summary: This will be an OpenGL scene using C++, implementing the topics that were done in my graphics summer class with Object Oriented principles.
Implemented Project Requirements:

Use Case	Requirement Description (User)	Class
5501	See the scene via a first-person camera	FirstPerson
5503	Turn left and right using the '←' and '→' keys respectively	FirstPerson
5504	Look up and down using the '↑' and '↓' keys respectively	FirstPerson
5505	Move Forward and back with the 'W' and 'S' keys respectively	FirstPerson
5302	Basic model classes (Sphere, Cube, etc)	Shapes
5303	Make big model classes using basic models	Trees, Mushrooms

Not Implemented Project Requirements:

Use Case	Requirement Description (System)
5301	Make scene with MVC
5502	A way to show controls so they are not on screen the whole time
5506	Bonk into objects instead of phasing through
5507	Include small animations/visual effects

Reasons for not using these:

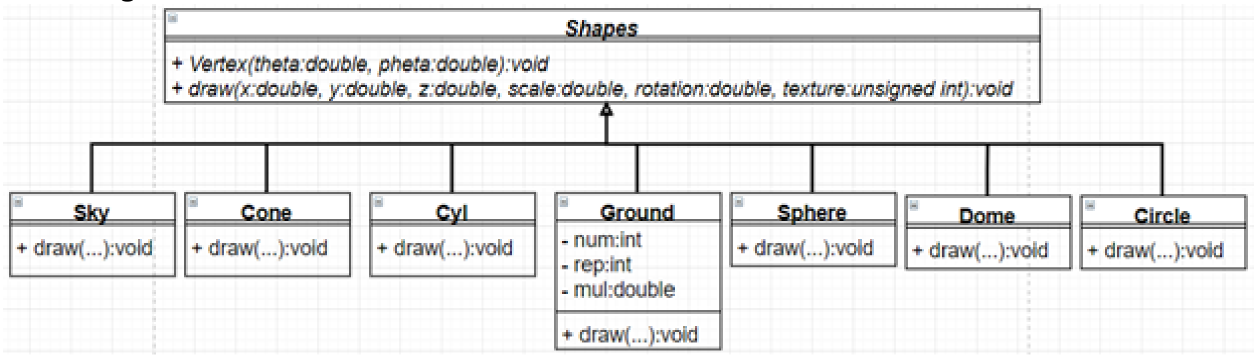
5301: I could not get an MVC to work with OpenGL/GLUT. The GLUT listener items (necessary for using OpenGL) must be in main.

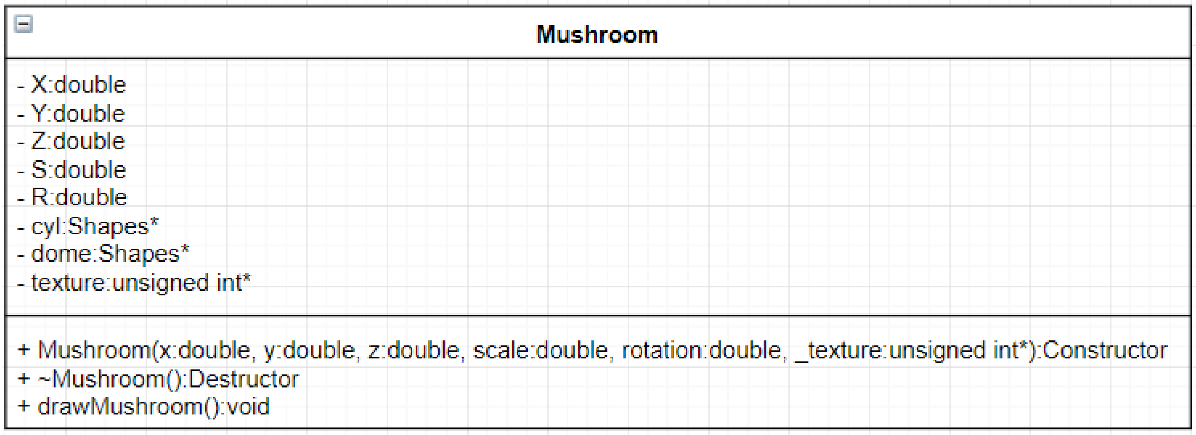
5502: The controls are very basic tank controls. They do not really need to be shown at all.

5506: In the class, we never learned physics. I decided to try and stick only to what we used there to have the graphics as a base to see how the Object-Oriented Principles were affecting the final product.

5507: What sorts of small animations would I put into this forest scene?

Class Diagram:

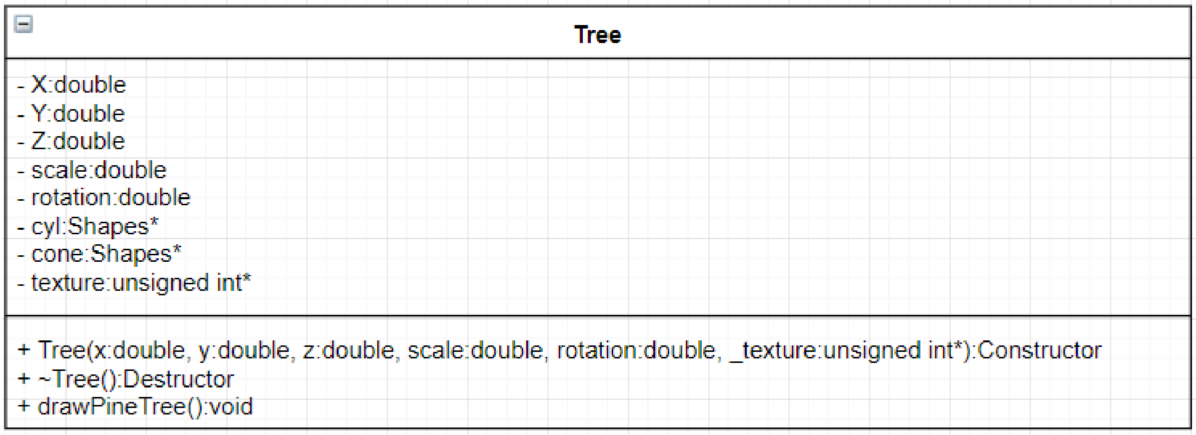




Mushroom

- X:double
- Y:double
- Z:double
- S:double
- R:double
- cyl:Shapes*
- dome:Shapes*
- texture:unsigned int*

+ Mushroom(x:double, y:double, z:double, scale:double, rotation:double, _texture:unsigned int*):Constructor
+ ~Mushroom():Destructor
+ drawMushroom():void

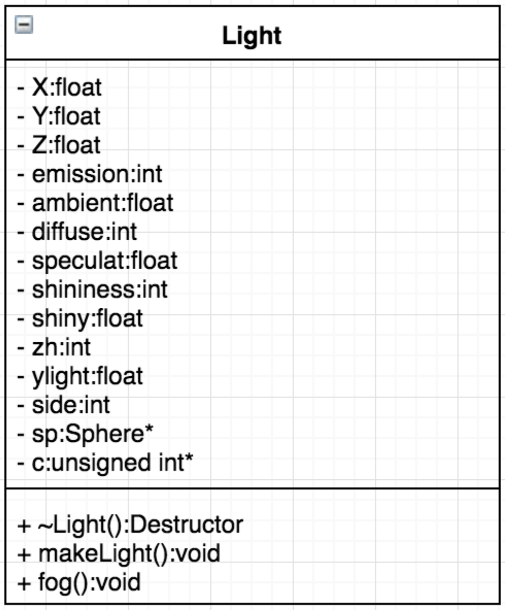


Tree

- X:double
- Y:double
- Z:double
- scale:double
- rotation:double
- cyl:Shapes*
- cone:Shapes*
- texture:unsigned int*

+ Tree(x:double, y:double, z:double, scale:double, rotation:double, _texture:unsigned int*):Constructor
+ ~Tree():Destructor
+ drawPineTree():void

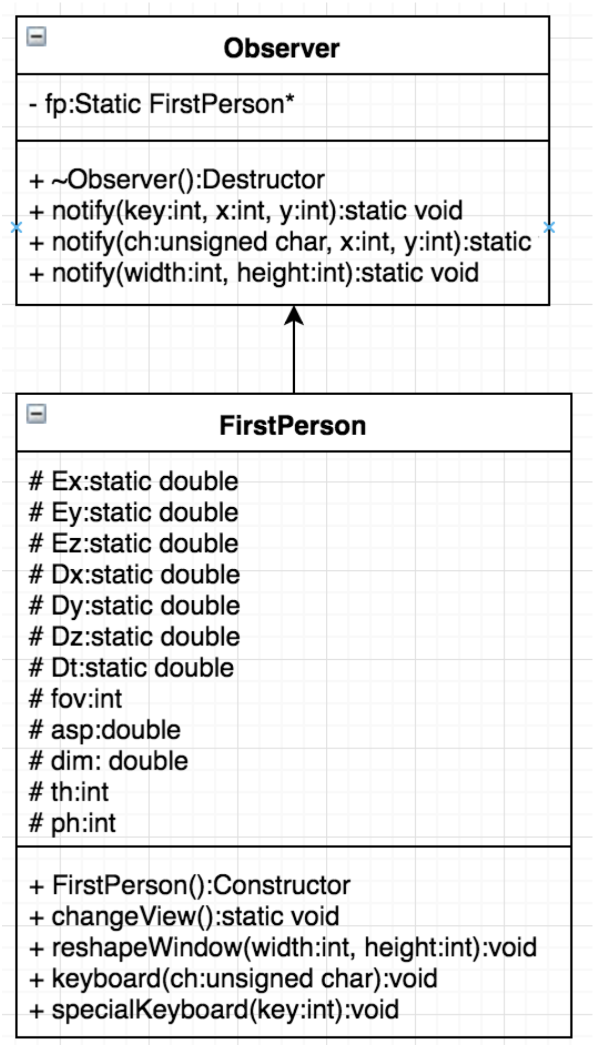
(Note: these have a non-inheritance/interface connection to the Shapes by use, but there is no way this could fit in this document)



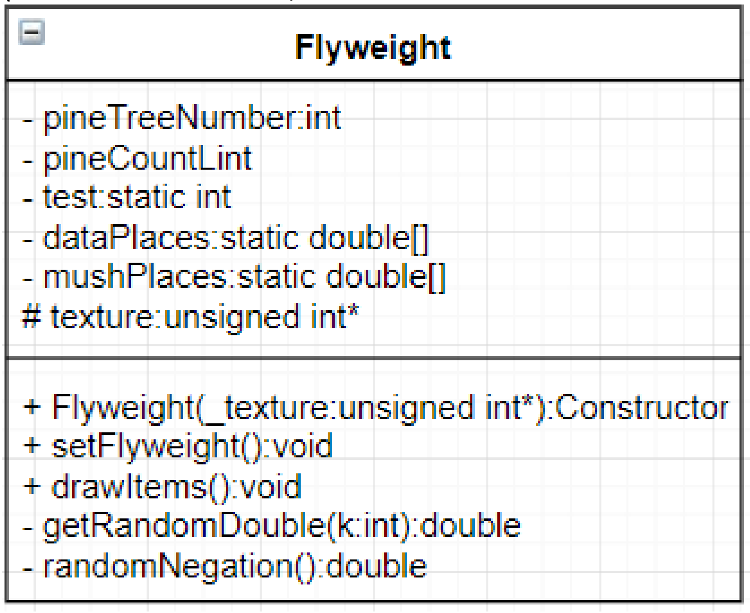
Light

- X:float
- Y:float
- Z:float
- emission:int
- ambient:float
- diffuse:int
- speculat:float
- shininess:int
- shiny:float
- zh:int
- ylight:float
- side:int
- sp:Sphere*
- c:unsigned int*

+ ~Light():Destructor
+ makeLight():void
+ fog():void

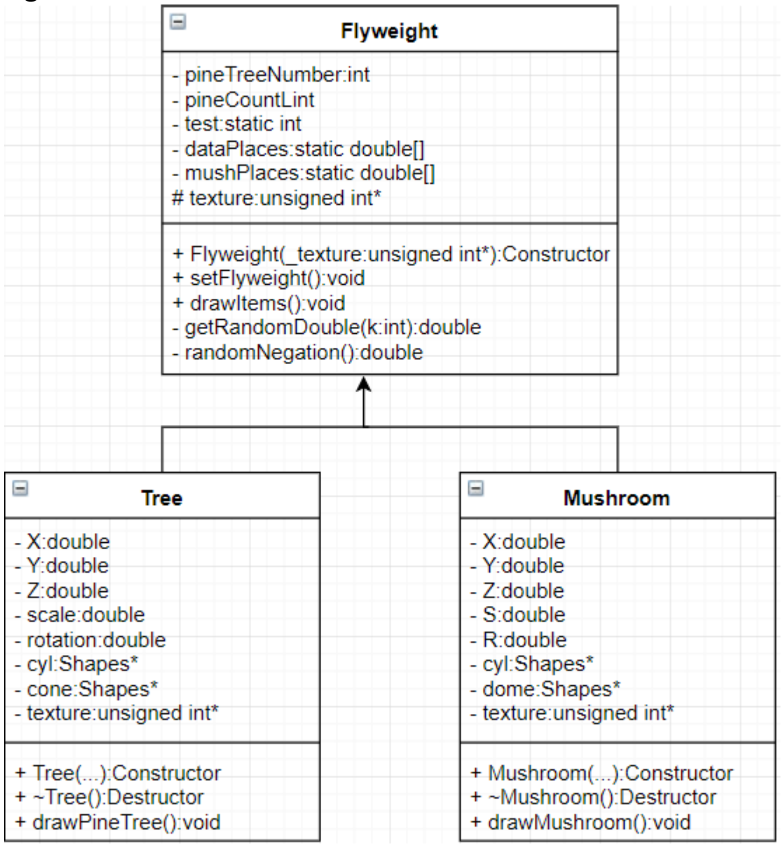

Observer
- fp:Static FirstPerson*
+ ~Observer():Destructor
+ notify(key:int, x:int, y:int):static void
+ notify(ch:unsigned char, x:int, y:int):static
+ notify(width:int, height:int):static void
FirstPerson
# Ex:static double
# Ey:static double
# Ez:static double
# Dx:static double
# Dy:static double
# Dz:static double
# Dt:static double
# fov:int
# asp:double
# dim: double
# th:int
# ph:int
+ FirstPerson():Constructor
+ changeView():static void
+ reshapeWindow(width:int, height:int):void
+ keyboard(ch:unsigned char):void
+ specialKeyboard(key:int):void

(There is a connection, but there is not inheritance or interface)


Flyweight
- pineTreeNumber:int
- pineCountLint
- test:static int
- dataPlaces:static double[]
- mushPlaces:static double[]
# texture:unsigned int*
+ Flyweight(_texture:unsigned int*):Constructor
+ setFlyweight():void
+ drawItems():void
- getRandomDouble(k:int):double
- randomNegation():double

Design Patterns:

-> Flyweight:

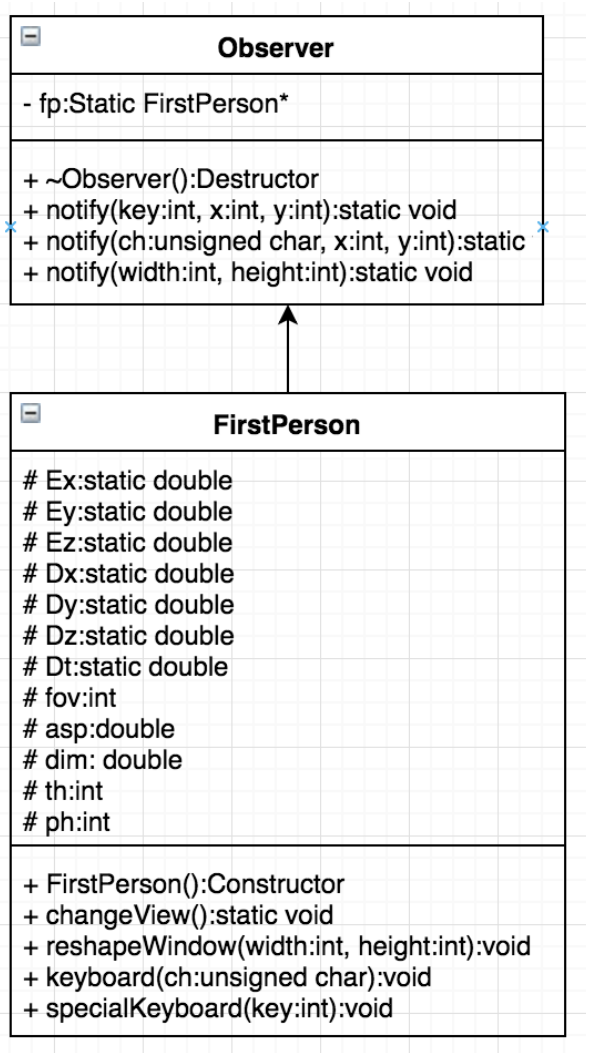


(Edited to make everything fit onto the screen legibly)

The Flyweight was used to randomize values and use those to draw the trees and mushrooms using those values. The Flyweight was chosen to calculate heavy calculations to make a more varied scene for this forest. These random values are calculated once, but stored in arrays, as OpenGL calls to display items for every frame, making the numbers need to be accessible without randomizing these values for every frame (which is why the test variable is included).

This is associated with the trees in mushrooms in that it draws them, but not doing any inheriting or anything of the sort as I do not want to give actual access to everything in each of those classes to the Flyweight or vice versa, as shown by the solid arrow in the class diagram

-> Observer


Observer
- fp:Static FirstPerson*
+ ~Observer():Destructor
+ notify(key:int, x:int, y:int):static void
+ notify(ch:unsigned char, x:int, y:int):static
+ notify(width:int, height:int):static void
FirstPerson
# Ex:static double
# Ey:static double
# Ez:static double
# Dx:static double
# Dy:static double
# Dz:static double
# Dt:static double
# fov:int
# asp:double
# dim: double
# th:int
# ph:int
+ FirstPerson():Constructor
+ changeView():static void
+ reshapeWindow(width:int, height:int):void
+ keyboard(ch:unsigned char):void
+ specialKeyboard(key:int):void

This one is an interesting case, as it was born out of necessity. The main function contains the GLUT listeners, which both had to stay in main and had peculiar behavior. They take in a function name and give their own parameters, which does not allow for lowering that number if you do not use some or putting more in when you need more. In addition, if you are calling a class method with these, they will get mad if it is not static, which can mess up the code inside the method if it wants to clash with being static.

This is where the Observer comes in. It has 3 static notifiers following the signatures for glutReshapeFunc, glutSpecialFunc, and glutKeyboardFunc, which are the listeners for the reshape function (reshaping the window), the special function (pressing keys besides the regular ones, like the arrow keys), and the keyboard function (pressing regular keys) respectively. The required values are then passed from the notifier to the first person, which moves using the set keys and also really is the window, so reshaping made sense for it to handle.

What I learned:

I used this project more to be able to see how these changes were affecting my code. I started to throw more and more at it to try and bog it down, but it never nearly got as slow as my project for the graphics class itself, despite possibly having more on screen. As I decided to use what was done in that class as a baseline, I can definitively say that the OOP I added was great for performance. Though I did not recreate the final project from graphics for originality and time purposes, I made sure to use the fog effect it used and about the same or more polygons than the project used to try to give each a fair chance. Even so, the graphics class final project is still so much slower than this came out to be on my computer.

In addition, working through limitations was an important thing to learn through this. I mentioned the listeners in the Observer section, which caused the observer to even be used to try to remove all the global variables associated with the functions they called. All worked well and good until it came to the fourth listener, being for the display function. Through trial and error, I found out that the textures would not load unless put specifically in the main function. It would give issue if tried in other functions. This meant I needed to get it to the display, which was a class at this point, from the main. This did not work either, as it would only give segmentation faults. It was forcing me to keep the display function in the main area and not a class. Not only that, since the listener would not let me pass the textures, it needed to, sadly, be kept as a global variable. That is just how limitations work, though. Sometimes you can find a way around it, but other times there is just no way.

The last thing I would say this taught was being clever with randomization. When I was first starting this and building the Flyweight, I had the issue of the randomizer only creating values in one area and that is it. This is reflected in the progress report II, where there are a bunch of cylinders piled in one place, but nowhere else. This was resolved by creating another randomizer to choose if the values will be positive or negative, which fixed the problem. Though there is still the possibility of trees randomizing within one another (as is the nature of randomization), this actually draws a forest scene quite well and makes it feel more akin to how trees actually grow and less man-made and calculated.

This was an interesting project to go through despite its hardships. I could see the effects of my actions easily in it and got to work with OpenGL's quirks, both good and bad.