CSci 1113, Spring 2018
Lab Exercise 12 (Week 13): Graphics

It's time to put all of your C++ knowledge to use to implement a substantial program. In this lab exercise you will construct a graphical game that employs many of the object-oriented techniques presented in class.

Warm-up

1) Getting started

Download and unzip the "Gar3D.zip" file. You will then have to use the terminal to goto this folder. To compile a program, use the following command:

```
make program FILE="<file names here>"
```

So to compile the "example.cpp" file, you would type:

```
make program FILE="example.cpp"
```

This will then create a program that you can run with the command (use up-arrows to get to previous commands instead of re-typing):

```
./gar3d-program.out
```

... or you can type (also try this if the above command doesn't work):

```
make run
```

NOTE: This code uses openGL and SDL. You might not be able to use this on your own computer.

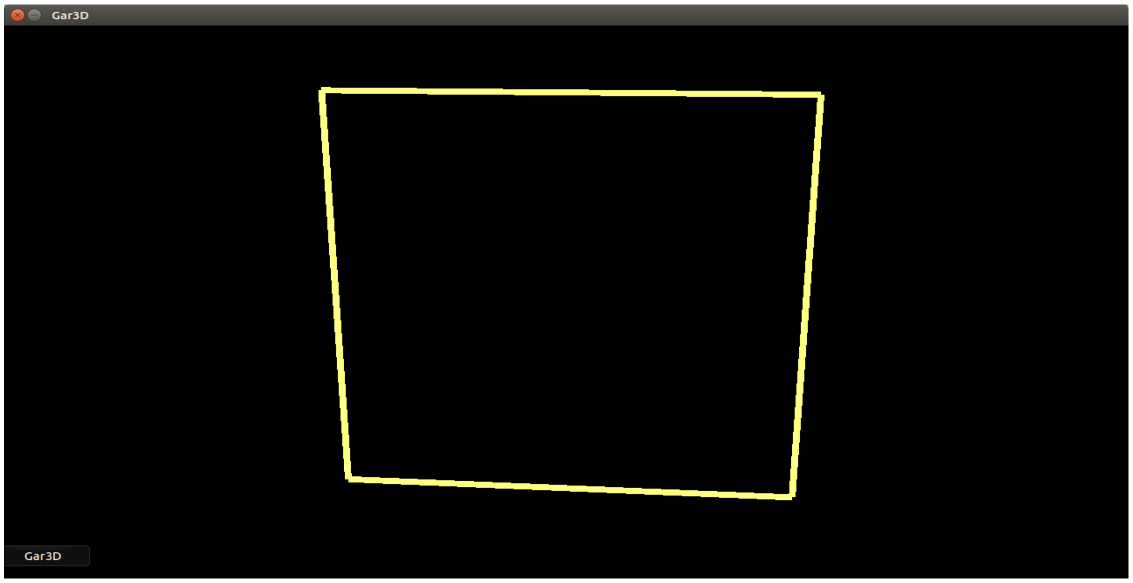
Use these commands to run the "example.cpp" file. You can press the "up arrow" key to switch scenes. (You can also drag with the mouse to move the camera.)

2) Drawing

Make your own program based off this sample file. Draw a simple rectangle using the drawLine() command. This command takes 6 inputs: (x0, y0, z0, x1, y1, z1). This rectangle should be colored white and extend from (-2,-2,0) to (2,2,0). If the line is too thin, you can use the lineSize() function to make it bigger.

See the appendix for a list of the more useful functions and how to use them.

At the end, you should get something that looks like this:



Stretch

1) Drawing shapes

Make two classes: one for the “game” and one for the “player”. The “game” class will hold general information about the game (the boundary square, the player, etc.). The “player” class holds information about the green symbol in the picture below (x and y position). These classes should be outlined as follows:

Player:
    Variables:
        - x and y position
    Functions:
        - default constructor (set to (0,0))
        - constructor with 2 inputs (double) to set xpos, ypos
        - draw() = makes some shape at (xpos,ypos,0). Again, you should use call-by-reference on the window object to pass it into draw. (Draw the shape size about 0.5)

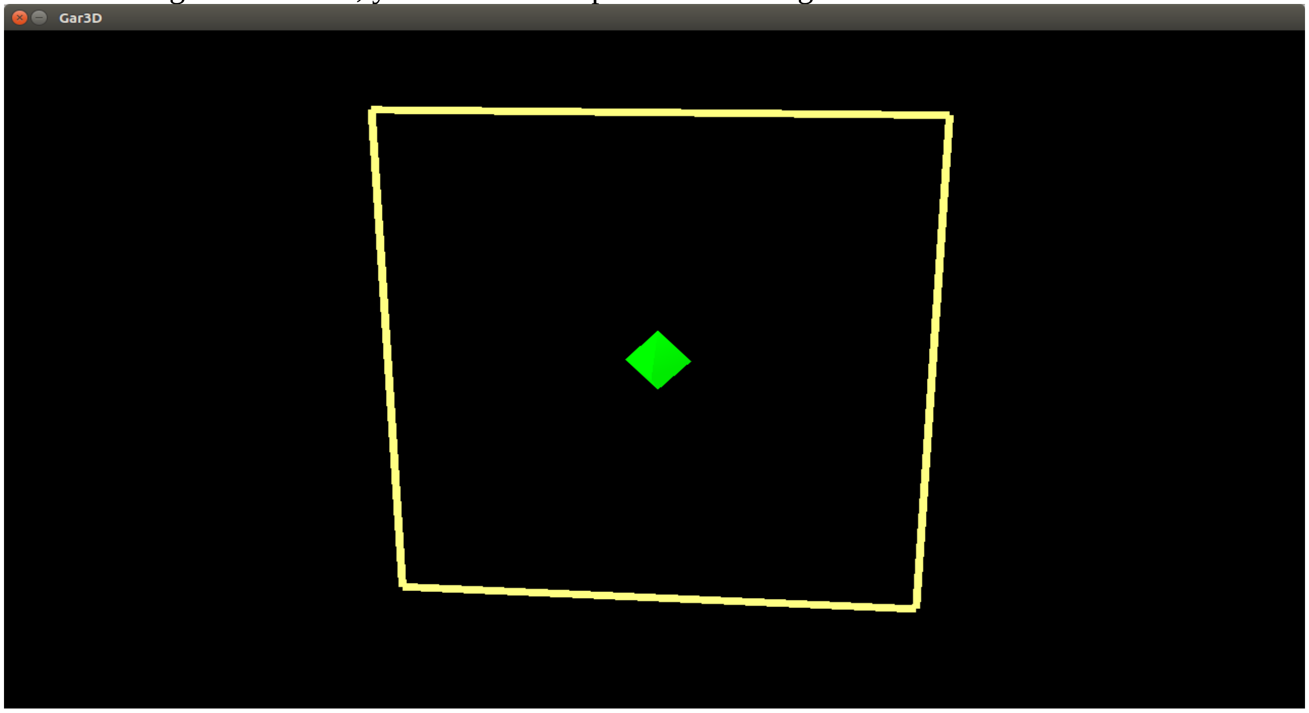
Game (this will be expanded in the next lab):
    Variables:
        - Player object/instance (see above)
        - square dimensions (top, bottom, left, right)
    Functions:
        - constructor = initialize player object
        - draw() = does two things: draws the square from warmup2 and calls player's draw. This should take as input a Gar3D object (by reference) (i.e. the window). Then main() should call this game's draw() function to draw the rectangle and player.

After applying all of these, your main should look like:

```
Gar3D window(1280, 720);

window.backgroundColor(black);
```

```
    window.lineSize(20);

    Game gameObj;
    while (!window.shouldQuit()) {
        gameObj.draw(window);

        window.renderFrame();
    }
```

After creating these classes, you should end up with something like:



Workout

1) Movement
Modify the player class to store an additional variable "speed" in addition to the "xpos" and "ypos". Either create a new constructor or update the existing one to take 3 arguments. Make the speed 0.5 for now.

Then modify the program to now take input from the arrow keys to move the player object in the corresponding direction. This input should happen instantly (as this is the easiest). To do this make a movement function in both Game and Player classes. In main(), run this function as this:

```
gameObj.movement(window.input());
```

The outline of this function would then be:

```
void Game::movement(const InputHandle& input)
```

Reference the example.cpp on how to handle input. Game's movement() function should then call player's movement() function (much the same way game's draw() chains to player's draw()).

2) Restricting movement
In workout 1, you should be able to move the player object wherever you want (even outside the square). Fix this so if you try to move left, but are already at the left-side of the square you simply go nowhere. (This applies to all four sides of the square). Assume you can overlap the boarder of the square, but not exist completely outside.

Make a function to do this either in Game or Player (you will either need to pass in the sides of the square to player, or make a function so game can change player's position). Call this new function inside the movement() function (after processing where to go).

3) Stopping
Finally, make the program stop when the player reaches the top left part of the screen. At this point the window should close and the program should stop.

<u>We will be building off this code next week, so please make sure you save a copy of the code.</u>

Challenge

1) Pretty picture
Modify the player's draw() function so that it creates some sort of animal (human, cat, dog, snail, etc.) rather than just a boring colored triangle or sphere. The pre-built shapes are:
sphere, box, cylinder, cone, pyramid_3 , pyramid_4, pyramid_6, plane, arrow, line

Or if you want to get down to the details, you can see how the render____() functions in Gar3Dclass.cpp are used. Regardless of which way you draw your animal, try to not make it too big (smaller than the amount it moves).

1) Animation
Have your animal from challenge 1 move its legs (or however it moves). Movement should still happen only when the arrow key is pressed (and this can still happen instantly). While you wait for the next input, make the legs wiggle, as if your player was walking.

<u>Appendix</u>

In the “Gar3D” class:

drawLine(x0, y0, z0, x1, y1, z1)	Draws a line from (x0, y0, z0) to (x1, y1, z1).
drawPrimitive(primitive, orientation, posx, posy, posz, sizex, sizey, sizez)	Draws a primitive object. Available primitives and orientations are shown in Gar3Dclass.hpp.
drawPrimitive(primitive, orientation, posx, posy, posz, size)	Same as above, but scales by a single size
renderFrame()	Renders drawn objects and updates the input handle. If this function is not called each frame, the program will not respond.

shouldQuit()	Returns true when the program should exit. This occurs when the user presses the escape key.
color(r, g, b)	Sets the color of the objects drawn. May be changed for each object. r, g and b are between 0 and 1 for “red”, “green” and “blue”.
input()	Gets the input handle. Methods for the input handle are described below in “InputHandle”.
lineSize(size)	Sets the thickness of lines .
backgroundColor(r, g, b) Sets the background color of the window. Value is only used when renderFrame() is called.	Sets the background color of the window. Value is only used when renderFrame() is called.
backgroundColor(color)	Same as above, but with a color type rather than 3 int types
deltaTime()	Return the amount of time (in seconds) elapsed during the previous frame (last renderFrame() call).

In the “InputHandle” class :

buttonPressed(Key)	Returns true on the first frame the button is pressed .
buttonHeld(Key)	Return true on every frame the button is held down .
buttonReleased(Key)	Returns true on the frame the key was released .
mousePosition(x, y)	Changes x and y variables to the mouse cursor's x and y position in the window.