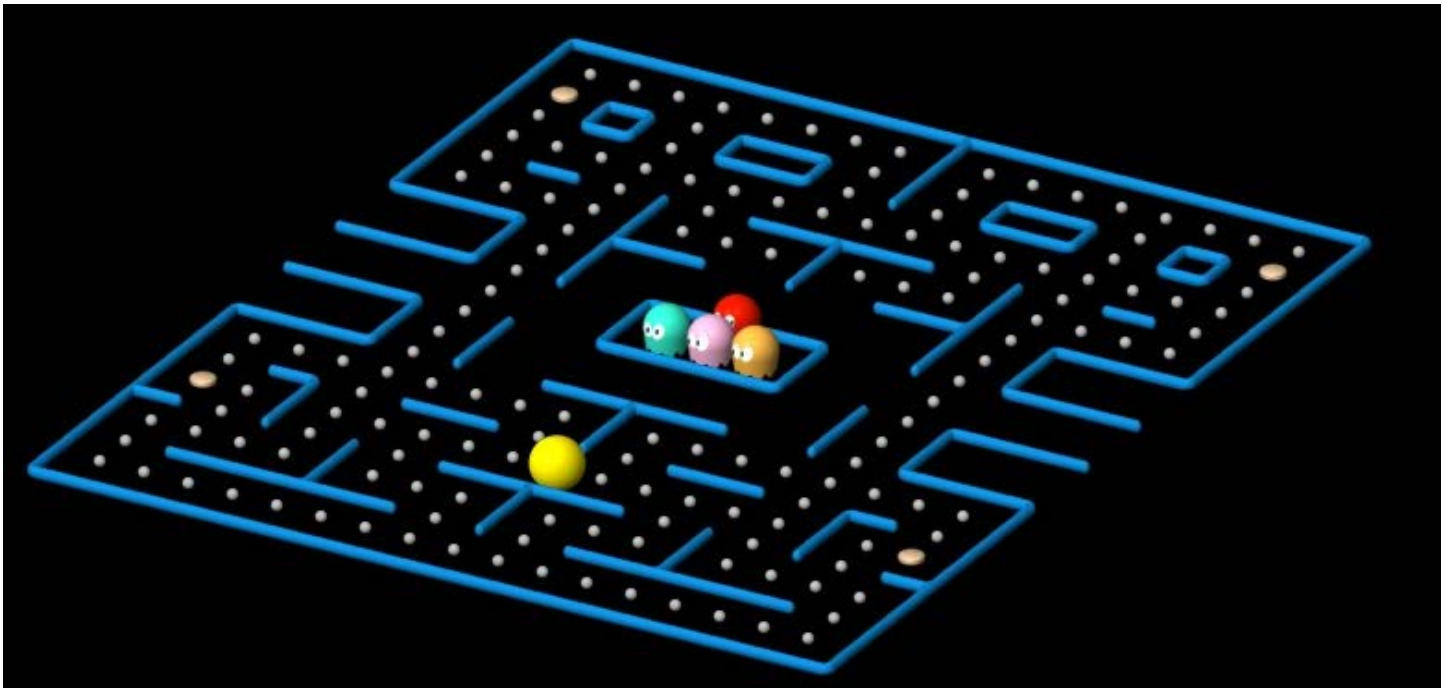
Lab4
3D Pac-Man Maze

(100 pts)

Section	Due Date
89313 - ECE 4122 - A	Oct 27th, 2020 by 11:59 PM
89314 - ECE 6122 - A	Oct 29th, 2020 by 11:59 PM
89340 - ECE 6122 – Q, QSZ, Q3	Oct 29th, 2020 by 11:59 PM

www.freewebarcade.com/game/3d-pacman/



Instructions:

1. For this assignment you need to develop software that recreates the pac-man maze shown in the image above using OpenGL and C++.
2. Your code must implement the following:
    a. Custom classes:
        i. **ECE_Ghost** class for drawing the four ghosts
            1. Ghost can be made with a sphere for a head and cylinder for body.
            2. Each ghost needs to have a unique color
                a. Red, green, orange, pink
            3. The ghosts are static and do not move.
        ii. **ECE_Pacman** class for drawing pac-man

1. Pac-man can be drawn as a yellow sphere
    a. Pac-man is static and does not move.

b. Maze
    i. You can use thin blue cylinders to form the walls of the maze.
    ii. You can scale the maze and objects however is easiest for you to draw and manage movements.

c. Coins are small silver spheres.

d. Power ups are golden flat discs.

e. Camera location should be similar to the location shown in image above. Object's should have specular reflections to enhance 3D appearance. Pressing the "R" key should rotate the maze clockwise by 5 degrees each time the key is pressed.

Grading Rubric

AUTOMATIC GRADING POINT DEDUCTIONS PER PROBLEM:

Element	Percentage Deduction	Details
Maze walls, coins, and power up are drawn correctly	Up to 30%	10% walls, 10% for coins, 10% for power ups.
Pace-man and ghost's location and shape in maze	Up to 30%	Pace-man need to be drawn as a yellow sphere. The Ghost need to be drawn as a sphere on top of a cylinder with the different colors.
Camera location, motion, and shading	Up to 30 %	Camera location should be similar to the location shown in image at start of assignment. Object's should have specular reflections to enhance 3D appearance.
Clear Self-Documenting Coding Styles	10%-25%	This can include incorrect indentation, using unclear variable names, unclear/missing comments, or compiling with warnings. (See Appendix A)

Extra Credit: (Grader's discretion up to 15% total)

Element	Percentage Extra Credit	Details
Texture wrap on pac-man and ghosts	Up to 5%	Create static texture wraps for pac-man and ghosts.
Dynamic texture wrap on pac-man and ghosts	Up to 10%	Create multiple texture maps and cycle through them making game seem more realistic.
Use obj files for pac-man and ghosts	Up to 15%	Use some online obj files for the objects.

LATE POLICY

Element	Percentage Deduction	Details
Late Deduction Function	score - (20/24)*H	H = number of hours (ceiling function) passed deadline note : Sat/Sun count as one day; therefore H = 0.5*$H_{weekend}$

Appendix A: Coding Standards

Indentation:

When using *if/for/while* statements, make sure you indent 4 spaces for the content inside those. Also make sure that you use spaces to make the code more readable.
For example:

```
for (int i; i < 10; i++)
{
    j = j + i;
}
```

If you have nested statements, you should use multiple indentions. Each { should be on its own line (like the *for* loop) If you have *else* or *else if* statements after your *if* statement, they should be on their own line.

```
for (int i; i < 10; i++)
{
   if (i < 5)
   {
       counter++;
       k -= i;
   }
   else
   {
       k +=1;
   }
   j += i;
}
```

Camel Case:

This naming convention has the first letter of the variable be lower case, and the first letter in each new word be capitalized (e.g. firstSecondThird). This applies for functions and member functions as well! The main exception to this is class names, where the first letter should also be capitalized.

Variable and Function Names:

Your variable and function names should be clear about what that variable or function is. Do not use one letter variables, but use abbreviations when it is appropriate (for example: “imag" instead of
“imaginary”). The more descriptive your variable and function names are, the more readable your code will be. This is the idea behind self-documenting code.

File Headers:

Every file should have the following header at the top

```
/*
Author: <your name>
Class: ECE4122 or ECE6122
Last Date Modified: <date>

Description:

What is the purpose of this file?

*/
```

Code Comments:

1. Every function must have a comment section describing the purpose of the function, the input and output parameters, the return value (if any).
2. Every class must have a comment section to describe the purpose of the class.
3. Comments need to be placed inside of functions/loops to assist in the understanding of the flow of the code.

Appendix B: Accessing PACE-ICE Instructions

ACCESSING LINUX PACE-ICE CLUSTER (SERVER)

To access the PACE-ICE cluster you need certain software on your laptop or desktop system, as described below.

Windows Users:

Option 0 (Using SecureCRT)- THIS IS THE EASIEST OPTION!

The Georgia Tech Office of Information Technology (*OIT)* maintains a web page of software that can be downloaded and installed by students and faculty. That web page is:

http://software.oit.gatech.edu

From that page you will need to install SecureCRT.

To access this software, you will first have to log in with your Georgia Tech user name and password, then answer a series of questions regarding export controls.

Connecting using SecureCRT should be easy.

- Open SecureCRT, you'll be presented with the "Quick Connect" screen.
- Choose protocol "ssh2".
- Enter the name of the PACE machine you wish to connect to in the "HostName" box (i.e. *pace-ice.pace.gatech.edu*)
- Type your username in the "Username" box.
- Click "Connect".
- A new window will open, and you'll be prompted for your password.

Option 1 (Using Ubuntu for Windows 10):

Option 1 uses the Ubuntu on Windows program. This can only be downloaded if you are running Windows 10 or above. If using Windows 8 or below, use Options 2 or 3. It also requires the use of simple bash commands.

1. Install Ubuntu for Windows 10 by following the guide from the following link:

   https://msdn.microsoft.com/en-us/commandline/wsl/install-win10

2. Once Ubuntu for Windows 10 is installed, open it and type the following into the command line:

   *ssh **YourGTUsername**@pace-ice.pace.gatech.edu* where **YourGTUsername** is replaced with

   your alphanumeric GT login. Ex: bkim334

3. When it asks if you're sure you want to connect, type in:
   yes

   and type in your password when prompted (Note: When typing in your password, it will not show any characters typing)

4. You're now connected to PACE-ICE. You can edit files using vim by typing in:

vi filename.cc OR *nano vilename.cpp*

For a list of vim commands, use the following link:

https://coderwall.com/p/adv71w/basic-vim-commands-for-getting-started

5. You're able to edit, compile, run, and submit your code from this command line.

Option 2 (Using PuTTY):

Option 2 uses a program called PuTTY to ssh into the PACE-ICE cluster. It is easier to set up, but doesn't allow you to access any local files from the command line. It also requires the use of simple bash commands.

1. Download and install PuTTY from the following link: www.putty.org

2. Once installed, open PuTTY and for the Host Name, type in:
   pace-ice.pace.gatech.edu and for the port,

   leave it as 22.

3. Click Open and a window will pop up asking if you trust the host. Click Yes and it will then ask you for your username and password. (Note: When typing in your password, it will not show any characters typing)

4. You're now connected to PACE-ICE. You can edit files using vim by typing in: *vim filename.cc* OR
   nano vilename.cpp

   For a list of vim commands, use the following link:

   https://coderwall.com/p/adv71w/basic-vim-commands-for-getting-started

5. You're able to edit, compile, run, and submit your code from this command line.

MacOS Users:.

Option 0 (Using the Terminal to SSH into PACE-ICE):

This option uses the built-in terminal in MacOS to ssh into PACE-ICE and use a command line text editor to edit your code.

1. Open Terminal from the Launchpad or Spotlight Search.

2. Once you're in the terminal, ssh into PACE-ICE by typing:

*ssh **YourGTUsername**@ pace-ice.pace.gatech.edu* where **YourGTUsername** is replaced with your alphanumeric GT login. Ex: bkim334

3. When it asks if you're sure you want to connect, type in:
   yes

   and type in your password when prompted (Note: When typing in your password, it will not show any characters typing)

4. You're now connected to PACE-ICE. You can edit files using vim by typing in:

   vi filename.cc OR *nano filename.cpp*

   For a list of vim commands, use the following link: https://coderwall.com/p/adv71w/basic-vim-commands-for-getting-started

5. You're able to edit, compile, run, and submit your code from this command line.

Linux Users:

If you're using Linux, follow Option 0 for MacOS users.