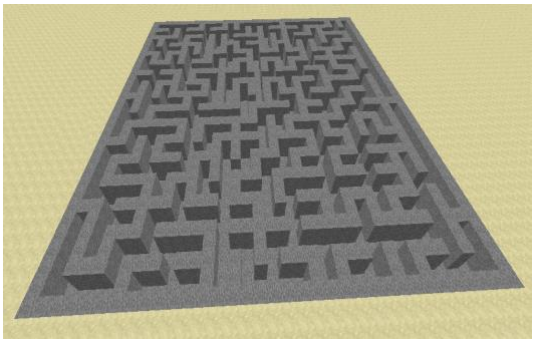
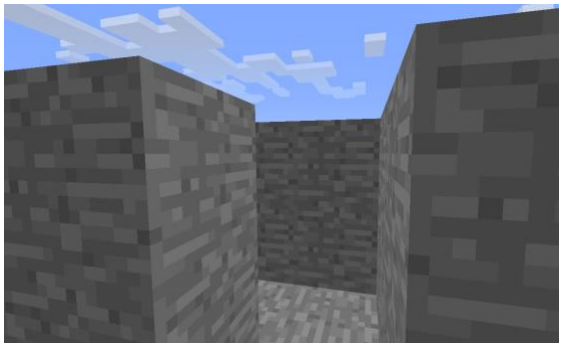
CSCE 4813 – Programming Project 5
Due Date – 04/15/2019 at 11:59pm
REVISED INSTRUCTIONS

1. Problem Statement:

The goal of this project is to create a video game called "MazeWorld" that allows the player to interactively explore a previously defined maze. To make the game look realistic, we will be using texture mapping when displaying the walls of the maze.

For this project, you must complete the following tasks:

1) Read the maze from an ASCII input file. The first line of the input file will contain two integers R (the number of rows in the maze) and C (the number of columns in the maze). The second line of the input file will also contain two integers SR (the start row) and SC (the start column). After the two header lines, there will be R rows of C characters that indicate where the walls in the maze are and what they are made of. We will use the following key: 'r'=rock, 'b'=brick, 'w'=wood, and ' '=empty. You should read these characters into a 2D array called "maze".

2) Read the texture images. The textures for the walls will be provided in the images "rock.jpg", "brick.jpg", and "wood.jpg". For the empty space texture, you can use either "grass.jpg" or "gravel.jpg". These four texture images should be resized so their dimensions are powers of two and stored in four OpenGL texture arrays. See sample texture mapping code on the class website for details.

3) Display the maze. Use the display callback to draw a large texture mapped rectangle representing the empty space in the maze. Then, loop over the maze array, and draw textured cubes at the corresponding maze locations. Use the character stored in maze[r][c] to determine which texture map should be used for each cube. You should adjust the size, shape of the "cubes" so there are no gaps in the wall, and they are tall enough to block the player's view of the rest of the maze.

4) Implement player navigation. Use keyboard callback function to control the motions of the "player" moving through your maze. One option would be to have the user type "f" to move forwards, "b" to move backwards, "l" to rotate left by 90 degrees, and "r" to rotate right by 90 degrees. Another option would be to use the familiar "w", "a", "s", "d" characters. You should display a small solid colored cube at the player's location so you can see where they are in the maze. Your program should always start the player at a specific (r,c) that corresponds to the start of the maze, and your program should use the maze array to prevent the player from walking through walls.

5) BONUS 10 POINTS (Required for graduate students).
Implement camera motion. When the program starts, you should position the camera so it is above the maze. This will give you a "top view" of the maze like the image on the left above. Use the keyboard callback function to control the movements of the camera. When they user enters "+" the camera should move one step towards the current player location. When they get right behind the player, you should see a "players view" that is similar to the image on the right above. When the user enters "-" the camera position should move one step back towards the starting position of the camera.

2. Design:

Your first design task is to select appropriate data structures to store the maze, the textures for the maze, the user's position and orientation, the camera's position and orientation. Then you need to design the code to read the maze and textures from input files.

Your second design task is to work out how user interactions will work. For example, how to implement player motion when the user types "f", "b", "l", "r" (or "w", "a", "s", "d") how to prevent the user from walking through walls, how far the camera moves when the user types "+" or "-", and how to keep the camera slightly behind the player as they explore the maze.

Finally, you need to decompose the tasks above into a number of functions that can be called from within the display and keyboard callbacks to create images of the maze as the player explores it.

3. Implementation:

This semester we will be using C++ and OpenGL to implement all of our programming projects. The instructions for downloading and installing a Linux VM and installing OpenGL are posted in README file the "Source Code" page of the class website.

For this project, you will need to use my libim library and the standard jpeg library to read and resize the texture images. To compile these libraries, you need to go to the src folder, type "cd libim; make clean; make; cd ../jpeg; make clean; make; cd .." Once you have these libraries installed, you can compile your graphics program using "g++ -Wall project.cpp -o project -lGL -lGLU -lglut libim/libim.a jpeg/libjpeg.a".

You are encouraged to look at sample OpenGL programs to see how the "main" function and the "display" function are normally implemented. As always, you should break the code into appropriate functions, and then add code incrementally writing comments, adding code, compiling, debugging, a little bit at a time.

Remember to use good programming style when creating your program. Choose good names for variables and constants, use proper indenting for loops and conditionals, and include clear comments in your code. Also, be sure to save backup copies of your program somewhere safe. Otherwise, you may end up retyping your whole program if something goes wrong.

4. Testing:

Test your program with different user inputs until you get some images that look fun/interesting. Take a screen shot of these images to include in your project report. You may also want to show some bad/ugly images that illustrate what happens if there is a problem somewhere. You can discuss how you corrected these problems in your project report.

5. Documentation:

When you have completed your C++ program, write a short report using the project report template describing what the objectives were, what you did, and the status of the program. Be sure to include several output images. Finally, describe any known problems and/or your ideas on how to improve your program. Save this report to be submitted electronically via Blackboard.

6. Project Submission:

In this class, we will be using electronic project submission to make sure that all students hand their programming projects and labs on time, and to perform automatic plagiarism analysis of all programs that are submitted. Please upload your project as follows:

1) If your code was developed on a Mac or Linux machine, please make sure you have the original #ifdef MAC lines at the top of your program so we can compile and run your code on either a Mac or Linux machine. If your code was developed on a PC, please edit your code so it can also be compiled on Linux. This can be easily done by adding something like this to your program.

```
#define LINIX
#ifdef LINUX
#include <GL/glut.h>
#else
#include <YOUR_OPENGL_INCLUDE.h>
#endif
```

When you want to compile on your PC, comment out the #define LINUX line. When you want to compile on turing, uncomment this line. The OpenGL libraries on turing are all installed in the normal way, so your code should compile. If you have X forwarding enabled, you can run your program on turing and display images on your PC.

2) Please copy your code over to turing, and create a small Makefile to compile your program. Something like this will do the trick (remember to use tab to indent the compile line).

```
# define libraries
LIB = -lGL -lGLU -lglut

# define g++ flags
CC = g++ -Wall -Wno-deprecated-declarations -O3

project: project.cpp
        $(CC) -o project project.cpp $(LIB)
```

3) When you get ready to upload into blackboard, please create a tar/zip file that contains your code, your makefile and report. This way we can download and untar/unzip your code, type "make" and then test it.

The dates on your electronic submission will be used to verify that you met the due date above. All late projects will receive reduced credit:

- 10% off if less than 1 day late,
- 20% off if less than 2 days late,
- 30% off if less than 3 days late,
- no credit if more than 3 days late.

You will receive partial credit for all programs that compile even if they do not meet all program requirements, so handing projects in on time is highly recommended.

7. Academic Honesty Statement:

Students are expected to submit their own work on all programming projects, unless group projects have been explicitly assigned. Students are NOT allowed to distribute code to each other, or copy code from another individual or website. Students ARE allowed to use any materials on the class website, or in the textbook, or ask the instructor and/or GTAs for assistance.

This course will be using highly effective program comparison software to calculate the similarity of all programs to each other, and to homework assignments from previous semesters. Please do not be tempted to plagiarize from another student.

Violations of the policies above will be reported to the Provost's office and may result in a ZERO on the programming project, an F in the class, or suspension from the university, depending on the severity of the violation and any history of prior violations.