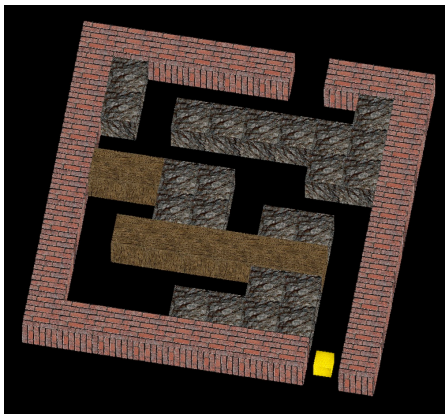
CSCE 4813 – Computer Graphics
Programming Project 4
Due Date – 4/10/2024 at 11:59pm

1. Problem Statement:

The goal of this two-part project is to create a retro video game where the player explores a 2D maze to find and collect hidden treasure. In this project, the focus is on creating and displaying the 2D maze made of texture mapped cubes. In the next project, you will add treasures at random locations in the maze and implement player navigation to collect these treasures while exploring the maze. Detailed instructions for this project are listed below.

```
10 10
0 8
bbbbbbbb b
b   rrrr b
b     rr b
b wwwwww b
b  rr rr b
bwwrr    b
b      rrb
br rrrrrb
br      rb
bbbbbb bbb
```



Sample maze file and corresponding maze image

Task 1: Read the maze file

You should start by reading and storing the maze. The first line of the "maze.txt" 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 contain the starting location of the player in the maze: 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".

Task 2: Read the texture images

The next step is to read in several images to initialize OpenGL textures. Textures for the walls of the maze will be in the images "rock.jpg", "brick.jpg", and "wood.jpg". The floor of the maze can be either "grass.jpg" or "gravel.jpg". All of these images have been resized so they are 512x512. You need to copy the RGB pixel values from these jpg images into corresponding OpenGL texture arrays.

Task 3: Display the maze

You need to display the maze as a collection of texture mapped cubes laying on top of a flat background. Start by drawing a large texture mapped rectangle for the floor in the maze (grass or gravel). Then, loop over the maze array, and draw texture mapped cubes at the corresponding maze locations. Use the character stored in maze[r][c] to determine which texture map should be used for the wall cubes and where treasures should be drawn. You should adjust the size, shape of the wall cubes so there are no gaps in the walls.

Task 4: Rotate the maze

In order to see the maze from different viewpoints, you should add keyboard-based rotations to your program. When the user types 'x' or 'X you should decrease or increase the x rotation angle. Similarly, when they type 'y', 'Y', 'z' or 'Z' you should modify the y and z rotation angles. Hopefully, you will be able to generate maze images like the example above.

2. Design:

Your first design task is to select appropriate data structures to store the maze, the textures for the maze, and the rotation angles. 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 to display texture mapped cubes in the correct locations on the screen. The maze should take up about 75% of the display window, and the cubes should be scaled accordingly.

As always, 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. Once you have OpenGL installed, you can compile your graphics program using "g++ -Wall hw4.cpp -o hw4 -lGL -lGLU -lglut".

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 random number generator seeds 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 (e.g. too many lines, lines too short, etc.). 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. When you have completed the tasks above go to Blackboard to upload your documentation (a single docx or pdf file), and all of your C++ program files. Do NOT upload an executable version of your program.

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.