Due: 11/28 (11:59PM)

Requirements:

- Write an OpenGL program that generates a simple 3D maze game. Name your source code `hw3.cpp`. The program should meet the following requirements:
- Modeling
  - Construct a 3D maze on an at least 11×11 grid of cells.
  - The maze should have one or two exits.
  - Walls may be constructed as rectangles in 3D, or as flattened cubes.
  - Walls are vertically placed on a big ground plane (which is a big square, bigger than the entire maze area). The ground plane is parallel to the *zx*-plane.
  - Right above the walls there should be another big horizontal plane, which represents ceiling or sky (See Fig. 1).
  - The maze must be complex enough so that it's challenging to find the exit.
  - Use `http://www.mazegenerator.net` to get an idea on how to construct a challenging maze.
- Texturing
  - All the walls, ground plane, and ceiling should be textured with distinct texture images (See Fig. 1).
  - Make sure to use *tileable* texture images in order to generate seamlessly textured objects. There are plenty of tileable texture images online.
  - For handling *.bmp* texture images, you may use *RGBpixmap.h* posted on Canvas.
- Camera Control
  - Camera moves as in *First Person Shooting (FPS)* game, navigating the maze from user's point of view.
  - When the program starts, the camera (user) must be at the center of the maze. The vertical height (*y*-coordinate) of the camera is at the middle-height of the wall, and is fixed throughout.
  - Camera is controlled by four arrow keys, UP (move forward), DOWN (move backward), RIGHT (right turn), LEFT (left turn). While the key is down, the camera should keep moving or rotating in the same direction.

  - Note that only horizontal movement of camera (*slide forward/backward* and *yaw*) is allowed.
  - For implementing Camera control, you may use *camera.h* posted on Canvas.

- Collision Detection
  - Keep the camera from penetrating walls while moving forward or backward. That is, the camera should be stopped by the system right before it touches any wall.
  - When such a camera-wall collision occurs during forward sliding (UP key), the system should let the camera continue to move *along* the wall while UP key is down. This way the user wouldn't have to back up every time the camera hits a wall.
  - If the camera is completely stuck at the corner of two intersecting walls, the user must either press DOWN key to back up, or press RIGHT/LEFT key to change the camera orientation and then move forward again.

- Floating/Spinning Objects
  - Generate several solid objects (for example, Teapot) here and there within the maze, each with a distinct color. They serve as marks to help user find the exit more easily.
  - Each of these objects must be slowly spinning on an arbitrary axis at a constant speed. The rotation is done *in place*, that is, their center positions never change.
  - These objects must be placed at the center of the hallway, and at the middle height of the wall (see Fig. 1).

What to submit:

- Submit your **source files (.cpp, .h)** that are needed for compilation.
- Also submit all your **texture image files (.bmp)** that are needed for texturing.
- Do not submit any project-related files (for example, `.sln` file).

How to submit:

- Use Canvas Assignment Submission system to submit your files.
- Make sure to zip all your files into `hw3.zip`, then submit your `hw3.zip` as a single file.

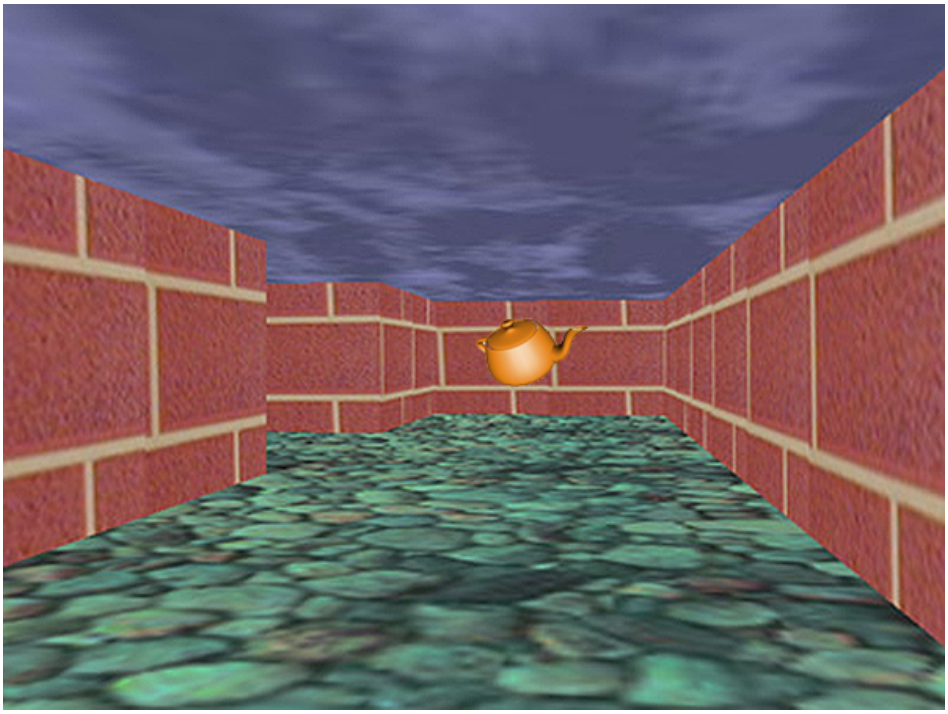

Figure 1: 3D Maze

Policy

- Do all the assignments on *Visual C++* using C++ and OpenGL.
- At the beginning of each file (.cpp, .h), provide comments specifying the author, date, and a brief description of the file.
- Source code (.cpp, .h) must contain enough comments here and there to make it easy to follow your code. Insufficient comments could lead to loss of points.
- Non-compilable program will get almost no credit (e.g., executable code not produced due to compile errors).
- Non-working program will get almost no credit (e.g., the executable is terminated immaturely due to run-time errors).
- Copying other's code is strictly prohibited. If identical (or nearly identical) submissions are found among students, every student involved will get automatic zero for the assignment. Same goes for copying existing code from online source.