A-Maze-ING! - A 2D Maze Using OpenGL

CSE-420 Project
by Andrew Yenalavitch & Kristian Howard Jr.

Objective

The objective of this project was to gain a better understanding of user interaction with 2-dimensional objects, data file processing and the real-time editing and rendering of this data using the OpenGL library.

Abstract

A-Maze-ING! creates a 10x10 maze based on a data file provided by the user, which is rendered using OpenGL. The objective of the game is to navigate the maze and arrive at the finish, wherein you win the game and are presented with a win screen. The program allows for tracking of player movement and live editing of the maze.

Introduction

We have never worked on a project like this before, so it was an interesting challenge. This project was an opportunity to gain a better understanding of OpenGL's 2d pipeline and user input.

Software Package

The software package includes a .cpp file, a starter maze data file, a template file for instructions on creating your own maze, and a Makefile for compilation. One of the biggest hurdles was overcoming the fact that we would need to use global variables because of a limitation with glutDisplayFunc() and passing objects. Our program reads in data from the file MazeData.txt, which is a list of 102 integers. The first 100 integers describe the maze using codes for the walls of each cell. The 101st integer is the number of the cell where the player will start, and the 102nd integer is the number of the cell where the game is won (finishing cell).

Here is a list of cell codes. Each code corresponds to a combination of walls surrounding the cell:
0 = left
1 = up
2 = right
3 = down
4 = left & up
5 = left & right
6 = left & down
7 = up & right
8 = up & down
9 = right & down
10 = left, up, & right
11 = up, right & down
12 = right, down & left
13 = down, left & up
14 = no walls

These codes are necessary for limiting player movement. So if you want to create a cell with walls on the left and right, you would assign that cell a value of 5.

We wanted the first cell, cell 0, to start in the upper left corner, which required us to think in reverse in terms of y, as (0,0) is the lower left corner of the viewport.

Here is a snippet of the drawMaze() function to display our solution to this challenge:

```
void drawMaze(){
for (int i = 0; i < 100; i++){
     if(cell[i] == 0){//left
       glBegin( GL_LINES );
         glVertex2i( (i%10)*50     , 500 - (i/10)*50 );             //left
         glVertex2i( (i%10)*50     , 500 - ((i/10)*50 + 50) );
       glEnd();
     }
     if(cell[i] == 1){//up
       glBegin( GL_LINES );
         glVertex2i( (i%10)*50     , 500 - (i/10)*50 );             //up
         glVertex2i( (i%10)*50 + 50, 500 - (i/10)*50 );
       glEnd();
     }
  (continues for all cases)
```

We chose to make each maze cell 50 pixels and the total maze size 10x10 for simplicity of calculations. One of the most exciting parts of the program was realizing that we could call the fillMaze() function on demand and it would allow us to edit the maze in real-time and re-display the data. Make a change to the data file, restart the maze. Don't like the change? Revert the change and restart the maze.

Here is the simple restart() function called by pressing 'r' on the keyboard:

```
void restart(){
  fillMaze();                        // reload maze data
  for (int i = 0; i < 100; i++){     // makes all cells unvisited
    visited[i] = false;
  }
}
```

We also added a function to fillMaze() called findDirection() that automatically aligns the player arrow towards an open wall when the maze is first rendered, which is nice for customized mazes:

```
void findDirection(){
  if ( validMove("UP") )
    faceDirection = "UP";
  else if ( validMove("RIGHT") )
    faceDirection = "RIGHT";
  else if ( validMove("DOWN") )
    faceDirection = "DOWN";
  else
    faceDirection = "LEFT";
}
```

We also wanted to find a way to easily display text for the win screen without having to manually draw the letters, and after doing some research found the glutStrokeCharacter() function. This allowed us to display "You Win!" will different colors for each character without having to draw them manually with GL_LINES as we did with the F for the winning cell. Here is a snippet from the drawWinScreen() function :

```
  glColor3f( 1.0, 0.0, 0.0 );
  glutStrokeCharacter(GLUT_STROKE_ROMAN, 'Y'); // "You win!" in various colors
  glColor3f( 0.0, 1.0, 0.0 );
  glutStrokeCharacter(GLUT_STROKE_ROMAN, 'o');
  glColor3f( 0.5, 1.0, 0.0 );
  glutStrokeCharacter(GLUT_STROKE_ROMAN, 'u');
  glutStrokeCharacter(GLUT_STROKE_ROMAN, ' ');
  glColor3f( 0.5, 0.0, 0.5 );
  glutStrokeCharacter(GLUT_STROKE_ROMAN, 'W');
  glColor3f( 1.0, 0.5, 0.0 );
  glutStrokeCharacter(GLUT_STROKE_ROMAN, 'i');
  glColor3f( 0.0, 1.0, 0.0 );
  glutStrokeCharacter(GLUT_STROKE_ROMAN, 'n');
  glColor3f( 1.0, 0.0, 0.0 );
  glutStrokeCharacter(GLUT_STROKE_ROMAN, '!');
}
```

We wanted a player to be able to see where he has already traveled, so we used an array of bools. It is initially set to false, but when a player visits a cell, that cell's corresponding bool is set to true. The drawTrail() function then iterates through the array and draws a blue square on visited cells when trailsEnabled has been toggled to true:

```
void drawTrail(){
   glPolygonMode( GL_FRONT, GL_FILL );
   glColor3f( 0.0, 0.0, 1.0 ); //blue

   for (int i = 0; i < 100; i++){
     if(visited[i] == true){
       glBegin( GL_POLYGON ); //draw trail
         glVertex2i( (i % 10)*50 + 20, 500 - (i / 10)*50 - 15);
         glVertex2i( (i % 10)*50 + 35, 500 - (i / 10)*50 - 15);
         glVertex2i( (i % 10)*50 + 35, 500 - (i / 10)*50 - 30);
         glVertex2i( (i % 10)*50 + 20, 500 - (i / 10)*50 - 30);
       glEnd();
     }
   }
}
```

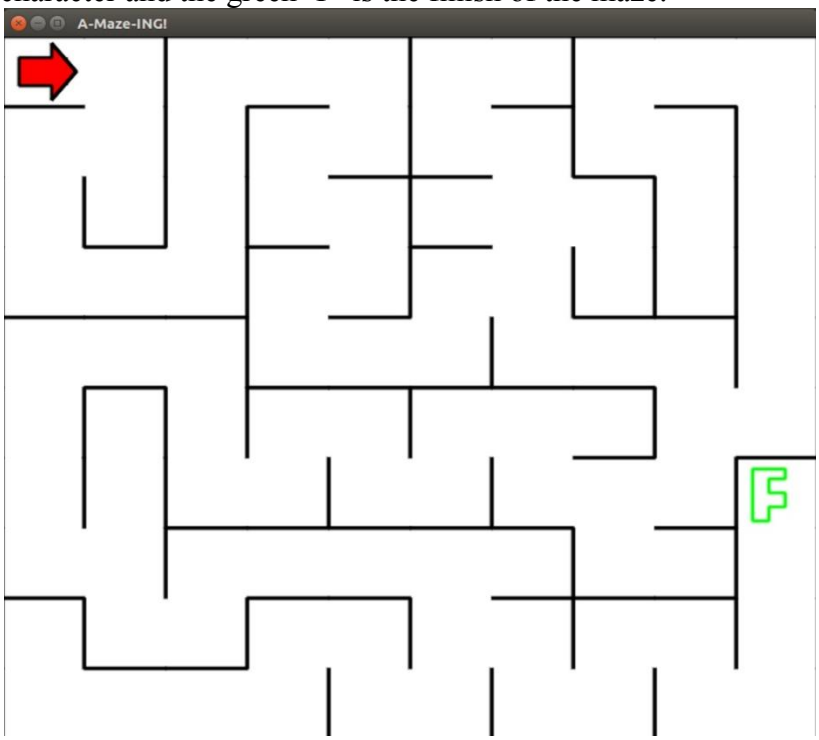
And lastly, we needed a way to control the most important object, the player arrow! This is another example of why we chose 10x10, as it made calculations for movement easier. Moving up and down changes the row, and is done in increments/decrements of 10. Left and right moves along the row and is done in increments/decrements of 1. Here is a snippet from the keyboard() function:

```
case 'w':  // move up if you can
  faceDirection = "UP";
  if (validMove("UP") && !checkWin()){
    setVisited(currentCell);
```

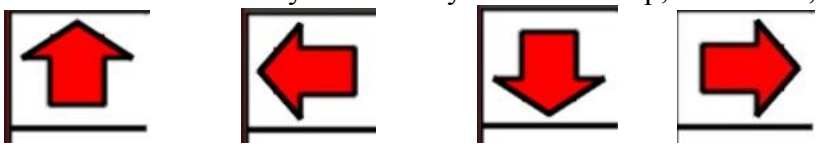
```
    currentCell -= 10;
  }
  glutPostRedisplay();
  break;
case 'a':  // move left if you can
  faceDirection = "LEFT";
  if (validMove("LEFT") && !checkWin()){
    setVisited(currentCell);
    currentCell--;
  }
  glutPostRedisplay();
  break;
    (right and down are also included)
```

The Interface

The user is presented with a render of the maze data on startup. The arrow represents the player character and the green ‘F’ is the finish of the maze.

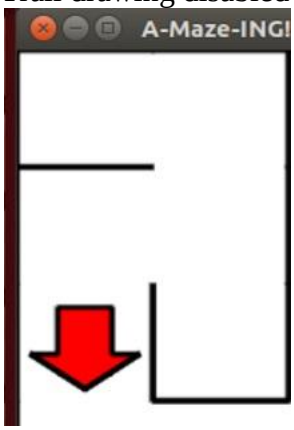


The arrow initially points to the an open wall and changes orientation whenever the user inputs one of the direction keys. These keys are ‘w’ for up, ‘a’ for left, ‘s’ for down, and ‘d’ for right.

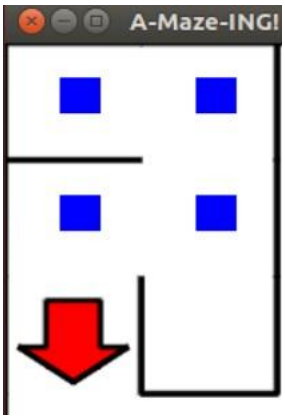


The game tracks what blocks the player has moved onto and displays a blue square in these cells. The setting is off by default to reduce visual clutter, but can be toggled on using the ‘t’ key on the keyboard.

Trail drawing disabled (default):

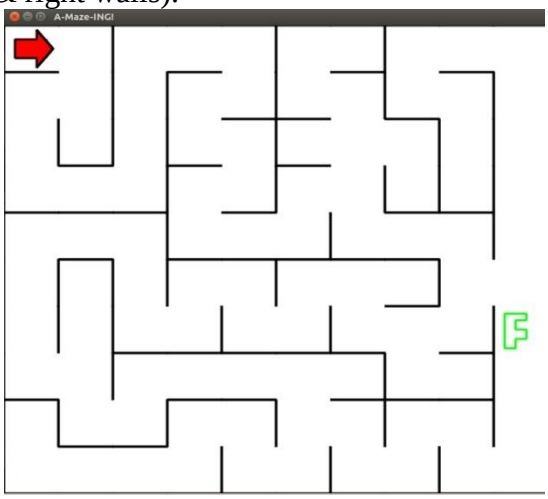


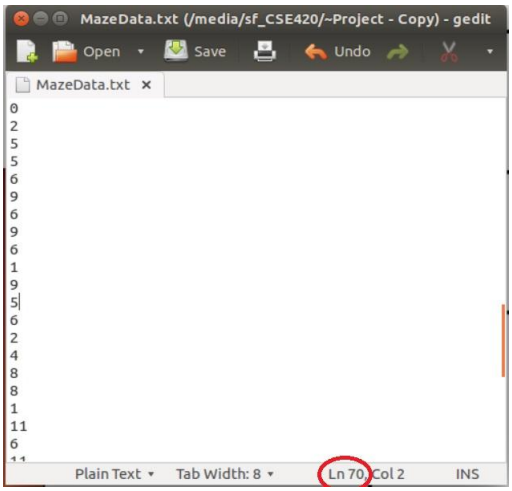
Trail drawing enabled:



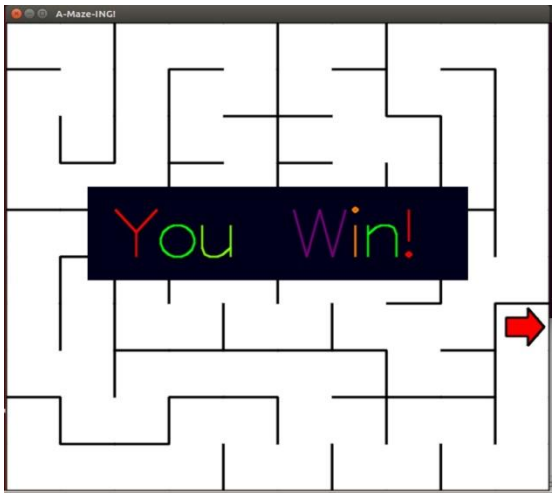
At any time, you can press the ‘r’ key to restart the game. This function also allows the user to edit the maze on the fly, making for easy maze creation with a text editor like gedit. Each of the first hundred lines of “MazeData.txt” corresponds to a cell, with cell 0 being the cell in the upper left and line 1 in “MazeData.txt”, and cell 99 being the cell in the lower right and being represented by line 100. The last 2 lines of “MazeData.txt,” lines 101 and 102, correspond to the player start and winning cells, respectively.

Say we want to cheat and remove the wall above the winning cell. To do so we need to alter lines 60 and 70 so that this wall is no longer displayed. Line 60 is changed from a 9 (right and bottom walls) to a 2 (right wall only), and line 70 is changed from a 10 (left, up & right walls) to a 5 (left & right walls).





The starting and winning cells can be changed in a similar fashion. When the player reaches the finish of the maze (the green F) the player is presented with the text “You Win!”, and player movement is disabled (however you can spin the arrow around using the wasd keys in a celebration dance if you like!).



Conclusions

This was a very challenging project that went through several iterations before coming to the final form you see here. We'd like to eventually take this into 3-dimensions, but time constraints from other class work will probably preclude this. Overall, overcoming the presented challenges was extremely satisfying

References

Dr. Tong Yu's lecture notes

http://cse.csusb.edu/tongyu/courses/cs420/notes/index.php

OpenGL Utility Toolkit (GLUT) Programming Interface API Version 3 (for glutStrokeCharacter**)**

https://www.opengl.org/resources/libraries/glut/spec3/node78.html