For assignment 1, part 1 is complete along with the drawing of part 2 and part 3. For part 1, the inner maze walls are hard-coded vectors that are always drawn. For the outer walls, they are drawn as 16 individual vectors that surround the maze. To draw more efficiency, I condensed all 16 lines into one buffer with its x and y values changing according to their value in relation to each other. For example, the y value will constantly increase until it reaches its max (1.0), then it will start increasing x. Once x is the max value, y lower to the minimum value, followed by x. With this method, all I need to create two exits is to skip 2 instances of drawing the vector. This is accomplished by comparing two random numbers that are not equal to each other with the current i value in the for loop, skipping the vector creation for that individual wall.

For part 2, I was able to draw the maze of grids through a similar method of the walls above. The squares for each grid are made by drawing individual squares whose coordinates are determined by the origin point x and y and the height and width each square should be. Once all the squares are drawn horizontally, the next square vertically is drawn at the left. To erase lines to make an exit, code was developed that uses 1 random number to choose a side to erase from and another random number to choose 1 of 4 vertices to draw over with the color of the background. Unfortunately, repeated attempts to draw over the lines to create exits were unsuccessful, so I was unable to progress as this requirement would be needed to draw the green lines and let the inner lines disappear.

For part 3, I implemented the Brownian motion fly into the maze from the first part. By implementing the init and display aspects of the Brownian motion code into the init and display of the first maze, allowing an easier implementation than attempting to reference both methods. For some reason, the first drawn vector is drawing additional vectors (i.e. a 2 part vector draws 3 lines for some reason). In order to work on collision, I use the current node's x and y coordinates to detect whether the node crosses the range where the line is. The program will exit once this condition works. Unfortunately, this collision detection is not working as expected.

After this initial submission, I will work on making lines disappear and draw green lines, then work on the mouse detection. After that, I will find the bugs that stop the collision detection from working and why the extra line is drawn.