Cassadie Moore

Nate Williams

Jacob Brook

Dungeon Crawler 🡪 Random Maze Generator

Updated requirements:

Our project is creating a randomized maze. We have it set up to output a 20 by 20 square that the maze can be generated within, but we have our code set up so that we can change the size of the maze as long as it is a square and not a rectangle. The maze generator is essentially a random, compiler-controlled bulldozer that goes through the starting block of ascii art and breaks it down, creating the open spaces that will help set up a visual of the maze. The bulldozer starts at a position that is right inside the top left corner of the block of ascii art. As the code runs through, there are various functions that check the availability of the directions around the bulldozer’s current position in the maze. The bulldozer can only cross the same spot once after the initial bulldozing so that the maze doen’t turn into a solid block of nothing. We used a stack along with a class of coordinates to ensure that we don’t create a big empty space. The stack holds the coordinates of where the bulldozer has been while making the maze, that way if the bulldozer gets to a spot where it has already bulldozed and intersected the bulldozer can pop a few coordinates off the top of the stack and undo the last few moves it made and go in a different direction to continue the maze.

We had originally planned to provide a representation of the user so they could navigate the maze and we planned to have monsters to stand in the way of the adventurer as they went through the maze but we ran out of time and ran into more issues than we had initially anticipated while writing the random maze generating code.

Our coordinate class is simply to keep track of where the bulldozer is while it’s making the maze in a way that makes the coordinates accessible to a stack. The UML is as follows:

|  |
| --- |
| Coordinates |
| * int x * int y |
| + Coords( )  + void setCoords(int a, int b)  + int getX( )  + int getY( ) |