public static int[][] mapMaker(){

/\*Map maker uses a 2d array to create a grid our player can navigate

It does this by setting each spot t oa specific value. The ones we use are:

0 - Wall(can't walk on it)

1 - Floor(also can't walk on it..... kidding)

2 - Player's starting location(acts as normal floor)

3 - Customer's staring location(can't be walked on, that's mean)

4-9 - different tacos

\*/

int dimensions = 15;//Represents the width and height of the grid. Our grid is gonna be a 15x15 square

int maxTunnels = 50;//The maximum amount of 'tunnels.' Higher or lower influences how many spaces on the grid are open floor vs wall

int maxLength = RNG.nextInt(3)+2;//Maximum length of each tunnel. By choosing a random number between 2 and 5, we allow for either really small or large maps. You'll see when you play it

int[][] map = new int[dimensions][dimensions];//Make a 2d array to represent our game grid, with the proper dimensions

//These two represent the current row and column we are working in. Initializing them like this means the grid-maker starts in a different spot every time

int currentRow = (RNG.nextInt(dimensions));

int currentCol = (RNG.nextInt(dimensions));

int tacoCount = 4;//Used to place the ingredients

//This 2d array is kinda cool. Basically the 4 elements represent the 4 compass directions (north, south, east, west)

int[][] directions = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

//To store the previous direction the grid-maker moved in. When we chose a new one, we want it to be different

int[] lastDir = new int[2];

//Stores that new direction

int[] randDir = new int[2];

while (maxTunnels>0){//The maxTunnels variable goes down by 1 each time a new tunnel is made, so stop when we've reached the limit

do{

randDir = directions[RNG.nextInt(directions.length)];//Choose a random direction to move in...

}while((randDir[0]==lastDir[0]\*-1 && randDir[1] == lastDir[1]\*-1 || randDir[0]==lastDir[0] && randDir[1] == lastDir[1]));//...so long as its perpendicular to the last one

int randLength = RNG.nextInt(maxLength)+1;//Choose a random length for the next tunnel. +1 to ensure there are no 0-length tunnels cause thats just weird

int tunnelLength = 0;//Length of the tunnel we are making right now. It'll increase until it's reached randLength

while(tunnelLength<randLength)//I told you so

{

if (((currentRow == 0) && (randDir[0] == -1)) || ((currentCol == 0) && (randDir[1] == -1)) ||((currentRow == dimensions - 1) && (randDir[0] == 1)) || ((currentCol == dimensions - 1) && (randDir[1] == 1))) {

break;

//Holy crap thats a lot of conditions.

//Basically we're checking to make sure we don't try and tunnel beyond the bounds of the grid. If we are, break out of the loop. Tunnel complete.

}

else{

map[currentCol][currentRow] = 1;//Set the current spot to 1, which represents walkable floor

currentRow += randDir[0]; //Add the 2 values from our current direction vector (physics flashbacks) to the currentRow/Column variables

currentCol += randDir[1];//These values are either 1, 0, or -1. This sets our location to the next place

tunnelLength++; //Tunnel is now a little bit longer

}

}

if (tunnelLength==maxLength){//If we've reached the length...

lastDir = randDir;//Save our last direction

maxTunnels--;//One less tunnel to make

}

//aaaaaaaand repeat

}

/\*now that we've made our map of 1s(floor) and 0s (walls), we can start placing in the other stuff, like Waluigi.\*/

Boolean playerSpawned = false;//We haven't placed Waluigi anywhere yet

Boolean customerSpawned = false;//Haven't placed anyone else yet either...

/\*Loop to place Waluigi in a random spot\*/

while (playerSpawned==false) {

xCord = RNG.nextInt(15);//Choose a random place to spawn our boy Waluigi

yCord = RNG.nextInt(15);

if (map[xCord][yCord]==1)//Make sure it's walkable floor

{

map[xCord][yCord]=2;//Set our spawn location to "2," that way we know it's waluigi

playerSpawned = true;//Yay we spawned him

}

//repeat until we've found a home for poor waluigi

}

/\*Loop to place Mario in a random spot\*/

while (customerSpawned==false) {

//Same as above loop, except we set the customer's spot to 3 instead of 2.

xCordCustomer = RNG.nextInt(15);

yCordCustomer = RNG.nextInt(15);

if (map[xCordCustomer][yCordCustomer]==1)

{

map[xCordCustomer][yCordCustomer]=3;

customerSpawned = true;

}

}

/\*Loop to place all of the ingredients\*/

while (tacoCount!=10){//The ingredient counter starts at 4 (cause remember how tiles with #s 4-9 are the different ingredients?). Go until it gets to 10

int randX = RNG.nextInt(15);//Choose a random x-y spot on the grid

int randY = RNG.nextInt(15);

if (map[randX][randY]==1) {//make sure that spot is open floor, we dont want to have a random taco in the middle of the wall

map[randX][randY] = tacoCount;//set that tile to whatever ingredient we're on

tacoCount++;//next ingredient

}

}

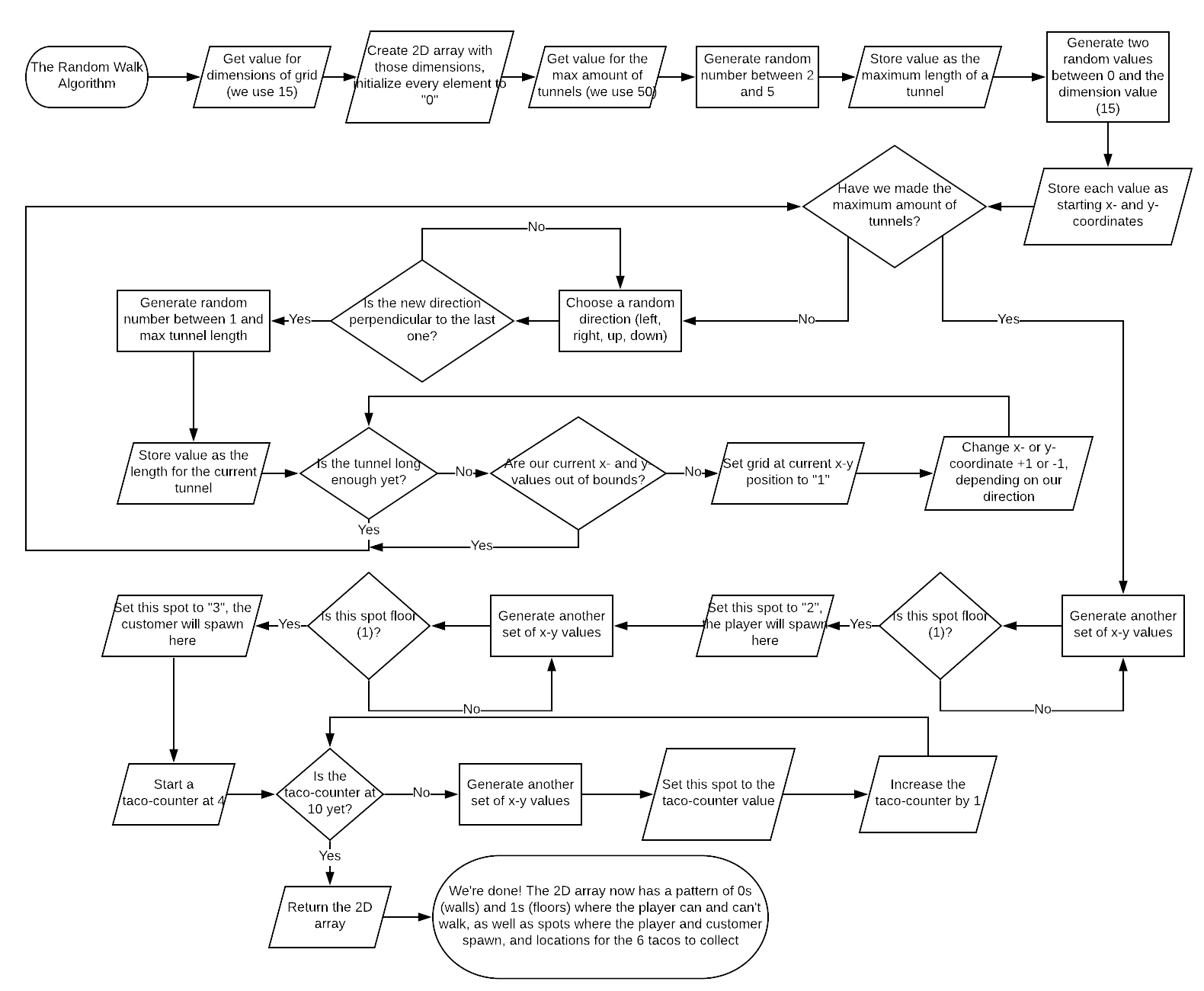
/\*Now that we've placed everything we have to place, we're done!\*/

return map;//We've made our map

}

Ipo for rand walk

|  |  |  |
| --- | --- | --- |
| i | p | o |
| Dimensions of square grid (ours is always 15\*15) | Make 2d array with the dimensions  (int[15][15])  Set every spot to 0 (wall) | A nicely made 2d array map of 0s and 1s that represent places a player can and can’t go |
| Max # of tunnels to make (complexity of map, we keep ours at 50 for a good balance) | Choose random direction (left, right, up, down) and a random x-y spot to start from |  |
| Maximum length of each tunnel(we chose a random between 2 and 5 so our maps are either reallysmall or really big) | Set current spot to 1 (floor) and move a spot ahead in whatever direction we chose, repeat until the current tunnel is as long as the max # of tunnels  Check each time we move ahead a spot if we’re going outside our grid, and if we are end the tunnel early |  |
|  | Repeat making tunnels until we’ve made how many we were told to make.  Choose a new direction after each tunnel that’s perpendicular to the last |  |

Flowchart:

Pseudocode:

Get dimensions (these 3 can be whatever you want)

Get maxTunnels

Get maxLength

Directions = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}}

map = 0 (it’s a 2d array, everything starts at 0)

currentRow = random less than dimensions

currentColumn = random less than dimensions

while(maxTunnels > 0)

while(RandomDirection is perpendicular to LastDirection)

\*We’ll figure out how to do this condition in the actual code\*

RandomDirection = directions[random between 0 and 3]

RandomLength = random between 1 and maxLength

TunnelLength = 0

while (TunnelLength < RandomLength)

If currentRow is 0 and RandomDirection is -1 stop

If currentColumn is 0 and RandomDirection is -1 stop

If currentRow is dimensions and RandomDirection is 1 stop

If currentRow is dimensions and RandomDirection is 1 stop

Else

map[currentRow][currentColumn] = 1

Add RandomDirection[0] to currentRow

Add RandomDirection[1] to currentColumn

Add 1 to TunnelLength

If tunnelLength equals maxLength

LastDirection = RandomDirection

Subtract 1 from maxTunnels

PlayerSpawned = false

CustomerSpawned = false

tacoCount = 4

While (player isn’t spawned)

Xcooordinate = random less than dimensions

Ycoordinate = random less than dimensions

If map[Xcooordinate][Ycooordinate] is 1

map[Xcooordinate][Ycooordinate] = 2

PlayerSpawned = true

While (customer isn’t spawned)

Xcooordinate = random less than dimensions

Ycoordinate = random less than dimensions

If map[Xcooordinate][Ycooordinate] is 1

map[Xcooordinate][Ycooordinate] = 3

CustomerSpawned = true

While (tacoCount isn’t 10)

Xcooordinate = random less than dimensions

Ycoordinate = random less than dimensions

If map[Xcooordinate][Ycooordinate] is 1

map[Xcooordinate][Ycooordinate] = tacoCount

tacoCount goes up 1

Return map