Specifications:

In this program you are creating islands/continents out of a simple procedural generation technique: Dirtball Dropping.

The user will enter sizes for a 2D Array that will represent the land to be terraformed. You will prompt for how big the dirtballs should be and how 'strong' they should be. Finally, you'll ask for how many you want to hit the land with. After running the generation algorithm, you'll refine the terraformed array into a polished ASCII-Art display.

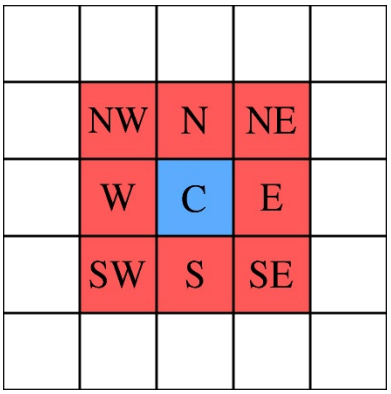
Setup:

1. Get a width and height from the user
2. Create a dynamically allocated 2D integer array based on that width and height
3. Initialize your array to 0
4. Ask the user for a value between 40 and 200 to use as the water-line
5. Get a radius from the user
6. Get a power rating from the user
7. Ask for how many dirtballs to drop

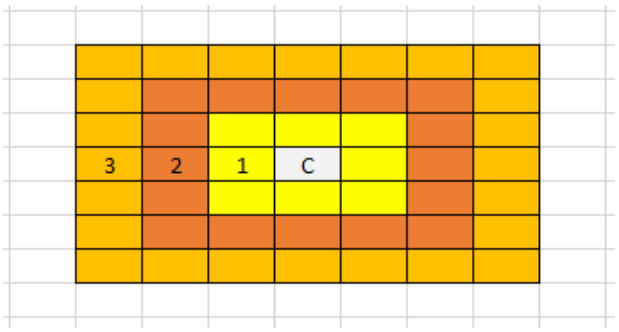
Dropping the Dirtballs

This is deceptively simple.

- Pick a random X and Y center point such that:
  - 0 <= centerX < width of your array
  - 0 <= centerY < height or your array
- Process an "extended Moore neighborhood" based on the radius of the dirtballs
  - A Moore Neighborhood is the 8 indexes directly around a center index in a 2D array
  - The "extended" version would add square layers around the center index based on the radius
  - *Hint: make sure the index is within bounds of the array*
- For each index in the "Extended Moore Neighborhood" calculate the distance from that index to the center point
  - $distance = \sqrt{(x - centerX)^2 + (y - centerY)^2}$
- If distance > radius then ignore the index
- If distance <= radius then calculate the impact value
  - $impact\ value = power\_rating - floor(distance)$
- Add the impact value to the value in the index



Moore Neighborhood around center index 'C'



Extended Moore Neighborhood with Radius 3

Repeatedly drop dirtballs until you've dropped as many as the user asked for.

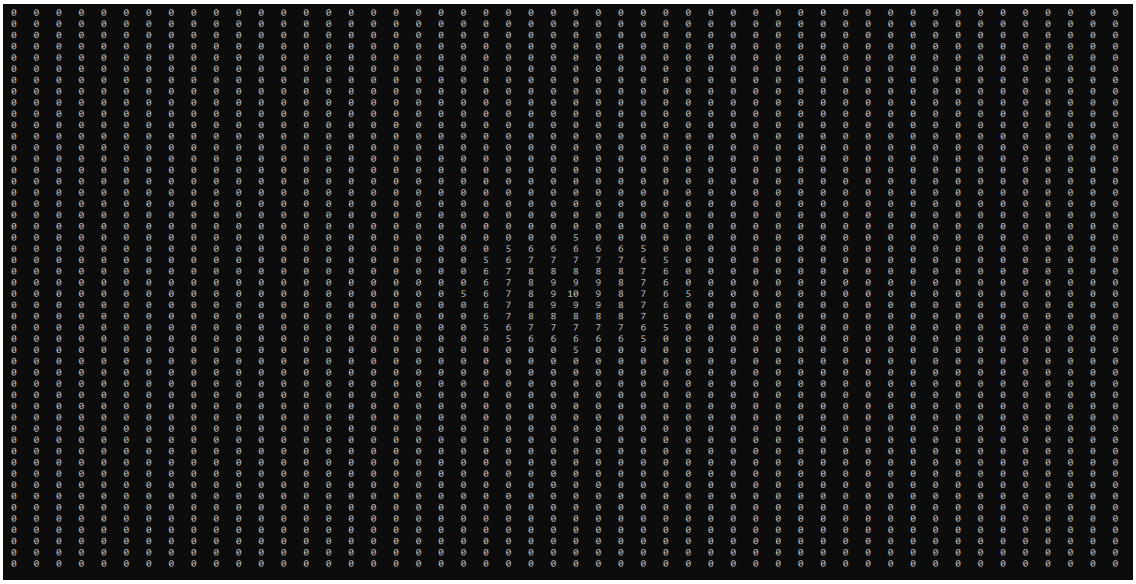
Extended Moor Neighborhood with impact calculation Pattern

If you drop a Dirtball with radius 5 and power 5 you should get a pattern of values that looks like this:

0	0	1	1	1	1	1	0	0
0	1	2	2	2	2	2	1	0
1	2	3	3	3	3	3	2	1
1	2	3	4	4	4	3	2	1
1	2	3	4	5	4	3	2	1
1	2	3	4	4	4	3	2	1
1	2	3	3	3	3	3	2	1
0	1	2	2	2	2	2	1	0
0	0	1	1	1	1	1	0	0

Single Dirtball dropped in the middle of the grid

- Width:50
- Height: 50
- Radius: 5
- Power: 10



Sample Raw Integers Terraforming:

- Width:50
- Height: 50
- Radius: 5
- Power: 10
- Number of drops: 100

Polishing the Landmass

After you have generated the values for the landmass, do a quick analysis to find the highest value in the 2D array.

Normalize the values in the 2D array by dividing every value in the 2D array by that maximum (don't forget to use floating point division) then multiplying by 255. This will make every index a value between 0 and 255.

Create a 2D character array of the same size as your value array.

Process through the values in your array and set a character in your 2D character array based on the value's relationship to the water-line (from the set-up).

Output your raw island, normalized island, and polished island to the console AND to a file (island.txt is fine).

Calculate land-zone as 255 – water-line.

- value < 50% of water-line
  - '#' – deep water
- value >= 50% of water-line && <= water-line
  - '~' – shallow water

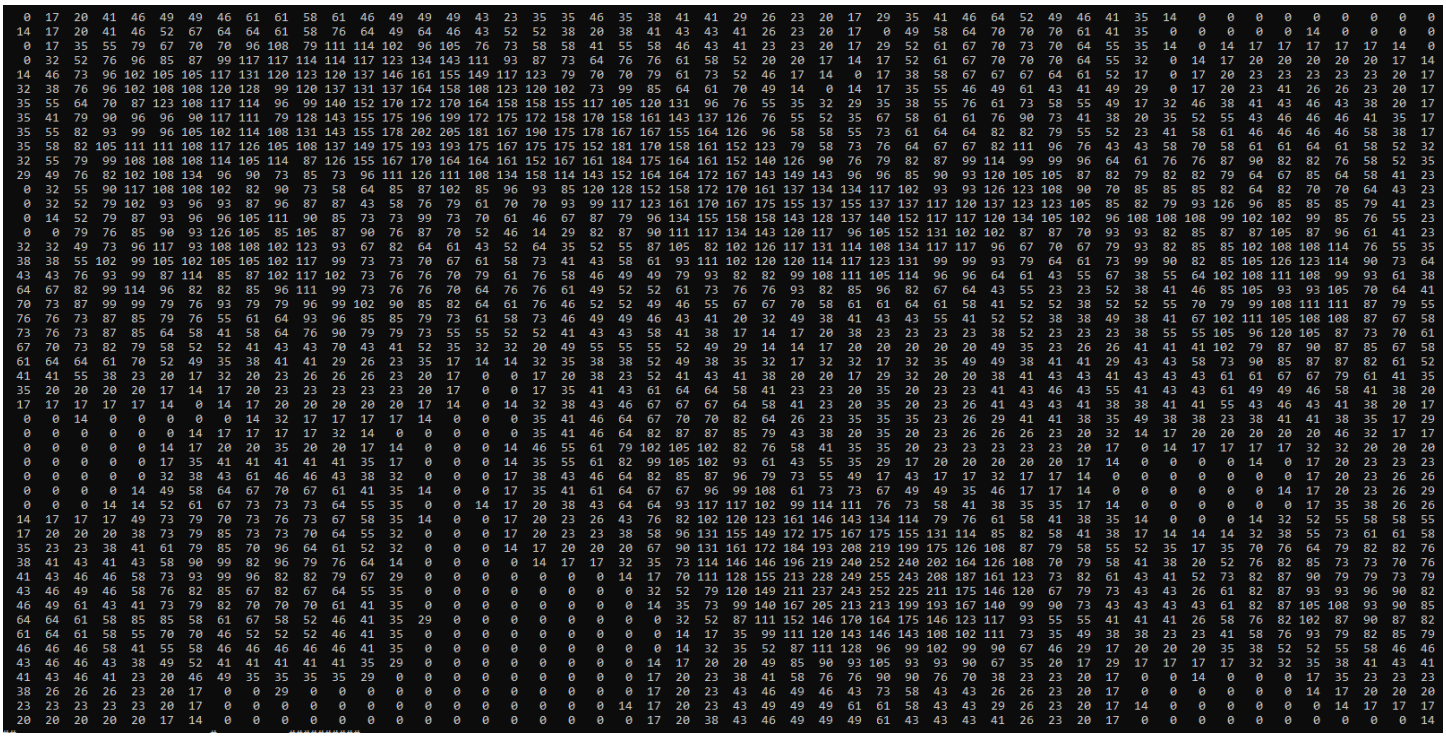
Everything else is > water-line &&

- < (water-line + 15% of land-zone)
  - '.' - coast/beach
- >= (water-line + 15% of land-zone) && < (water-line + 40% of land-zone)
  - '-' – plains/grass
- >= (water-line + 40% of land-zone) && < (water-line + 80% of land-zone)
  - '*' – forests
- else:
  - '^' – mountains

Example calculations:

waterline = 70
landzone = 185
0 - 34 → deep water '#'
35 - 70 → shallow water '~'
71 - 97 → coast '.'
98 - 143 → plains '-'
144 - 217 → forest '*'
> 217 → mountain '^'

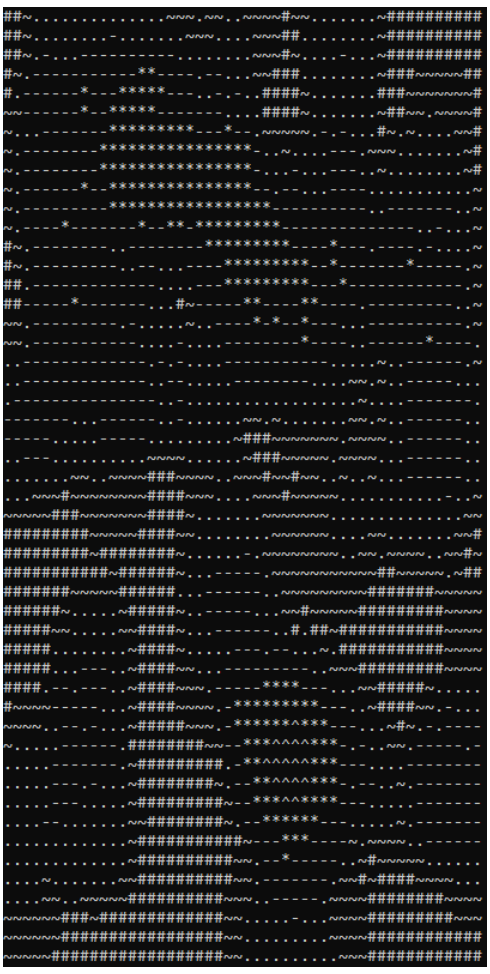
Landmass Normalized:



Polished

Waterline = 70

Waterline = 40



Interface:

Command line argument:

Add a command line option -s <integer> to allow the user to provide a seed for the random number generator. This seed value should be given to srand(<int>). If no -s option is given, then seed the random number generator with time: srand(time(0));

Console input:

Your interface must follow this pattern.

```
Welcome to <your name here>'s CSE240 Terraformer!!

Enter grid width:
Enter grid height:
Enter value for waterline (40-200):
Enter dirtball radius (minimum 2):
Enter dirtball power rating (minimum = radius):
Enter number of dirtballs to drop:
```

Once again, this is just the script for the inputs. It is not exhaustive. I'm leaving out defensive checks and re-prompts. You can also customize the wording.

Output:

You are to output to both the console and files (you will output 3 files):

- Raw 2D array to console
  - out to file raw_landmass.txt
  - this should use setw(4) or %4d (with printf/fprintf)
- 2 newlines to console
- Normalized 2D array
  - out to file normalized_landmass.txt
  - this should use setw(4) or %4d (with printf/fprintf)
- 2 newlines to console
- Polished ASCII character array
  - out to file final_landmass.txt
  - there should be no extra spaces in between characters

Recommended Functions

Quality modularization of your code is part of your Code-Quality score in the rubric.

These function recommendations are for your benefit to make organizing this project easier …

dropDirtBall

```
void dropDirtBall(int** landmass, int maxWidth, int maxHeight,
                      int centerX, int centerY, int radius, int power)
```

This function performs the dirtBall drop centered on [centerY][centerX] on your raw value 2D integer array. See previous for algorithm with Extended Moore neighborhood, radius and power calculations.

boundsCheck

```
bool boundsCheck(int x, int y, int minx, int miny, int maxx, int maxy)
```

This function returns True or False that the index [y][x] is within the bounds of your 2D array.

findMax

```
int findMax(int** map, int width, int height)
```

This function finds the maximum value in the map and returns it.

normalizeMap

```
void normalizeMap(int** map, int width, int height, int maxVal)
```

Performs the normalization operation on the map data. You could have this return a new array instead if you want to hold onto the original map data for any reason.

frand

```
double frand()
```

Returns a random number between 0 and 1.

printLand

```
void printLand(int** land, int width, int height)
```

Outputs your landmass array. If you're using C++ feel free to overload it for your other arrays too.

finalizeMap

```
char** finalizeMap(int** map, int maxX, int maxY)
```

Performs the integer data to character data calculations and creates the polished map.