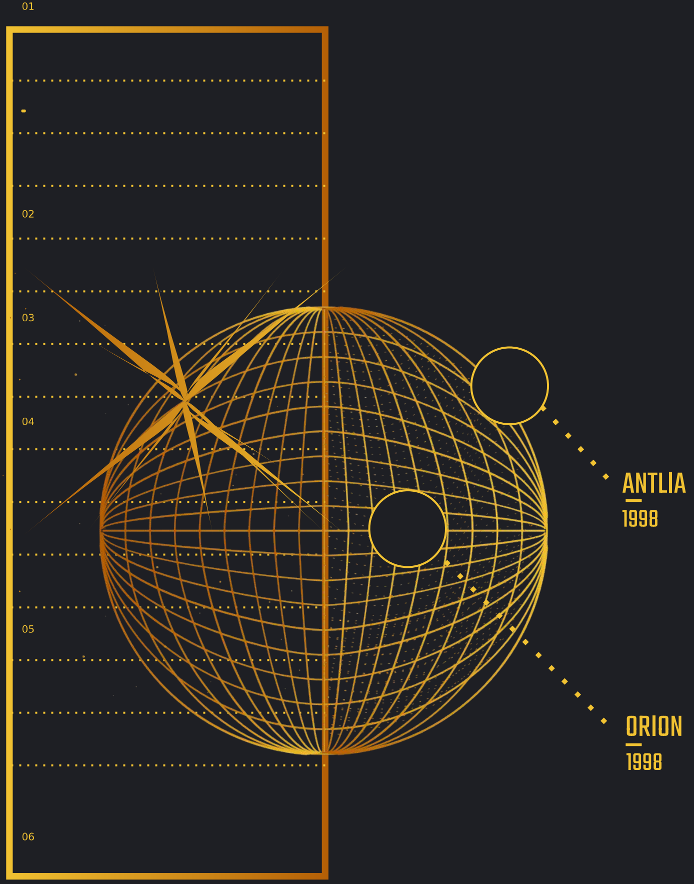
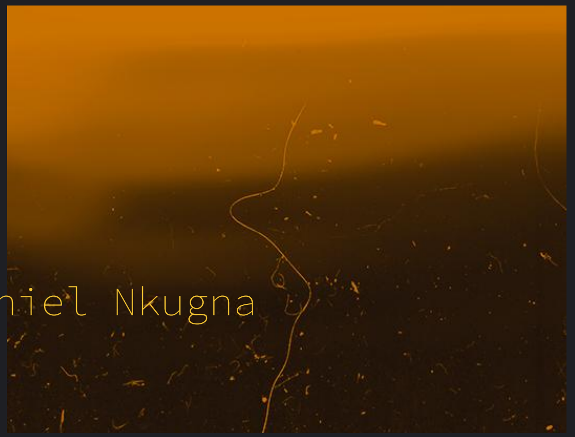
MCSD Burrows

Building Dwellings





Daniel Nkugna

TABLE OF CONTENTS

OVERVIEW 01.

Explanation of the MCSD Project

Search Process and Algorithms 02.

How and Why I attacked the problem

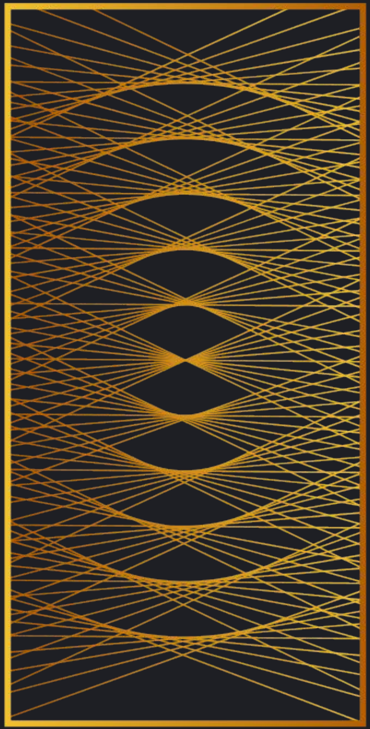
Results 03.

The dwelling spots I choose

Advancements 04.

How results would have improved

003-1040559 1250 003-77156.8 1760 0009-14563.7 73273



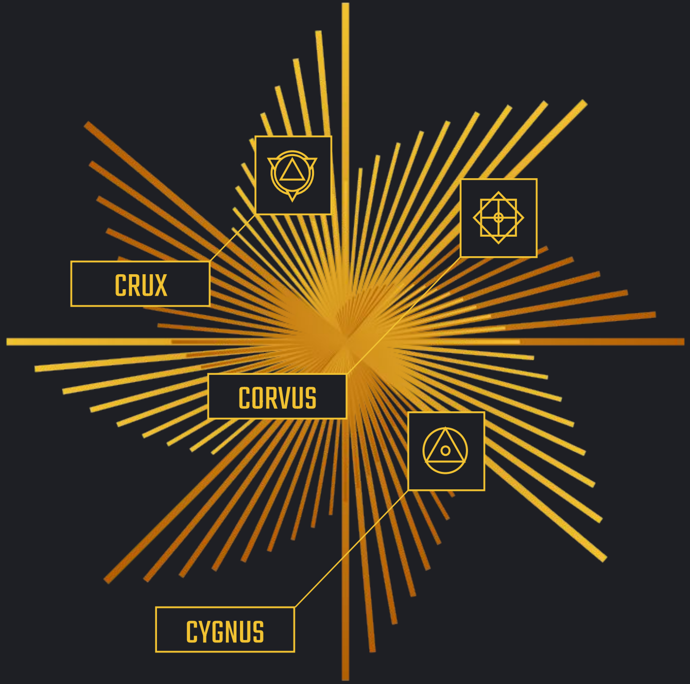
01.

Overview

An explanation of Martian Squirrel Cave Dwellings and “Mars Colonization”

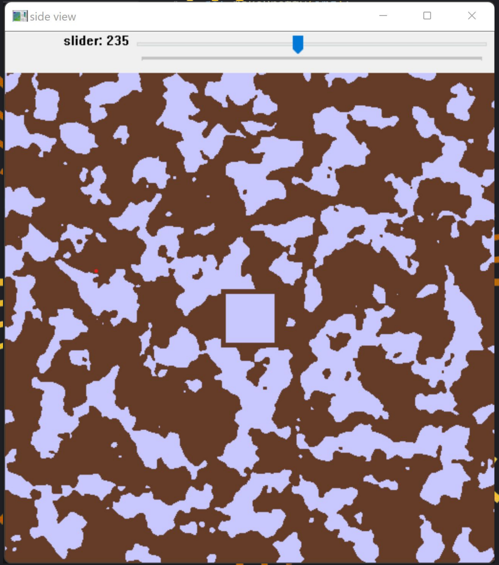
What is MCSD?

Martian Cave Squirrel Dwellings is a made up problem set in the year 20XX. Squirrels are sent to colonize Mars where the terrain consist of only dirt and air. The squirrels are meant live is a sizable underground in a burrow that must have some sort of connection to other martian land (Mars still has gravity so things can’t float!). Dirt also cannot be created or destroyed.



What is MCSD in CS Terms?

Mars is a 512x512x512 binary cube where 1 is dirt and 0 is air. A dwelling 51x51x51 cube where the inner 41x41x41 volume is hollow. Replacing dirt means after creating a burrow, the dirt must be replaced somewhere around the burrow and making sure the cave can’t float means making sure there is some part of the dwelling touching some other dirt.

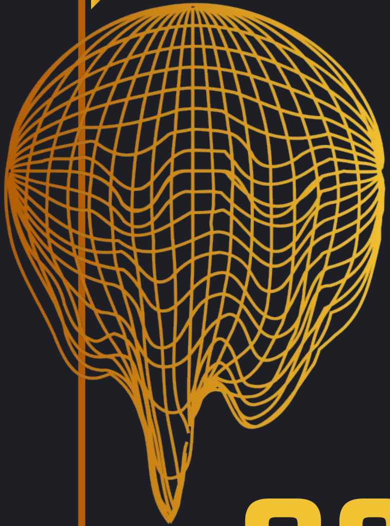


T

2

Search Process and Algorithms

How to search for the best cave



02.

003-1040559 1250 003-77156.8 1760 0009-14563.7 73273

Development Timeline

T

4

1

Search Process

Create a faster cost estimate for burrows

2

Running Price Check

Multiprocessing cost estimate

3

Trimming Caves

Determining which caves to build

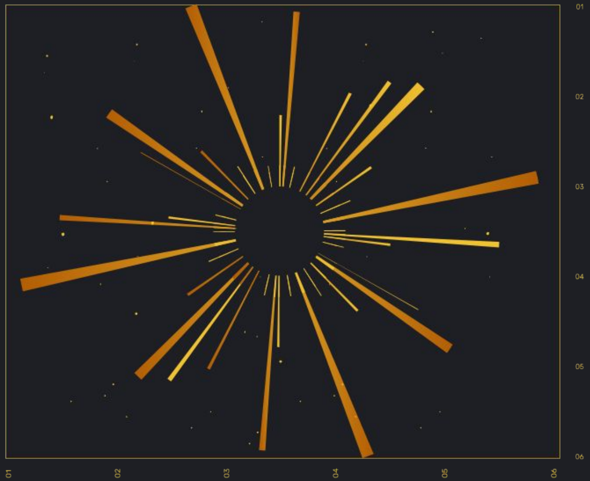
4

Replacing Dirt

Floodfilling missing dirt

003-1040559 1250 003-77156.8 1760 0009-14563.7 73273

1. Search Process



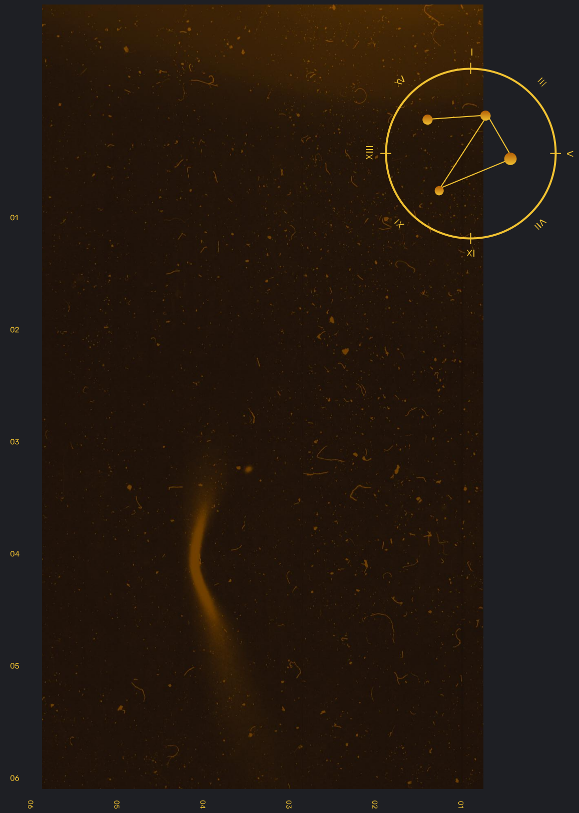
- The best way to find the cheapest cave is to iterate through every possible location for a cave
    - Given cost check takes 1.3 seconds to run which would make the process take 48,467 hours
- Creating a newer, faster cost estimate is needed to reasonable do the process

1. Search Process Continued

- New cost estimate determines cost of the cave by determining how much dirt needs to be moved
    - Sums how much dirt is in the interior, air is in the exterior, and returns it as the cost (with a bit more math)

```
def pce(cave, x, y, z, size = 41, wall = 5):
    a = np.sum(cave[x - size // 2 : x + size // 2 + 1,
                    y - size // 2 : y + size // 2 + 1,
                    z - size // 2 : z + size // 2 + 1])

    b = np.sum(cave[x - size // 2 - wall : x + size // 2 + 1 + wall,
                    y - size // 2 - wall : y + size // 2 + 1 + wall,
                    z - size // 2 - wall : z + size // 2 + 1 + wall])
    full = b
    interior = a
    pc = 63730 #perfect cave = pc
    cost = (interior + max(pc - full, 0)) * 2
    return cost
```

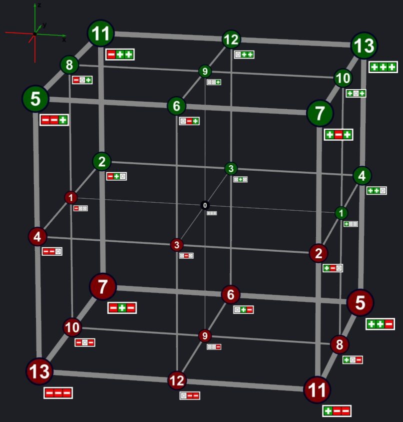


2. Running Price Check

- Previous attempts at this project took eight hours to find all locations while still skipping locations
- To make this attempt faster, the cave was split into octant with eight instance of the programs running

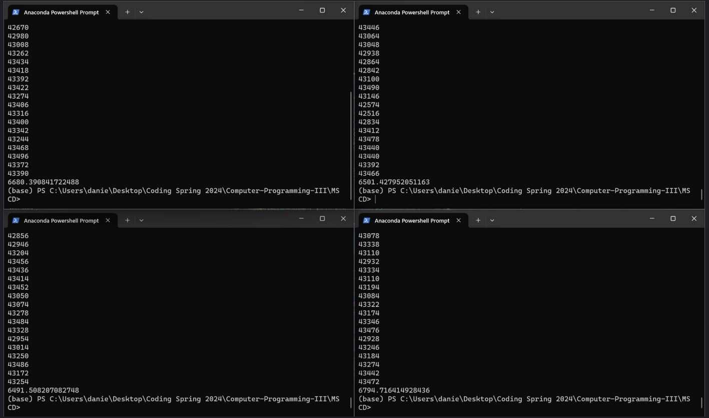
2. Running Price Check

All programs finished under two hours



Example of octants of a cube

Example of for instances of the program running

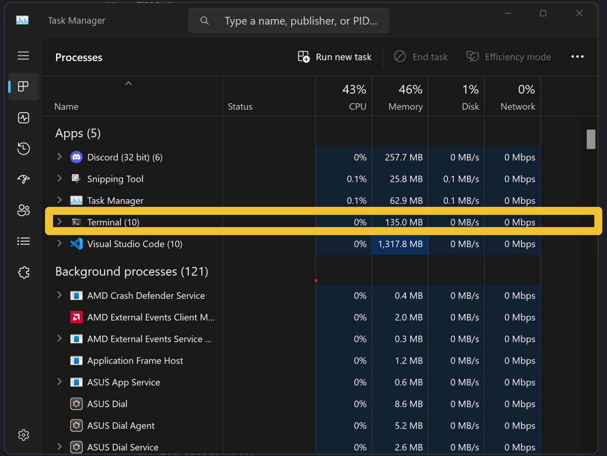


```
42670
42980
43008
43262
43434
43418
43392
43422
43274
43406
43316
43400
43342
43244
43468
43496
43372
43390
6680.390841722488
(base) PS C:\Users\danie\Desktop\Coding Spring 2024\Computer-Programming-III\MS
CD>
```

```
43446
43064
43048
42938
42864
42842
43100
43490
43146
42574
42516
42834
43412
43478
43440
43440
43392
43466
6501.427952051163
(base) PS C:\Users\danie\Desktop\Coding Spring 2024\Computer-Programming-III\MS
CD>
```

```
42856
42946
43204
43456
43436
43414
43452
43050
43074
43278
43484
43328
42954
43014
43250
43486
43172
43254
6491.508207082748
(base) PS C:\Users\danie\Desktop\Coding Spring 2024\Computer-Programming-III\MS
CD>
```

```
43078
43338
43110
42932
43334
43110
43194
43084
43322
43174
43346
43476
42928
43246
43184
43274
43442
43472
6794.716414928436
(base) PS C:\Users\danie\Desktop\Coding Spring 2024\Computer-Programming-III\MS
CD>
```



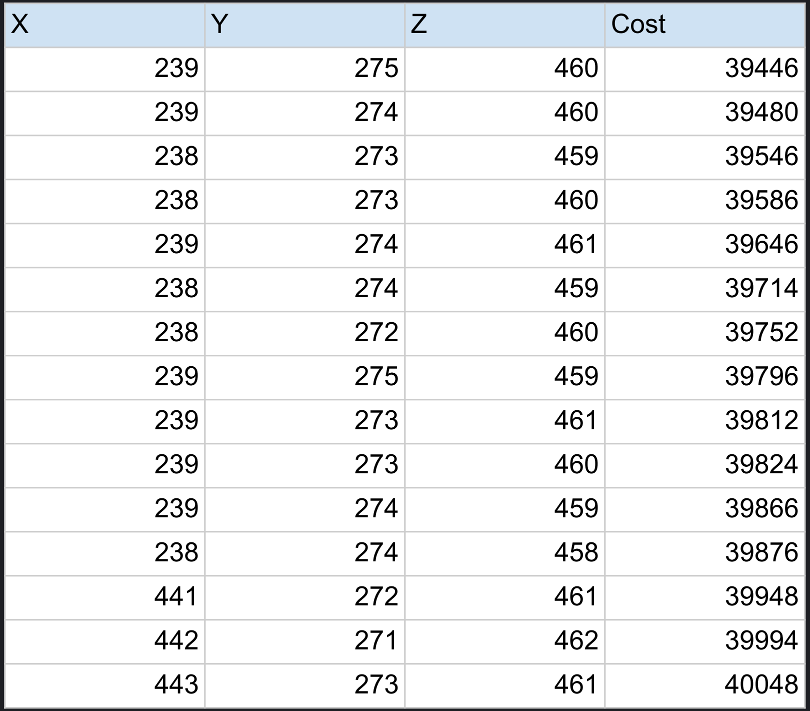
Name	Status	43% CPU	46% Memory	1% Disk	0% Network
Apps (5)					
Discord (32 bit) (6)		0%	257.7 MB	0 MB/s	0 Mbps
Snipping Tool		0.1%	25.8 MB	0.1 MB/s	0 Mbps
Task Manager		0.1%	62.9 MB	0.1 MB/s	0 Mbps
Terminal (10)		0%	135.0 MB	0 MB/s	0 Mbps
Visual Studio Code (10)		0%	1,317.8 MB	0 MB/s	0 Mbps
Background processes (121)					
AMD Crash Defender Service		0%	0.4 MB	0 MB/s	0 Mbps
AMD External Events Client M...		0%	2.0 MB	0 MB/s	0 Mbps
AMD External Events Service ...		0%	0.3 MB	0 MB/s	0 Mbps
Application Frame Host		0%	1.2 MB	0 MB/s	0 Mbps
ASUS App Service		0%	0.6 MB	0 MB/s	0 Mbps
ASUS Dial		0%	8.6 MB	0 MB/s	0 Mbps
ASUS Dial Agent		0%	5.2 MB	0 MB/s	0 Mbps
ASUS Dial Service		0%	2.6 MB	0 MB/s	0 Mbps

Memory used throughout process

003-1040559   1250 003-77156.8   1760 0009-14563.7   73273

3. Trimming Locations

All octaves saved into their own .csv files so they were all merged in to a major file called “Caves.csv” which was then sorted from lowest cost to highest



X	Y	Z	Cost
239	275	460	39446
239	274	460	39480
238	273	459	39546
238	273	460	39586
239	274	461	39646
238	274	459	39714
238	272	460	39752
239	275	459	39796
239	273	461	39812
239	273	460	39824
239	274	459	39866
238	274	458	39876
441	272	461	39948
442	271	462	39994
443	273	461	40048

Example of top values

4. Replacing Dirt

```
def preserveDirt(cave, x, y, z, missing, fill_value = 1, size = 41, wall = 5,
buff = 5, replace = 0):
     if missing < 0:
          fill_value = 0
          missing = abs(missing)
          replace = 1
     seed = [x - size - wall, y - size - wall, z - size - wall]
     while cave[*seed] != replace:
        seed[0] += 1
     queue = deque([tuple(seed)])
     filled = np.zeros_like(cave, dtype = bool)
     filled[seed] = True

     while queue and missing > 0:
          i, j, k = queue.popleft()

          if cave[i, j, k] == fill_value:
               continue
          cave[i, j, k] = fill_value
          missing -= 1

          neighbors = [(i + 1, j, k), (i, j + 1, k)]

          for nx, ny, nz in neighbors:
            if 0 <= nx < cave.shape[0] and 0 <= ny < cave.shape[1] and 0 <= nz
< cave.shape[2]:
                 if not filled[nx, ny, nz] and cave[nx, ny, nz] == replace:
                    queue.append((nx, ny, nz))
                    filled[nx, ny, nz] = True

     return missing
```

- Because the assignment required the amount of dirt at the start and end to be the same, a floodfill algorithm was implemented to replace missing dirt of fill extra air

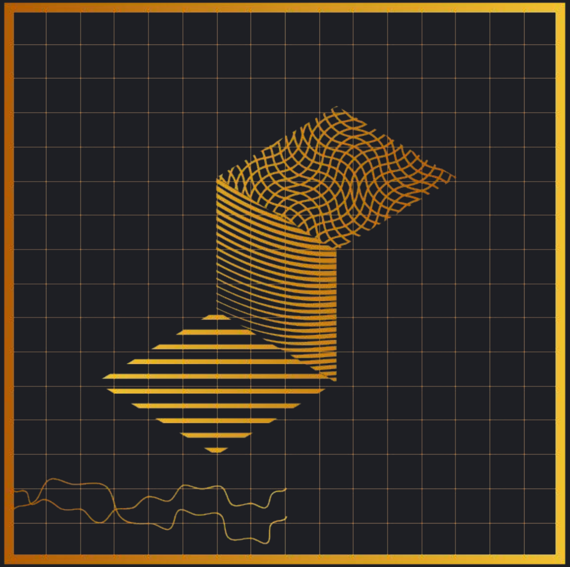
T

3

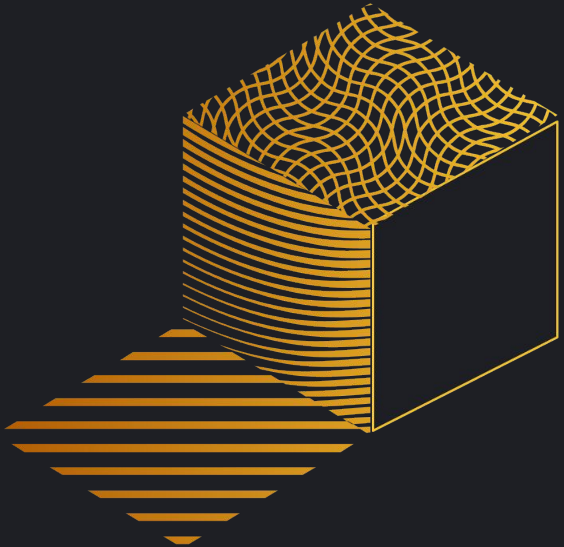
03.

Results

Where were caves made



- Caves were built at:
  - 239, 275, 460, $39446
  - 210, 061, 247, $40406
  - 441, 476, 163, $40214
  - 444, 273, 462, $40484
  - 143, 266, 262, $41282
- Though these weren’t all the best locations, the actual best coordinates overlapped a lot so I manually chose these out of the top 125



Advancements

Processes to implement with more time

Improvements to be Made

Cost Estimator

- Using numpy’s count_nonzerocommand is twice as fast as the .sum() command and easily could halved the dime of operations
- Earlier iterations of of the program began so skip cells if twere was a string of bad terms and this could’ve been implemented in this current iterations to save some time
- Learning how to use numpy’s cumsum features is apparently able to further cut the run time in three

Multiprocessing

- In this project, I just ran multiple variations of the program to “multi process” it. Using python’s multiprocess import could allow for more dynamic improvements in processing speed and ultimately a great result with more control