Martian Squirrel Cave Dwellers

by

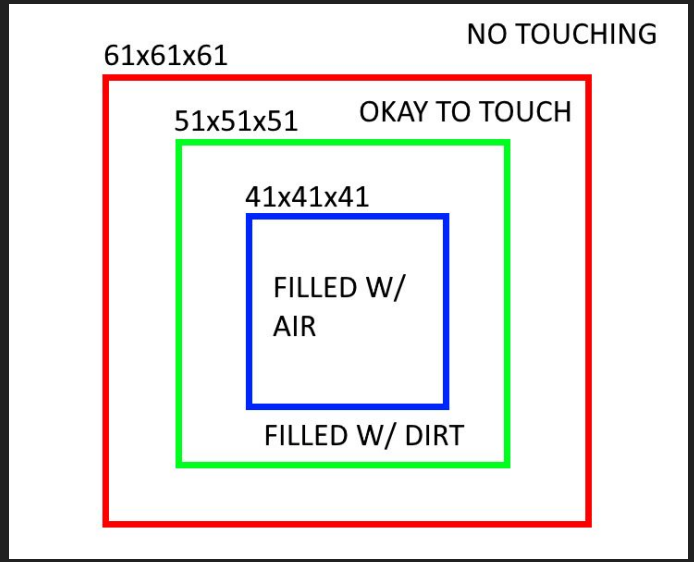
Brent Orlina

Computer Programming III

Nicholas Seward

Assignment Goals

- Burrows must be created in a 512x512x512 cave filled with air and dirt.
- Create a four burrows centered on some chosen coordinates
- Burrows cannot overlap and must fully be within the cave.
- A burrow consists of a 41x41x41 space filled with air and an outer wall that is 5 dirt blocks thick.
- Dirt outside of the 61x61x61 space centered on the burrow cannot be touched.
- The cost of the burrow is calculated by the number of differences between the original cave and the cave with the burrow. The cost must be under 80,000 differences.
- No dirt can be floating. Dirt must be attached to another dirt.



Filtering Coordinates

Filtering coordinates is necessary to not have to iterate through every single coordinate of the cave.

A coordinate to center a burrow on is valid if:

- The burrow will be within the cave centered on that coordinate.
- The coordinate is not filled with dirt.

The reason for the second condition is that burrows will typically be lower cost if centered on a coordinate that is air.

```
coordinates = [
  (z,y,x)
  for z, S in enumerate(A[25:512-26], 25)
  for y, R in enumerate(S[25:512-26], 26)
  for x, C in enumerate(R[25:512-25], 26)
  if not C
]
```

Cost Function

The goal in building the burrow is simply moving dirt from the inner 41x41x41 space into the empty space of the outer wall.

If there isn't enough dirt in the inner space, the coordinate is simply skipped. This is to lessen the complexity of the problem.

The cost function can then simply be calculated by counting the number of dirt in the inner space and multiplying it by 2.

```python
def get_cost(A, C):
    z, y, x = C

    # amount of dirt in inner space
    B_VOL = A[z-20:z+21,y-20:y+21,x-20:x+21].sum()

    # inverting inner space + wall to get amount of air later
    T = (1-A[z-25:z+26,y-25:y+26,x-25:x+26])

    # amount of air in the wall
    # (inner space + wall)  - inner space
    W_EMP = T.sum() - (T[5:46,5:46,5:46]).sum()

    # (validity, cost, coordinate, dirt leftover in the inner space)
    return (B_VOL >= W_EMP, 2*B_VOL, C, int(B_VOL)-int(W_EMP))
```

Multiprocessing

To speed up the process of finding costs for each coordinate, Python's multiprocessing library was used with 24 total processes.

10, 16, and 24 processes were used to experiment with how much faster the calculations could go. 24 processes was consistently the fastest.

```
import multiprocessing as mp
pool = mp.Pool(processes=24)
N = [
  (cost, coord, leftover)
  for V, cost, coord, leftover in pool.map(partial(get_cost, A), coordinates)
  if V and leftover >= 0
]
```

Checking Overlaps

Once a list of valid coordinates, along with their cost, are calculated, the list is sorted from least to greatest by cost.

The coordinate with the least cost is put into a final list of coordinates.

The list of valid coordinates is then iterated through, and if it does not overlap any of the coordinates in the final list, it is added to the final list.

This continues until there are 4 coordinates in the final list.

```
list_of_coordinates = sorted(N, key=lambda n: n[0])
final_list = [N[0]]
for n in list_of_coordinates[1:]:
  if len(valids) == 4:
    break
    _, ncoord, _ = n

    good = True
    for v in valids:
      _, vcoord, _ = v
      if (vcoord[0]-25<=ncoord[0]-25<=vcoord[0]+25 or
          vcoord[0]-25<=ncoord[0]+25<=vcoord[0]+25 or
          vcoord[1]-25<=ncoord[1]-25<=vcoord[1]+25 or
          vcoord[1]-25<=ncoord[1]+25<=vcoord[1]+25 or
          vcoord[2]-25<=ncoord[2]-25<=vcoord[2]+25 or
          vcoord[2]-25<=ncoord[2]+25<=vcoord[2]+25):
        good = False
        break

    if good:
      final_list.append(n)
```

Creating Burrows

To create burrows, the entire inner space and wall are filled with dirt.

Then, the inner space are filled with air.

The leftover dirt are simply put straight down the z-axis on one of the corners of the burrow at the bottom of the wall

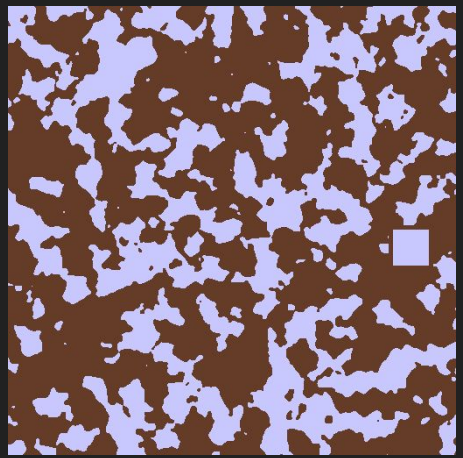
If it hits a dirt from the cave, it moves over and starts again from the bottom of the wall.

```
def make_burrow(cave, coord, leftover):
    z, y, x = coord
    L = leftover
    cave[z-25:z+26,y-25:y+26,x-25:x+26] = 1
    cave[z-20:z+21,y-20:y+21,x-20:x+21] = 0

    for x in range(x-25,x+26):
        for y in range(y-25,y+26):
            for z in range(z+26,z+26+L):
                if z >= 512:
                    break
                if A[z,y,x]:
                    break
                if not leftover:
                    break

                cave[z,y,x] = 1
                leftover -= 1

            if not leftover:
                break
        if not leftover:
            break

    return cave
```

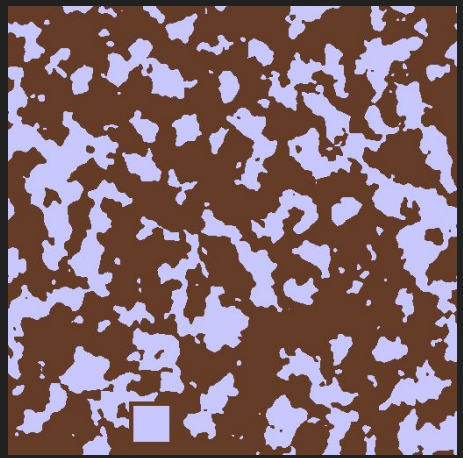
Results

Each image shows a slice of the cave dwelling from the top at the center of dwelling.

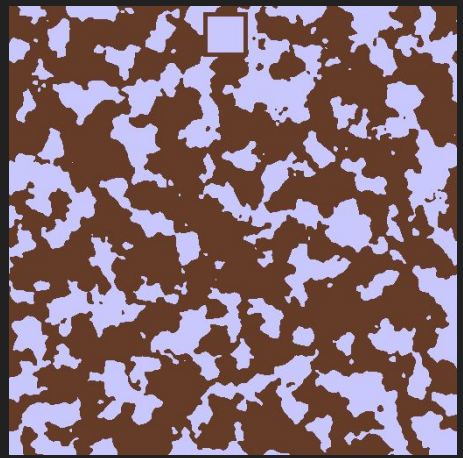
(460, 275, 239)
39446 differences



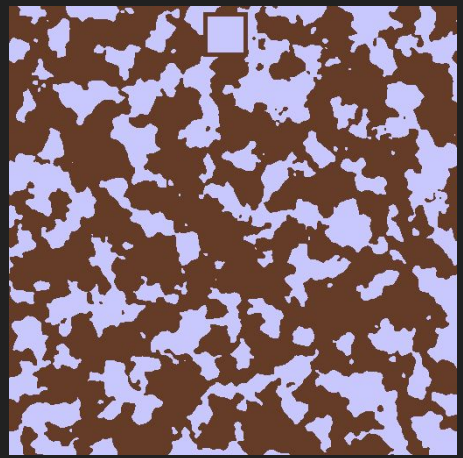
(163, 476, 440)
40236 differences



(247, 32, 365)
40370 differences



(323, 132, 117)
43244 differences



Future Works

A bottleneck of the program is the cost function. It can be improved by using cumulative sum to pre-calculate the amount of dirt or air in an area from the origin. This can be used to calculate the amount of dirt or air in a certain area.

Iterating through most of the air coordinates is also inefficient. This can be improved by using gradient descent in each area of empty air so that it doesn’t have to iterate through air coordinates with worse costs.