Name: Daniel Ems
Date: 03/25/2017
Current Module: Python Capstone
Project Name: Mining

Project Goals:

The goal of this project was to collect minerals and return them back to the Overlord. In order to do so, drones must navigate obstacles on a map of an unknown size. Finally, in order for the minerals to be counted as collected, the drones must return back to the landing zone before the given number of ticks runs out.

Considerations:

- Identifying the optimal search algorithm
  - spiral
  - grid
  - strip
- Determining how to circumnavigate obstacles
  - Counting left and right turns
  - wall hugging
  - coordinate based
- Determining how to collect minerals
  - one drone identifies minerals, and the other collects
  - collect as you move
- Calculating the return back to the landing zone
  - relative to the drone’s current position
  - relative to the furthest corner of the map
- Calculating the optimal number of ticks to recall your drones
  - select an arbitrary number
  - using a relative distance
- Determining how to handle multiple drones deployed to the same map
  - implement the same search pattern in separate areas
  - implement separate search patterns in the same are
- Identifying how you will keep track of what drones are deployed to which map
  - keeping a dictionary with the Overlord
  - storing the information in the drones
- Determining what the optimal use of your six drones will be
  - divide them evenly among the number of maps
  - modify the number of drones to a map based on its mineral density

Initial Design:

The programs structure is built off of two classes, Drone and Overlord. The Overlord class controls instantiating and deploying drones, storing maps, keeping track of the return distance to ticker ratio, and recalling the drones. The Overlord is responsible for the prior because it is the only component of the program that has access to all other aspects. This is important because it centrally locates logic and methods that are to be affect the rest of the program.

The Drone class has two responsibilities, collect minerals, and navigate obstacles to various locations. Each drone has access to its immediate vicinity in all four cardinal directions, in addition to the coordinate it occupies. This makes the responsibilities given to the drone optimal. The logic the drones implement is relative to their unique position on any given map.

The initial design of the program incorporated a third class, GPS. This would handle the responsibilities of rerouting drones if they encountered obstacles, storing their landing zone, and providing recall instructions. The logic needed to avoid obstacles is constant amongst all of the drones. It is the result of the logic that is relative to each drones unique position. However unfortunately, there was not enough time to incorporate this class so the respective methods remain with the drones.

Data Flow:
The program runs based off of ticks. Each tick the overlord is allowed to perform one task. Each drone is also allowed to make one move per tick.

1. The program starts with the Overlord Class
   1. overlord calls its add map method to receive map ids and summaries
   2. the overlord instantiates 6 drones
   3. the drones are deployed to the various maps
2. The drone touches down on the landing zone
   1. the drone will set their landing coordinates
   2. the drone will then calculate if the southern, or western border is closest, and then set that as its direction.
   3. the drone will then call it’s navigate method where it will determine if there is an obstacle immediately in front of it. If so, it will reset its direction until it finds an open coordinate
   4. the drone moves in its final directions
3. The overlord will call its farthest method distance method while the drones are on their border search
   1. the method will receive the drones updated northern, easter, southern, and western borders.
   2. The method will calculate the distance from the landing zone to each corner.
   3. The method will store the update the largest value in a dictionary associated with the drone.
4. The drone call mineCheck
   1. mineCheck returns the values of the coordinates immediately surrounding the drone
   2. if minerals are present, the drone will mine them
5. the overlord checks the number of ticks, to current return distance ratio
   1. if the number of ticks falls below the return distance, the overlord sets the drones return value to true
6. the drone will stay its path collecting minerals until it encounters an obstacles
   1. once an obstacle is encountered, the drone moves right and the right turn counter is incremented by one
7. the drone will check to see if its right counter is greater than its left counters
   1. if so, the drone will hug what ever is to its left
   2. if there is nothing to the drones left, then the drone will make a left turn and the left turn counter is incremented by one
8. if the drones left turn counter exceeds the right turn counter, the drone turns right and the counters are reset.
9. While the drones circle the border of the map, they update the northern and eastern border.
   1. If the current x or y coordinate is greater than a previously stored value, the value is updated.
10. The drone will circle the map until it arrives at its starting point
    1. once the drone returns to the starting point they begin their inward spiral search
11. the drone will utilize the same logic as the border search, but will use the borders it identified to spiral inward.
    1. If the current x or y respective to the direction is with in 3 of the per-discovered borders, the drone turns right and updates that border based on

its current locations.

12. When the number of ticks falls below each drones return distance, their return values are set to true by the overlord.
    1. Each drone calls their return instructions method which calls calculates their current distance from the landing zone, and populates the information in a deque.
13. While returning the drones disregard hazards and avoid mineral patches.

Potential Pitfalls:

Undoubtedly, the most notable pitfall was circumnavigating obstacles. There are countless searching algorithms that offer solutions to this problem, however almost all are incomplete. The method adopted was a left and right counter (explained above). This method offered the most accurate solution to navigating most all obstacles in any scenario. The only exception was if a drone encountered another drone. This created a slue of issues that resulted in only implementing one drone per map. However, if given more time the logic could have been worked out.

The issue arose from the wall hugging component of the obstacle avoidance logic. If a drone encountered another drone they both move to avoid the obstacle. They would then struggle to find a wall to hug since their obstacles were not stationary. The efficiency of one drone that completed a reliable search each time, superseded the possibility of two drone’s colliding and throwing off their search patter.

Test Plan:

User Test:

1. on a 8 by 8 map deploy one drone to implement a search algorithm
2. hard code coordinates just to accomplish a basic search
3. once this was achieved, implement a method to check your surroundings for minerals
4. once this was achieved attempt to incorporate collecting minerals into the search.
5. Once the drones were able to search and collect minerals, next was identifying how to dodge obstacles.
6. This was done by simply avoiding one obstacle at a time and attempting to recreate the affect for multiple minerals
7. Test to see how the drone can be recalled
8. test to see what the best case for recall will be

Test Cases:

1. test one drone on an 8 by 8 with 1000 ticks and no recall to see how long it would survive
2. implement mining and repeat step one
3. implement obstacle avoidance and repeat test one
4. implement recall and repeat test one.
5. Implement a mineral max count to instantiate a recall at various different times.
6. Reduce the number of ticks on an 8 by 8 to 100 and calculate the number of minerals returned.
7. Repeat steps 1 through 6 on map sizes 10,8 / 15,15 / 20,20 / 20,40 with a tick count of 1000
8. Try and kill the drones by removing logic from obstacle avoidance to determine where things can be optimized.
9. Implement left and right turn counters to simply navigate around obstacles, not concerned with staying on track
10. attempt to have drone encounter multiple obstacles,

circumnavigate them, and continue on the same axis

11. implement the same tests on map sizes from 7
12. reduce the number of ticks and calculate the number of minerals collected when the ticks hit 0.
13. run at least ten check cases as the logic becomes more mature to try and identify any bugs that may reside. If bugs reside, fix and re implement
14. implement a recall method and with the map sizes from 7, implement various tick counts 1000, 500, 300, 100, 50.
15. Identify if the drone's struggle to return in any of these scenarios.
16. If so, fix and reimplement.

Conclusion:

It was both frustrating and exciting to work with searching algorithms one thing I believe went well was the implementation of the left right logic. This served very reliable in navigating around obstacles. It also lent a hand into developing a search algorithm that was solely dependent on the map, and not outside factors.

One thing I believe was a problem, and did not work well was working with multiple drones. It would seem the logic should be similar, however, optimizing two search patterns, be it the same or different was difficult. I would have been more pleased to be able to implement two drones simultaneously to optimize my search pattern.