Phase Two Capstone Writeup

Name: Joshua Abraham

Date: 21 SEP 2017

Current Module: Phase Two Capstone

Project Name: "Mining"

Project Goals:

This project was quite large and involved many modules, classes, and unit tests to create a package that implements an Overlord and Drones to mine minerals from a map. The Overlord and it's drones operate in a simulation that uses 'ticks' to denote discrete units of time. The simulation begins with a limited number of 'refined minerals' which are used to spawn Drones, who scout and mine the map.

Considerations:

- The project should make use of concepts learned during phase two.
- The 'mining' package must instantiate an Overlord and at least two subclasses of Drone.
- All Zerg units must have health (minimum of 1) and an action method that takes a map context as a parameter.
- All code in the package must follow the PEP8 guidelines.
- No work should be performed in the 'master' branch.
- The Overlord class has 1 second to perform it's action method, while the Drones have 1 millisecond.

Initial Design:

The project is a Python package, composed of several modules and tests:

· *area.py*: This module contains the Area class that is used by drones to represent their view of their map.

· *dashboard.py*: This module contains the Dashboard class that represents all three maps in the simulation.

· *overlord.py*: This module contains the Overlord class that is responsible for creating, deploying, and returning Drones. The Overlord also commands Drones to mine minerals.

· *drone/*: This directory contains the drone and location modules.

· *drone.py*: This file contains the Drone, Scout, and Miner classes of drone units.

· *location.py*: This file contains the Location class, which is used by drones to store their current and adjacent tiles.

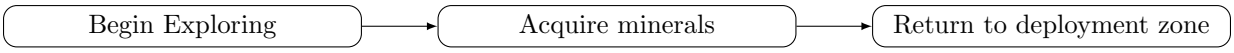
· *path.py*: This file contains functions used by the Overlord and Drone classes during pathfinding.

· *zerg.py*: This file contains the abstract class Zerg from which Drones and Overlord inherit from.

· *tests/*: This directory contains unit tests for the package.

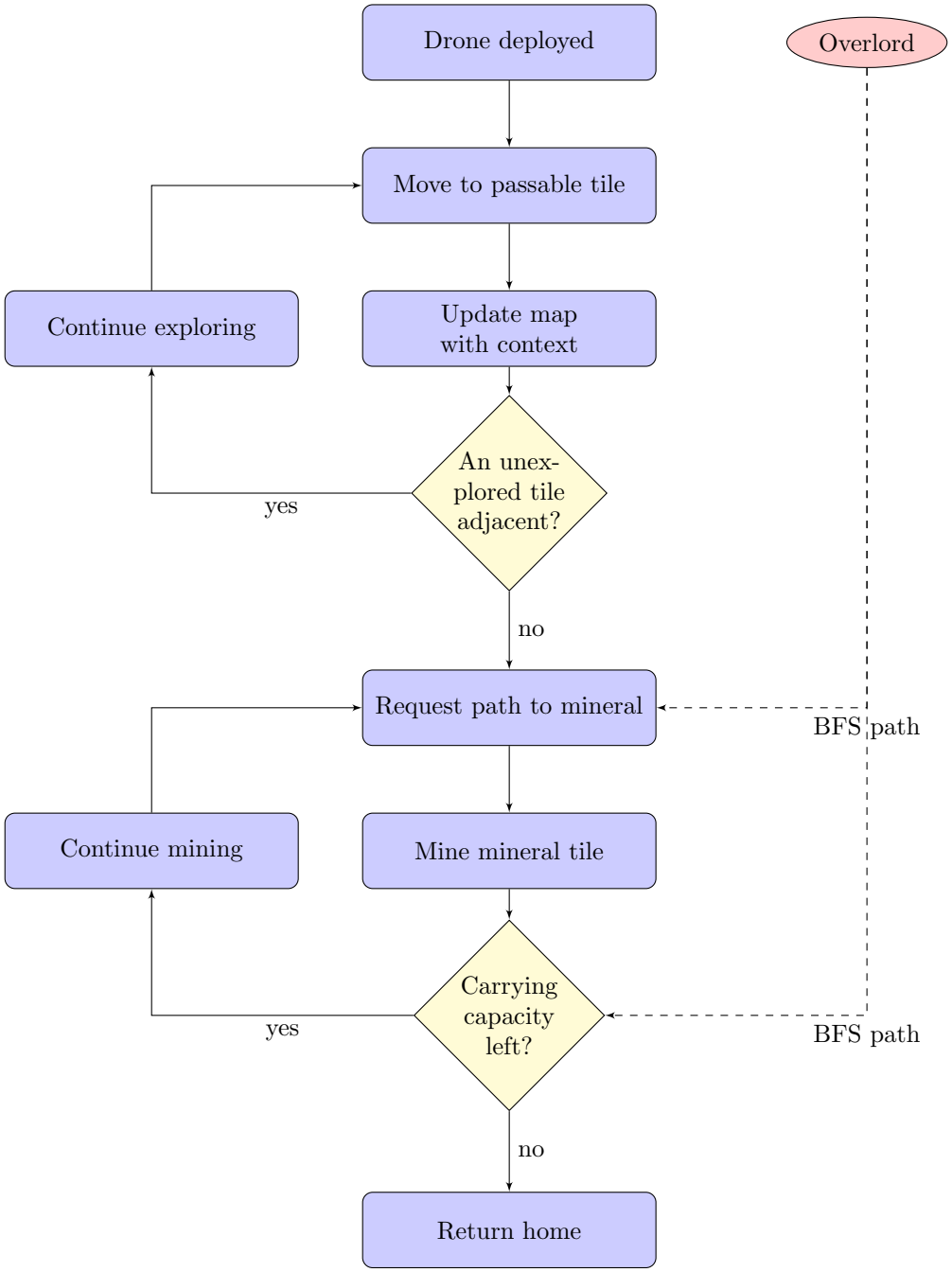
· *runtest.sh + runlinter.sh*: These are bash scripts to run unit tests and the pep8 linter utility.

Data Flow:



(1)

The Overlord spawns the Drones that it can afford with it's refined minerals. These Drones are then deployed to each map and begin exploring. They follow the basic steps in figure 1.



(2)

Drones explore by following a simple set of rules: if an adjacent tile is both unexplored and passable, it will move there. Each time a Drone moves it updates it's internal representation of the map. When the drone is finished exploring it sets a flag that signals it is ready to mine. The next steps are shown in figure 2. The Overlord polls every deployed Drone and monitors the status of their flags. When a Drone signals it is ready to mine, the Overlord converts the Drone's map into an adjacency list and performs a breadth-first-search to generate a path to a mineral field to mine. The Overlord selects this field with a naive linear distance calculation:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

The Overlord gives this path to the Drone, who carries them out. This process repeats until the Drone's carrying capacity is depleted. The Drone then generates a path back to the deployment zone and returns home. The Drone sets it's returning flag and when the Overlord sees this flag, it returns the unit.

Communications Protocol:

The Overlord and Drones "communicated" via setting attributes. No other communications were used in this project.

Potential Pitfalls:

- Having too much logic in the Drone and causing timeouts.
- Deeply nested if..else statements for path decision making. (Especially for readability)
- Conforming to PEP8.
- Pathfinding algorithm running in time constraints. (1s for the Overlord, and 1ms for the Drones)

Test Plan:

User Tests:

· Use the started code provided by instructor to "drive" package.

· Run the *world.py* starter file with various values for ticks, and refined minerals.

· Run *world.py* with user defined maps.

Test Cases:

· Unit tests for every class and method. (where feasibly posible)

· Test inter-object communication.

· Test path finding with a given map and goal.

Conclusion:

The 'mining' capstone has been both interesting and challenging. I learned many lessons while designing and creating the package. This was my first time using Python's abstract base class, and I used it to create the Zerg base class because 1. It will never be instantiated and 2. Both overlords and drones must implement certain shared methods and attributes so it made sense to create an abstract class from which they both would inherit. I also learned a great deal about inheritance versus composition. The Area class originally was a child class of 'dict', but I wanted more control over what I was inheriting so I switched the implementation to contain a dictionary. This project also allowed me to gain a deeper understanding of

'dunder' methods. For example, I used '__contains__()' in my Area class so that I could use the 'in' keyword to determine whether a certain tile had been discovered in a given Area. I also made use of Test Driven Development, and wrote unit tests before starting to implement a given requirement. This was an immense aid both during the initial implementation and while experimenting later on. Lastly, using the '@property' and setters/getters where needed made monitoring attribute values very simple.

There are aspects of the project that I would like to improve on if I had more time. Instead of using a breadth-first-search to generate paths, I would like to implement A*. This would be an improvement because I could assign a cost to moving over an acid tile, instead of not allowing moves over acid. I would also like to improve my Drone exploration logic. With A* I would find a path to unexplored parts of the map.

This capstone was a challenge to implement but I was able to do so successfully. I enjoyed researching various path-finding algorithms and am happy with the choices I made. I learned quite a bit about python classes, unit testing, and data structures while designing this project.