Zerg Mining Expedition

The Zerg, an uplifted race of the Xel'Naga, need minerals to grow new Zerg, required for hardening their carapaces and developing strong teeth. These minerals are harvested by a Drone, small Zerg that are capable of being carried by a type of flying Zerg known as an Overlord.

In this project you will be required to write a `mining` package that instantiates the following classes and their parents:

- `class Overlord(Zerg)` - Responsible for the instantiation and deployment of Drones and collection of their resources. It is also responsible for the visual tracking of the movements of the Drones and their surroundings.
- `class Drone(Zerg)` - Responsible for the exploration and excavation of map resources.
- At least two specialized subclasses of Drone, of your own design using the point system specified in the Drone Descsription section.
- `class Dashboard()` - Responsible for the visualization of the Drones movements and their surroundings.

Be aware that all types of `Zerg` possess at least two things: A health with a minimum value of 1, and an action that takes a map context as a parameter.

The simulation will move in *ticks*, where each tick of the simulation allows for one action from each Zerg (the Overlord and the Drones). The ticks are discrete units of time. A given run of the program will involve a variable number of ticks and refined_minerals. The ticks argument to your `Overlord` object will be the number of ticks for that run of the program.

The main goal is for the Drones to harvest the most minerals from three maps and return them to the Overlord.

Requirements

- The name of your repo must be `mining`. The following set of instructions **must** work for your solution to be graded.
- All GUI components will use the `tkinter` module and all windows that are used within the program must be non-modal.
- The pep8 utility will be run against your code to determine how strictly it adheres to the pep8 guidelines.
- All code will adhere to the standard Python naming conventions of pep8.
- You will be evaluated on proper use of version control.
- No aspect of this project should be worked on within the master branch of the repo.

Failure to adhere to any requirement will result in a loss of points in the correctness category of the rubric.

Overlord Description

```
$ git clone git@github:yourgithubname/mining.git
$ python3
>>> from mining import Overlord
>>> o = Overlord(ticks=100, refined_minerals=54)
```

The `Overlord` must use a two argument constructor as shown above.
The `ticks` argument has already been explained above. The `refined_minerals` argument represents the total amount of resources available to the Overlord for the instantiation of the Drones. (All Drones must be instantiated during the instantiation of the Overlord, and Drones cannot be repaired at any point in their lifetime).

The `Overlord` must also expose the following methods:

`add_map(map_id, summary)`
: Registers an identifier for a map with the Overlord, along with a summary of the map. The summary is a measure of the density of minerals on the map (a `float`), which may be useful for deciding which maps to prioritize.

`action(context)`
: The action for the Overlord to take (see below). This method must take a map context object (described below) that is currently not utilized by the Overlord but may be used in the future. The action also needs to return a value, representing the action taken, within 1 second, otherwise the Overlord will take no action for the given tick.

As the drones are instantiated the corresponding amount of `refined_minerals` required by the Drone must be subtracted from the Overlord's refined_minerals.

Additionally, an instance of `Overlord` must expose a public member variable `zerg`, which is a dictionary of the Drones available. The key-value relationship will be the `id()` of the Drone for the key, and the Drone object itself for the value. The Overlord **may** remove Drones that die.

Overlord Actions

Each tick of the simulation, the Overlord will take one action before any Drone action. It may choose to retrieve a deployed Drone, drop a carried Drone into one of the maps, or simply return a string of "NONE" if it performs no action.

To deploy a Drone, the Overlord's action must return a string of the (example) form `'DEPLOY 140322512297264 2'`, where `140322512297264` is the `id` of the Drone, and `2` is the `id` of the map. If that Drone is currently carried by the Overlord, and the deployment zone for that map is clear (not occupied by another Drone), then the carried Drone will now be deployed to the map in the deployment zone.

To retrieve a Drone, the Drone must be on a deployment zone. The Overlord's action must return a string of the (example) form `'RETURN 140322512296984'`, where `140322512296984` is the `id` of the Drone. When it does so, any minerals carried by the Drone will be scored, and the Drone becomes available for redeployment on the next tick.

Drone Description

The basic `Drone` has the following three integer attributes: `health`, `capacity`, and `moves`.

`health`
: A Drone's health is specified when it is created and will never increase or be restored. Moving into a wall or through acid will reduce a Drone's health (discussed in the Map Description section). A Drone dies when its health becomes 0. If a Drone dies, all minerals it was carrying disintegrate completely from the map.

`capacity`
: This attribute represents the maximum capacity of minerals a Drone can carry at one time, though it does not limit the total number of minerals a Drone can collect over time.

`moves`
: This is the total number of moves the Drone can take within a single tick.

- The Drone needs to expose all 3 of the attributes by name as given, (`health`, `capacity`, and `moves`).
- None of the three attributes named above for a Drone are allowed to be initialized to zero.
- The actual values assigned to these attributes are defined by the type of Drone.

- The upper limit for each of these values is limited by the total number of refined_minerals within the Overlord and the number of Drones created (More Drones mean less refined_minerals available per drone to distribute amongst the three attributes).
- 1 refined_mineral from the Overlord is required for 10 units of health in a Drone.
- 1 refined_mineral from the Overlord is required for 5 units of capacity in a Drone.
- 3 refined_minerals from the Overlord are required for 1 unit of moves in a Drone.

The basic Drone is required to have health of 40, capacity of 10, and moves of 1. This requires 9 refined_minerals to produce the above values for the above attributes of the basic Drone

The Drone objects are required to expose the following methods:

`action(context)`
: This method receives a map context object (see below), and must return the direction that the Drone desires to move in, `'NORTH'`, `'SOUTH'`, `'EAST'`, or `'WEST'`. Any other value will result in the Drone staying where it is for the given tick. This method must return a value within 1 millisecond, otherwise the Drone will stay where it is for the given tick.

`steps()`
: This method must return the cumulative steps taken by the Drone up to the point the method is called.

The Drone class must define the following class method as opposed to an instance method:

`get_init_cost()`
: This method will return the number of refined_minerals it takes to initalize the drone. The Overlord will be able to use this to determine, prior to constructing a Drone, how much it will cost.

Map Description

Any given map may have the following features:

`#` A wall.
: These are indestructible and cannot be moved through. Any Drone attempting to walk into a wall will suffer 1 point of damage against their health.

`_` Deployment zone.
: Drones may only be deployed or returned from these areas on the map. There will **always** be exactly one such tile on any given map.

`*` Minerals.
: These are the goal of the exercise. Any Drone attempting to move into such a square will instead harvest 1 unit of minerals. Eventually such a tile will run out of minerals, becoming an empty space.

`~` Acid.
: These are damaging squares that may be moved through. Any Drone that starts a tick on these squares will suffer 3 points of damage against their health.

Empty Space
: All blank spaces are empty space that may be moved through.

A map will **always** be completely surrounded by walls; there is no way to escape the edges of the map. A map's border is not necessarily square.

Map Context Object

The map context object (received as a parameter to the Drone's `move()` method) contains the following attributes.

`x`
: The x-coordinate of the Drone in the current map. Will be a positive or negative integer.

`y`
: The y-coordinate of the Drone in the current map. Will be a positive or negative integer.

`north`
: The tile to the north of the Drone's position, as a string.

`south`
: The tile to the south of the Drone's position, as a string.

`east`
: The tile to the east of the Drone's position, as a string.

`west`
: The tile to the west of the Drone's position, as a string.

If another Zerg occupies a nearby tile, the string `'Z'` (Upper Case) will be returned instead of the terrain of the tile.

Dashboard Description

The dashboard is required to have the following features:

- A root window that will act as a console.
  - The console is required to, at minimum, display the title of each window with the corresponding id of the map it displays.
- One separate window for each of the 3 maps
  - Each window must display all of the known elements of the map, including the position of each Drone within the map.

The Overlord needs to update the console and the map windows with all of the current information known about each map for each tick.

Suggested Flourishes

- Submit your writeup in TeX or LaTeX.
- Modify the console to display the stats of each Drone and updates them with each tick.
- Enhance the GUI to use images rather than the "#~_*Z " characters to display the map.

Things You May Find Useful

- Graph Paper, for drawing maps and paths
- `unittest`
- Caffeine
- Stopping for the day when you are on a roll, rather than stuck
- Building the map in the Overlord's memory
- Expanding the sample driver program
- Taking a break when you are stuck, rather than on a roll
- Communicating between the Drones and the Overlord

- Not Moving on to a new feature of the project until completing the current one
- Sleep
- Pathfinding in Roguelikes
- Not attempting any extra features until the core requirements are done and checked in to your master branch.
- Keeping track internally of hit points and minerals
- Working on extra features in separate branches until you are satisfied with them.
- Spending the first day reading up on any references and sketching out ideas, before beginning to code.

Rubric

Correct	30%
Architecture	30%
Efficient	20%
Writeup	20%

Final Wisdom

- Remember that your only insight into the map will be through your Drones.
- Keep it DRY, keep it SOLID, and don't talk to your neighbor's neighbor.
- Remember when to optimize.
- You are not your code.

20170824_06