HW8 Supplemental File

Levi Cai

2012-04-16 Mon

Contents

HW8 Supplemental - The Ants API (for HW8 only)
Basic steps for Programming Part Re-iterated:

- Part 1:
  - Fill in get_reward in qlearner.py
  - Fill in "reasonable" values for explore_and_exploit in qlearner.py
    - You can use self.value() which will compute the Q-value for you (see comments for usage)
  - Come up with a reasonable explore or exploit strategy based on self.ngame (Example: if ngame % 10 == 0, then explore)
  - Modify ngames in run_qlearner.py to be smaller for testing purposes
  - run the following:
    - python run_qlearner.py
    - Watch the magic (hopefully)
  - It's okay if you lose a lot, as long as your bot is clearly doing intelligible actions (such as going for food and avoiding enemies, it is usually very clear when this is happening)
- Part 2:
  - Append feature name to BasicFeatures.init_from_dict in src/features.py
  - Append a boolean value calculation to BasicFeatures.extract() that makes sense with the feature name
  - This can be simple, but we encourage interesting ones
  - run the following (run_qlearner.py will automatically generate a new .json file of the write length for you):
    - mv saved_bots saved_bots.bak
    - mkdir saved_bots
    - python run_qlearner.py
- Submission:

- Write up of the answers to the questions, have the name of BOTH MEMBERS at the top of the file and their associated pennkeys
- zip the entire directory such that if we run "python valuebot.py" or "python qlearner.py 100" it should run a single game.

1 Features

There are 3 types of features. Basic, Qualifying, and Composite Features. The human-readable names given to features are arbitrary (you can call them "Bob's awesome new feature" if you liked, but that's not very informative when debugging). Each feature should be a binary value (it has a True or False result) based on the world.

1.1 BasicFeatures

Basic features are stand-alone features,

For example, the following can be a basic feature: feature_name: "2 Enemy Ants Visible" f: True if we can see 2 or more enemy ants, False otherwise

1.2 QualifyingFeatures

Qualifying features are those that you can combine with basic features in composite features (think of them almost as adjectives I guess). These are NOT stand-alone (are NOT included as a separate feature, more on this in the composite features section)

For example, the following might be a qualifying feature: feature_name: "Moving Towards Enemies" f: True if the location resulting from the given action at the current location is closer to the nearest enemy ant. False otherwise.

1.3 CompositeFeatures

Each composite feature is built from a single basic feature combined with a single qualifying feature. For example, if we had the basic features "2 Enemy Ants Visible", "3 Enemy Ants Visible" and the qualifying feature "Moving Towards Enemy", all the composite features that are generated for you are:

- "2 Enemy Ants Visible"
- "3 Enemy Ants Visible"

- "2 Enemy Ants Visible AND Moving Towards Enemy"
- "3 Enemy Ants Visible AND Moving Towards Enemy"

This is why you have nearly 200 features to begin with, even though you only see a few.

1.4 Adding New Features

Let's say you wish to add a new Basic Feature called "Enemy Ant Visible". Here are the steps required:

1.4.1 Append name to feature_names

In BasicFeatures.init_from_dict method, append the name "Group of Enemy Ants Visible" to the END of feature_names. The ordering of the names is extremely important! Since this is how we pair names to the values calculated for them!

```
# add the feature name
this.feature_names.append("Group of Enemy Ants Visible")
```

1.4.2 Append binary feature value to f in extract()

In BasicFeatures.extract method, append the following calculation:

```
# get the list of nearby enemy ant locations
list_of_nearby_enemies = state.lookup_nearby_enemy(loc)

# get the number of enemy ants we see
num_nearby_enemies = len(list_of_nearby_enemies)

# add the boolean value of the feature
# if there is more than 1 ant, then we see a group,
# so we return True, else we return False
f.append( num_nearby_enemies > 1 )
```

And those are the basic steps to creating a new feature. You should NOT use this feature for your homework (if you do, you need to add an additional one as well).

2 The Overall API

This is a list of functions that you can use from within BasicFeatures.extract() and their corresponding behaviours

2.1 Functions for BasicFeatures

2.1.1 self.find_closest(world, loc, list_of_locs)

This function takes in an AntWorld object (which is passed into the extract function), a location, and a list of other locations. Where locations are tuples of the form (x,y).

This function returns the location (x,y) from the list_of_locs that is nearest to the loc given (nearest mean the smallest manhattan distance).

2.2 Functions for GlobalState (passed into BasicFeatures.extract as state)

2.2.1 state.lookup_nearby_food(loc)

Returns a list of food locations nearby of the form [(1,2), (3,4) . . .]

2.2.2 state.lookup_nearby_enemy(loc)

Returns a list of enemy locations

2.2.3 state.lookup_nearby_friendly(loc)

Returns a list of friendly ant locations

2.2.4 state.get_visited(loc)

Returns how many times this location has been visited by a friendly ant

2.3 Functions for AntWorld (passed into BasicFeatures.extract as world)

2.3.1 world.passable(loc)

Returns False if the location (x,y) has water, and True if not

2.3.2 world.unoccupied(loc)

Returns True if it is only land at location (x,y), False if there is food or water or ants

2.3.3 world.manhattan_distance(loc1, loc2)

Returns an integer, the computed manhattan distance between loc1 and loc2

2.3.4 world.euclidean_distance2(loc1, loc2)

Same as above, but the euclidean distance squared

2.3.5 world.sort_by_distance(loc, list_of_locs)

Returns a sorted list of locations by distance from loc (nearest to furthest)

2.3.6 world.toward(loc1, loc2)

Returns a list of direction ('n', 's'...) that brings you closer to loc2 from loc1

2.3.7 world.closest_food(loc)

Returns the location of the closest food to loc

2.3.8 world.closest_enemy(loc)

Returns the location of the closest enemy to loc

2.3.9 world.closest_friend(loc)

Returns the location of the closest friendly ant to loc

2.3.10 world.get_passable_directions(loc, dirs)

Returns passable (see above for definition) directions of the list of dirs that are reachable from loc