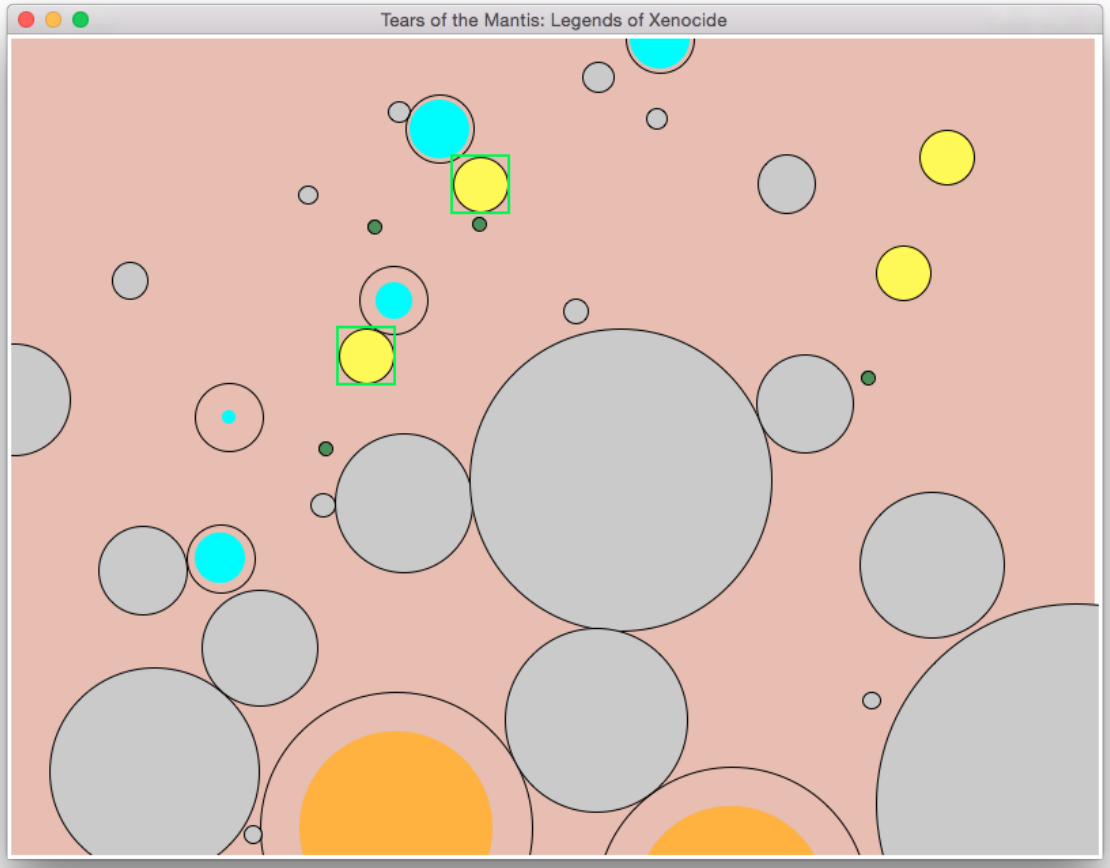
P4: Micro-RTS Unit Controller



Game Overview

Yellow: *Slug*. The user gives slugs commands with the mouse and keyboard. Slugs obey those commands, making local decisions using a state machine. They are relatively big and slow. Left-click-drag to select a group of slugs (or double-left-click to select all slugs). Then, given them an order with a keyboard key or right click a location to issue a basic move order.

Green: *Mantis*. Mantises small and fast, but they mostly ignore slugs unless they bump into one. The player cannot control the mantises, but the state machine code that controls them may be inspected and modified.

Cyan: *Resource.* These are collectable resource points. As you harvest them, they shrink and eventually disappear, opening up new navigation paths.

Orange: *Nest.* Slugs can bring harvested resources back to a nest or they can use a build action to repair the nest.

Use the *Escape* key to quit the game.

Base Code Overview

Find the base code here:
https://drive.google.com/a/ucsc.edu/folderview?id=0B-PPiU3Ga8Z7fnJERkEtUWV1VkRFenZCaWtCRnZTWFlRWkY5WHAzZEhldjNJMzVNOTFMQms&usp=sharing

p4_game.py: This is the main module for this assignment, but you do not need to modify the file. The program takes no special command line arguments. Focus your attention on the next file.

p4_brains.py: This module defines the behavior for slugs and mantises. Use the mantis code as a reference when building your own slug controller. Fill in details for the SlugBrain class.

- At the end of this file, you'll find a dictionary called *world_specification* defined. You can modify the parameters here to simplify your tests.
- Also, you'll find a dictionary called *brain_classes*. If you want to have several different implementations of your slug and mantis brains available, you can modify this definition to pick which one will be used during gameplay.

The GameObject Class

Subclasses of GameObject include Slug, Mantis, Nest, Resource, and Obstacle

Fields

- ***amount***: a float between 0 and 1 that we will use to represent the health of slug or mantis, the repair status of a base, or the remaining resource value of a resource object. If *amount* ever becomes negative, the object will be removed from the simulation.
    - Your brains will periodically modify this field to model combat and repair.
- ***position***: a pair of floats representing the pixel coordinates of the center of this GameObject.
    - Your brains **should not** need this information, but it may help for debugging.
- ***radius***: a single float measuring the radius of the visual representation of the GameObject in pixels
    - Your brains **should not** need this information.

Methods

- ***stop()***: Stop all motion for this GameObject (such as Slug or Mantis).

- ***go_to(target)***: Begin walking to a given pixel location using a flowfield pathfinding system. Once told to go somewhere, the object will keep moving towards the target point without further commands. The target may either be a (x,y) pair of pixel coordinates or a GameObject. If you intent to make contact with another object, pass the object rather than the object's position to make sure the built-in collision-avoidance doesn't stop you from reaching it.
- ***follow(obj)***: Begin immediately approaching the given GameObject (which could be a non-moving object such as a Resource or Nest). Once told to follow, this object will keep moving regardless of intervening obstacles.
- ***set_alarm(dt)***: Schedule a 'timer' event to be dispatched to the brain of this object in approximately dt seconds (a float) of simulation time.
- ***find_nearest(class_name)***: Find the nearest instance of the given class (passed by string name) where distance is reckoned as the approximate navigation path length (going around the current positions of obstacles, not just the euclidean distance).

The Brain Classes

Brain classes (SlugBrain and MantisBrain defined in p4_brains.py) should have a constructor that takes *body* (***__init__(self, body)***) , a GameObject which this brain will be used to control.

Brains should implement a ***handle_event(self, message, details)*** method. During the execution of this method, various methods on the body (a GameObject) should be called to take actions and sense the world.

Events

- **'order', details = (x,y)**
  - When the user right-clicks a map location when some units selected, an event like this will be dispatched.
- **'order', details = some_char**
  - When the user presses a keyboard key when some units are selected, that key will be relayed to the brains of selected units using this event.
  - Suggested keyboard commands for controlling slugs:
    - 'i': idle
    - 'a': attack
    - 'h': harvest
    - 'b': build
- **'timer', details=None**
  - A previously set alarm has fired. The brain should make some decisions and then likely set another alarm to check on the progress of those decisions with an expression like *self.body.set_alarm(1)*.
- **'collide', details={'what': class_name, 'who': obj}**
  - When the collision detection system finds two objects overlapping, it allows the brain of each object to respond to the situation. The 'who' entry of the details

dictionary provides a reference to the other object in the collision. Use this to decrement the other object's *amount* field when modeling combat.

Requirements / Grading Criteria

(As usual, each bulleted item receives equal weight in the overall score.)

- **Implement a move command:**
  - When the user right-clicks a location on the map, the selected slug units should approach (using go_to) that point.
  - The example code implements this behavior. However, you should make sure this behavior is preserved even as you modify the example code.

- **Implement an Idle command:**
  - When the user presses 'i', the selected slug units should stop what they are doing and hold still (use the obj.stop() method to stop the slug).

- **Implement an Attack command:**
  - When the user presses 'a', the selected slug units should enter an attack mode.
  - In this mode, the slug should periodically (every one or two seconds) find the nearest Mantis unit (using find_nearest) and begin following it (using follow).
  - Upon colliding with a mantis in attack mode, the mantis' *amount* should be decremented by 0.05 units per collision event.
  - The logic of this command should allow a single slug to seek and destroy several mantises (not getting stuck after the first one).

- **Implement a Build command:**
  - When the user presses 'b', the selected slug units should enter a build mode.
  - In this mode, the slug should periodically find the nearest nest and approach it (using go_to).
  - When colliding with a Nest in build mode, the *amount* of the nest should be incremented by 0.01 per collision event.
  - It is **not** expected that a slug in build mode will approach the Nest with least amount or move on to another Nest once the nearest has been fully built. Manipulating the builder's focus is the players responsibility for this particular game.

- **Implement a Harvest command:**
  - When the user presses 'h', the selected slug units should enter a harvest mode.
  - In order to correctly implement harvesting behavior, you will need to store some extra state in the brain as to whether the slug carries some resource or not.

(Define an extra field in the SlugBrain constructor. Consider a line like '*self.have_resource = False*'.)
    - If the slug is holding a resource, it should periodically find and approach the nearest Nest. If the slug is not holding a resource, it should periodically find and approach the nearest Resource object.
    - When colliding with a Resource object while not holding a resource, decrement the *amount* of the Resource object by 0.25 (and set your *have_resource* flag accordingly)
    - When colliding with a Nest while holding a resource, simply reset the resource holding flag. (Leave the *amount* of the Nest untouched so that it is clear when it is being changed by a slug in the build mode.)
    - Several slugs should be able to autonomously mine out several Resource objects near a given nest without user intervention. (The Resource objects are destroyed when their amount drops below zero.)

- **Implement automatic Flee behavior:**
    - Whenever slug has low health (*amount* less than 0.5), the slug should immediately start moving to (using either go_to or follow) the nearest Nest. No user intervention should be required to enter flee mode.
    - When colliding with a Nest in Flee mode, the slug's health (*amount)* should be restored (whether it goes to full in one step or several small steps is up to the student to decide).

OPTIONAL Funtimes

After your SlugBrain is working satisfactorily, consider making some fun changes to the MantisBrain implementation.

- By modifying the *radius* and *speed* fields of a mantis, you can make them bigger and stronger after each encounter with a slug.

- Try using using some randomness to occasionally break mantises out of the 'idle' state and into a new 'thievery' state that works opposite the effect of the slug's harvesting activities. Can your mantises steal from the Nest and restore partially-drained Resources?

- By examining the *position* field of the nearest slug, can you create a variation of the mantis behavior where mantises follow but never touch the slugs as they work?