

recording for my ants

```
"""CS 61A presents Ants Vs. SomeBees."""
```

```
import random
from ucb import main, interact, trace
from collections import OrderedDict
```

```
#####
# Core Classes #
#####
```

```
class Place:
```

```
    """A Place holds insects and has an exit to another Place."""
```

```
    is_hive = False
```

```
    def __init__(self, name, exit=None):
```

```
        """Create a Place with the given NAME and EXIT.
```

```
        name -- A string; the name of this Place.
```

```
        exit -- The Place reached by exiting this Place (may be None).
```

```
        """
```

```
        self.name = name
```

```
        self.exit = exit
```

```
        self.bees = []          # A list of Bees
```

```
        self.ant = None        # An Ant
```

```
        self.entrance = None   # A Place
```

```
        # Phase 1: Add an entrance to the exit
```

```
        # BEGIN Problem 2
```

```
        """* YOUR CODE HERE """
```

```
        if self.exit:
```

```
            exit.entrance = self
```

```
        # END Problem 2
```

```
    def add_insect(self, insect):
```

```
        """Asks the insect to add itself to this place. This method exists so
        that it can be overridden in subclasses.
        """
```

```
        insect.add_to(self)
```

```

def remove_insect(self, insect):
    """Asks the insect to remove itself from this place. This method exists so
    that it can be overridden in subclasses.
    """
    insect.remove_from(self)

def __str__(self):
    return self.name

```

```

class Insect:
    """An Insect, the base class of Ant and Bee, has health and a Place."""

    next_id = 0 # Every insect gets a unique id number
    damage = 0
    # ADD CLASS ATTRIBUTES HERE

    def __init__(self, health, place=None):
        """Create an Insect with a health amount and a starting PLACE."""
        self.health = health
        self.place = place

        # assign a unique ID to every insect
        self.id = Insect.next_id
        Insect.next_id += 1

    def reduce_health(self, amount):
        """Reduce health by AMOUNT, and remove the insect from its place if it
        has no health remaining.

        >>> test_insect = Insect(5)
        >>> test_insect.reduce_health(2)
        >>> test_insect.health
        3
        """
        self.health -= amount
        if self.health <= 0:
            self.zero_health_callback()
            self.place.remove_insect(self)

    def action(self, gamestate):
        """The action performed each turn."""

```

```

def zero_health_callback(self):
    """Called when health reaches 0 or below."""

def add_to(self, place):
    self.place = place

def remove_from(self, place):
    self.place = None

def __repr__(self):
    cname = type(self).__name__
    return '{0}({1}, {2})'.format(cname, self.health, self.place)

```

```

class Ant(Insect):
    """An Ant occupies a place and does work for the colony."""

    implemented = False # Only implemented Ant classes should be instantiated
    food_cost = 0
    is_container = False
    # ADD CLASS ATTRIBUTES HERE

    def __init__(self, health=1):
        super().__init__(health)

    def can_contain(self, other):
        return False

    def store_ant(self, other):
        assert False, "{0} cannot contain an ant".format(self)

    def remove_ant(self, other):
        assert False, "{0} cannot contain an ant".format(self)

    def add_to(self, place):
        if place.ant is None:
            place.ant = self
        else:
            # BEGIN Problem 8b
            assert place.ant is None, 'Too many ants in {0}'.format(place)
            # END Problem 8b
        Insect.add_to(self, place)

```

```

def remove_from(self, place):
    if place.ant is self:
        place.ant = None
    elif place.ant is None:
        assert False, '{0} is not in {1}'.format(self, place)
    else:
        place.ant.remove_ant(self)
    Insect.remove_from(self, place)

def double(self):
    """Double this ant's damage, if it has not already been doubled."""
    # BEGIN Problem 12
    """* YOUR CODE HERE """
    # END Problem 12

```

```

class HarvesterAnt(Ant):
    """HarvesterAnt produces 1 additional food per turn for the colony."""

    name = 'Harvester'
    implemented = True
    # OVERRIDE CLASS ATTRIBUTES HERE
    food_cost = 2

    def action(self, gamestate):
        """Produce 1 additional food for the colony.

        gamestate -- The GameState, used to access game state information.
        """
        # BEGIN Problem 1
        """* YOUR CODE HERE """
        gamestate.food = gamestate.food + 1

        # END Problem 1

```

```

class ThrowerAnt(Ant):
    """ThrowerAnt throws a leaf each turn at the nearest Bee in its range."""

    name = 'Thrower'
    implemented = True
    damage = 1

```

```

# ADD/OVERRIDE CLASS ATTRIBUTES HERE
lower_bound = 0
upper_bound = float('inf')
food_cost = 3

def nearest_bee(self):
    """Return the nearest Bee in a Place (that is not the hive) connected to
    the ThrowerAnt's Place by following entrances.

    This method returns None if there is no such Bee (or none in range).
    """
    # BEGIN Problem 3 and 4
    destination = self.place
    count = 0
    while not destination.bees and not destination.is_hive or self.lower_bound > count:
        destination = destination.entrance
        count += 1
    if count > self.upper_bound:
        return None
    elif not destination:
        return None
    elif destination.is_hive:
        return None
    return random_bee(destination.bees)

    # END Problem 3 and 4

def throw_at(self, target):
    """Throw a leaf at the target Bee, reducing its health."""
    if target is not None:
        target.reduce_health(self.damage)

def action(self, gamestate):
    """Throw a leaf at the nearest Bee in range."""
    self.throw_at(self.nearest_bee())

def random_bee(bees):
    """Return a random bee from a list of bees, or return None if bees is empty."""
    assert isinstance(bees, list), \
        "random_bee's argument should be a list but was a %s" % type(bees).__name__
    if bees:
        return random.choice(bees)

```

```
#####  
# Extensions #  
#####
```

```
class ShortThrower(ThrowerAnt):  
    """A ThrowerAnt that only throws leaves at Bees at most 3 places away."""  
  
    name = 'Short'  
    food_cost = 2  
    # OVERRIDE CLASS ATTRIBUTES HERE  
    # BEGIN Problem 4  
    lower_bound = 0  
    upper_bound = 3  
    implemented = True    # Change to True to view in the GUI  
    # END Problem 4
```

```
class LongThrower(ThrowerAnt):  
    """A ThrowerAnt that only throws leaves at Bees at least 5 places away."""  
  
    name = 'Long'  
    food_cost = 2  
    # OVERRIDE CLASS ATTRIBUTES HERE  
    # BEGIN Problem 4  
    lower_bound = 5  
    upper_bound = float('inf')  
    implemented = True    # Change to True to view in the GUI  
    # END Problem 4
```

```
class FireAnt(Ant):  
    """FireAnt cooks any Bee in its Place when it expires."""  
  
    name = 'Fire'  
    damage = 3  
    food_cost = 5  
    # OVERRIDE CLASS ATTRIBUTES HERE  
    # BEGIN Problem 5  
    implemented = True    # Change to True to view in the GUI  
    # END Problem 5
```

```
def __init__(self, health=3):
    """Create an Ant with a HEALTH quantity."""
    super().__init__(health)
```

```
def reduce_health(self, amount):
    """Reduce health by AMOUNT, and remove the FireAnt from its place if it
    has no health remaining.
```

Make sure to reduce the health of each bee in the current place, and apply the additional damage if the fire ant dies.

```
"""
```

```
# BEGIN Problem 5
```

```
**** YOUR CODE HERE ****
```

```
copy = list(self.place.bees)
back_amount = amount
if amount >= self.health:
    back_amount += self.damage
for i in copy:
    i.reduce_health(back_amount)
return super().reduce_health(amount)
# END Problem 5
```

```
# BEGIN Problem 6
```

```
# The WallAnt class
```

```
class WallAnt(Ant):
    name = 'Wall'
    implemented = True
    food_cost = 4
    def __init__(self, health = 4):
        super().__init__(health)
```

```
# END Problem 6
```

```
# BEGIN Problem 7
```

```
# The HungryAnt Class
```

```
class HungryAnt(Ant):
    name = 'Hungry'
    implemented = True
    food_cost = 4
    chewing_turns = 3
    def __init__(self, health=1):
        super().__init__(health)
        self.turns_to_chew = 0
```

```

def action(self, gamestate):
    if self.turns_to_chew == 0:
        aim = random_bee(self.place.bees)
        if aim:
            aim.reduce_health(aim.health)
            self.turns_to_chew = self.chewing_turns
    else:
        self.turns_to_chew -= 1

```

END Problem 7

```

class ContainerAnt(Ant):
    """
    ContainerAnt can share a space with other ants by containing them.
    """
    is_container = True

    def __init__(self, health):
        super().__init__(health)
        self.ant_contained = None

    def can_contain(self, other):
        # BEGIN Problem 8a
        """ YOUR CODE HERE """
        # END Problem 8a

    def store_ant(self, ant):
        # BEGIN Problem 8a
        """ YOUR CODE HERE """
        # END Problem 8a

    def remove_ant(self, ant):
        if self.ant_contained is not ant:
            assert False, "{} does not contain {}".format(self, ant)
        self.ant_contained = None

    def remove_from(self, place):
        # Special handling for container ants
        if place.ant is self:
            # Container was removed. Contained ant should remain in the game
            place.ant = place.ant.ant_contained
            Insect.remove_from(self, place)

```



```

        else:
            # default to normal behavior
            Ant.remove_from(self, place)

def action(self, gamestate):
    # BEGIN Problem 8a
    """ YOUR CODE HERE """
    # END Problem 8a

class BodyguardAnt(ContainerAnt):
    """BodyguardAnt provides protection to other Ants."""

    name = 'Bodyguard'
    food_cost = 4
    # OVERRIDE CLASS ATTRIBUTES HERE
    # BEGIN Problem 8c
    implemented = False # Change to True to view in the GUI
    # END Problem 8c

# BEGIN Problem 9
# The TankAnt class
# END Problem 9

class Water(Place):
    """Water is a place that can only hold waterproof insects."""

    def add_insect(self, insect):
        """Add an Insect to this place. If the insect is not waterproof, reduce
        its health to 0."""
        # BEGIN Problem 10
        """ YOUR CODE HERE """
        # END Problem 10

# BEGIN Problem 11
# The ScubaThrower class
# END Problem 11

class QueenAnt(ThrowerAnt):
    """QueenAnt boosts the damage of all ants behind her."""

```

```

name = 'Queen'
food_cost = 7
# OVERRIDE CLASS ATTRIBUTES HERE
# BEGIN Problem 12
implemented = False # Change to True to view in the GUI
# END Problem 12

def action(self, gamestate):
    """A queen ant throws a leaf, but also doubles the damage of ants
    in her tunnel.
    """
    # BEGIN Problem 12
    """ YOUR CODE HERE """
    # END Problem 12

def reduce_health(self, amount):
    """Reduce health by AMOUNT, and if the QueenAnt has no health
    remaining, signal the end of the game.
    """
    # BEGIN Problem 12
    """ YOUR CODE HERE """
    # END Problem 12

```

```

#####
# Optional #
#####

```

```

class NinjaAnt(Ant):
    """NinjaAnt does not block the path and damages all bees in its place.
    This class is optional.
    """

    name = 'Ninja'
    damage = 1
    food_cost = 5
    # OVERRIDE CLASS ATTRIBUTES HERE
    # BEGIN Problem Optional 1
    implemented = False # Change to True to view in the GUI
    # END Problem Optional 1

```

```
def action(self, gamestate):
    # BEGIN Problem Optional 1
    """ YOUR CODE HERE """
    # END Problem Optional 1
```

```
#####
# Statuses #
#####
```

```
class LaserAnt(ThrowerAnt):
    # This class is optional. Only one test is provided for this class.
```

```
    name = 'Laser'
    food_cost = 10
    # OVERRIDE CLASS ATTRIBUTES HERE
    # BEGIN Problem Optional 2
    implemented = False # Change to True to view in the GUI
    # END Problem Optional 2
```

```
def __init__(self, health=1):
    super().__init__(health)
    self.insects_shot = 0
```

```
def insects_in_front(self):
    # BEGIN Problem Optional 2
    return {}
    # END Problem Optional 2
```

```
def calculate_damage(self, distance):
    # BEGIN Problem Optional 2
    return 0
    # END Problem Optional 2
```

```
def action(self, gamestate):
    insects_and_distances = self.insects_in_front()
    for insect, distance in insects_and_distances.items():
        damage = self.calculate_damage(distance)
        insect.reduce_health(damage)
        if damage:
            self.insects_shot += 1
```

#####

Bees

#####

```
class Bee(Insect):
    """A Bee moves from place to place, following exits and stinging ants."""

    name = 'Bee'
    damage = 1
    is_waterproof = True

    def sting(self, ant):
        """Attack an ANT, reducing its health by 1."""
        ant.reduce_health(self.damage)

    def move_to(self, place):
        """Move from the Bee's current Place to a new PLACE."""
        self.place.remove_insect(self)
        place.add_insect(self)

    def blocked(self):
        """Return True if this Bee cannot advance to the next Place."""
        # Special handling for NinjaAnt
        # BEGIN Problem Optional 1
        return self.place.ant is not None
        # END Problem Optional 1

    def action(self, gamestate):
        """A Bee's action stings the Ant that blocks its exit if it is blocked,
        or moves to the exit of its current place otherwise.

        gamestate -- The GameState, used to access game state information.
        """
        destination = self.place.exit

        if self.blocked():
            self.sting(self.place.ant)
        elif self.health > 0 and destination is not None:
            self.move_to(destination)

    def add_to(self, place):
        place.bees.append(self)
        super().add_to( place)
```

```
def remove_from(self, place):
    place.bees.remove(self)
    super().remove_from(place)
```

```
class Wasp(Bee):
    """Class of Bee that has higher damage."""
    name = 'Wasp'
    damage = 2
```

```
class Boss(Wasp):
    """The leader of the bees. Damage to the boss by any attack is capped.
    """
    name = 'Boss'
    damage_cap = 8

    def reduce_health(self, amount):
        super().reduce_health(min(amount, self.damage_cap))
```

```
class Hive(Place):
    """The Place from which the Bees launch their assault.

    assault_plan -- An AssaultPlan; when & where bees enter the colony.
    """
    is_hive = True

    def __init__(self, assault_plan):
        self.name = 'Hive'
        self.assault_plan = assault_plan
        self.bees = []
        for bee in assault_plan.all_bees():
            self.add_insect(bee)
        # The following attributes are always None for a Hive
        self.entrance = None
        self.ant = None
        self.exit = None

    def strategy(self, gamestate):
        exits = [p for p in gamestate.places.values() if p.entrance is self]
        for bee in self.assault_plan.get(gamestate.time, []):
```

```
bee.move_to(random.choice(exits))
gamestate.active_bees.append(bee)
```

```
#####
# Game Components #
#####
```

```
class GameState:
```

```
    """An ant collective that manages global game state and simulates time.
```

```
    Attributes:
```

```
    time -- elapsed time
```

```
    food -- the colony's available food total
```

```
    places -- A list of all places in the colony (including a Hive)
```

```
    bee_entrances -- A list of places that bees can enter
```

```
    """
```

```
def __init__(self, beehive, ant_types, create_places, dimensions, food=2):
```

```
    """Create an GameState for simulating a game.
```

```
    Arguments:
```

```
    beehive -- a Hive full of bees
```

```
    ant_types -- a list of ant classes
```

```
    create_places -- a function that creates the set of places
```

```
    dimensions -- a pair containing the dimensions of the game layout
```

```
    """
```

```
    self.time = 0
```

```
    self.food = food
```

```
    self.beehive = beehive
```

```
    self.ant_types = OrderedDict((a.name, a) for a in ant_types)
```

```
    self.dimensions = dimensions
```

```
    self.active_bees = []
```

```
    self.configure(beehive, create_places)
```

```
def configure(self, beehive, create_places):
```

```
    """Configure the places in the colony."""
```

```
    self.base = AntHomeBase('Ant Home Base')
```

```
    self.places = OrderedDict()
```

```
    self.bee_entrances = []
```

```
def register_place(place, is_bee_entrance):
```

```
    self.places[place.name] = place
```

```
    if is_bee_entrance:
```

```

        place.entrance = beehive
        self.bee_entrances.append(place)
register_place(self.beehive, False)
create_places(self.base, register_place,
              self.dimensions[0], self.dimensions[1])

def ants_take_actions(self): # Ask ants to take actions
    for ant in self.ants:
        if ant.health > 0:
            ant.action(self)

def bees_take_actions(self, num_bees): # Ask bees to take actions
    for bee in self.active_bees[:]:
        if bee.health > 0:
            bee.action(self)
        if bee.health <= 0:
            num_bees -= 1
            self.active_bees.remove(bee)
    if num_bees == 0: # Check if player won
        raise AntsWinException()
    return num_bees

def simulate(self):
    """Simulate an attack on the ant colony. This is called by the GUI to play the game."""
    num_bees = len(self.bees)
    try:
        while True:
            self.beehive.strategy(self) # Bees invade from hive
            yield None # After yielding, players have time to place ants
            self.ants_take_actions()
            self.time += 1
            yield None # After yielding, wait for throw leaf animation to play, then ask bees
            num_bees = self.bees_take_actions(num_bees)
    except AntsWinException:
        print('All bees are vanquished. You win!')
        yield True
    except AntsLoseException:
        print('The bees reached homebase or the queen ant queen has perished. Please try again')
        yield False

def deploy_ant(self, place_name, ant_type_name):
    """Place an ant if enough food is available.

```

This method is called by the current strategy to deploy ants.

"""

```
ant_type = self.ant_types[ant_type_name]
if ant_type.food_cost > self.food:
    print('Not enough food remains to place ' + ant_type.__name__)
else:
    ant = ant_type()
    self.places[place_name].add_insect(ant)
    self.food -= ant.food_cost
    return ant
```

```
def remove_ant(self, place_name):
    """Remove an Ant from the game."""
    place = self.places[place_name]
    if place.ant is not None:
        place.remove_insect(place.ant)
```

@property

```
def ants(self):
    return [p.ant for p in self.places.values() if p.ant is not None]
```

@property

```
def bees(self):
    return [b for p in self.places.values() for b in p.bees]
```

@property

```
def insects(self):
    return self.ants + self.bees
```

```
def __str__(self):
    status = ' (Food: {0}, Time: {1})'.format(self.food, self.time)
    return str([str(i) for i in self.ants + self.bees]) + status
```

```
class AntHomeBase(Place):
```

```
    """AntHomeBase at the end of the tunnel, where the queen normally resides."""
```

```
def add_insect(self, insect):
    """Add an Insect to this Place.
```

Can't actually add Ants to a AntHomeBase. However, if a Bee attempts to enter the AntHomeBase, a AntsLoseException is raised, signaling the end of a game.


```

        """
        assert isinstance(insect, Bee), 'Cannot add {} to AntHomeBase'
        raise AntsLoseException()

def ants_win():
    """Signal that Ants win."""
    raise AntsWinException()

def ants_lose():
    """Signal that Ants lose."""
    raise AntsLoseException()

def ant_types():
    """Return a list of all implemented Ant classes."""
    all_ant_types = []
    new_types = [Ant]
    while new_types:
        new_types = [t for c in new_types for t in c.__subclasses__()]
        all_ant_types.extend(new_types)
    return [t for t in all_ant_types if t.implemented]

def bee_types():
    """Return a list of all implemented Bee classes."""
    all_bee_types = []
    new_types = [Bee]
    while new_types:
        new_types = [t for c in new_types for t in c.__subclasses__()]
        all_bee_types.extend(new_types)
    return all_bee_types

class GameOverException(Exception):
    """Base game over Exception."""
    pass

class AntsWinException(GameOverException):
    """Exception to signal that the ants win."""
    pass

```

```

class AntsLoseException(GameOverException):
    """Exception to signal that the ants lose."""
    pass

#####
# Layouts #
#####

def wet_layout(queen, register_place, tunnels=3, length=9, moat_frequency=3):
    """Register a mix of wet and and dry places."""
    for tunnel in range(tunnels):
        exit = queen
        for step in range(length):
            if moat_frequency != 0 and (step + 1) % moat_frequency == 0:
                exit = Water('water_{0}_{1}'.format(tunnel, step), exit)
            else:
                exit = Place('tunnel_{0}_{1}'.format(tunnel, step), exit)
        register_place(exit, step == length - 1)

def dry_layout(queen, register_place, tunnels=3, length=9):
    """Register dry tunnels."""
    wet_layout(queen, register_place, tunnels, length, 0)

#####
# Assault Plans #
#####

class AssaultPlan(dict):
    """The Bees' plan of attack for the colony.  Attacks come in timed waves.

    An AssaultPlan is a dictionary from times (int) to waves (list of Bees).

    >>> AssaultPlan().add_wave(4, 2)
    {4: [Bee(3, None), Bee(3, None)]}
    """

    def add_wave(self, bee_type, bee_health, time, count):

```

```
"""Add a wave at time with count Bees that have the specified health."""
bees = [bee_type(bee_health) for _ in range(count)]
self.setdefault(time, []).extend(bees)
return self

def all_bees(self):
    """Place all Bees in the beehive and return the list of Bees."""
    return [bee for wave in self.values() for bee in wave]
```