Ants

● Graded

Group

Seong Jae Ahn
Sangwon Ji
View or edit group

Total Points

27 / 24 pts

Autograder Score

27.0 / 24.0

Autograder Results

```
==================================================================
Assignment: Project 3: Ants Vs. SomeBees
OK, version v1.18.1
==================================================================

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Scoring tests

--------------------------------------------------------------------
Problem 0
Passed: 1
Failed: 0
[ooooooooook] 100.0% passed

--------------------------------------------------------------------
Problem 1
Passed: 1
Failed: 0
[ooooooooook] 100.0% passed

--------------------------------------------------------------------
Problem 2
Passed: 1
Failed: 0
[ooooooooook] 100.0% passed

--------------------------------------------------------------------
Problem 3
Passed: 1
Failed: 0
[ooooooooook] 100.0% passed

--------------------------------------------------------------------
Problem 4
Passed: 3
```

```
Failed: 0
[ooooooooook] 100.0% passed

------------------------------------------------------------
Problem 5
Passed: 2
Failed: 0
[oooooooooook] 100.0% passed

------------------------------------------------------------
Problem 6
Passed: 2
Failed: 0
[oooooooooook] 100.0% passed

------------------------------------------------------------
Problem 7
Passed: 2
Failed: 0
[oooooooooook] 100.0% passed

------------------------------------------------------------
Problem 8a
Passed: 1
Failed: 0
[oooooooooook] 100.0% passed

------------------------------------------------------------
Problem 8b
Passed: 1
Failed: 0
[oooooooooook] 100.0% passed

------------------------------------------------------------
Problem 8c
Passed: 4
Failed: 0
[oooooooooook] 100.0% passed

------------------------------------------------------------
Problem 9
Passed: 3
Failed: 0
[oooooooooook] 100.0% passed

------------------------------------------------------------
Problem 10
Passed: 2
Failed: 0
[oooooooooook] 100.0% passed

------------------------------------------------------------
Problem 11
Passed: 3
```

```
Failed: 0
[ooooooooook] 100.0% passed

---------------------------------------------------------------------
Problem 12
Passed: 3
Failed: 0
[ooooooooook] 100.0% passed

---------------------------------------------------------------------
Problem EC 1
Passed: 1
Failed: 0
[ooooooooook] 100.0% passed

---------------------------------------------------------------------
Problem EC 2
Passed: 1
Failed: 0
[ooooooooook] 100.0% passed

---------------------------------------------------------------------
Point breakdown
Problem 0: 0.0/0
Problem 1: 1.0/1
Problem 2: 1.0/1
Problem 3: 2.0/2
Problem 4: 2.0/2
Problem 5: 3.0/3
Problem 6: 2.0/2
Problem 7: 3.0/3
Problem 8a: 1.0/1
Problem 8b: 1.0/1
Problem 8c: 1.0/1
Problem 9: 2.0/2
Problem 10: 1.0/1
Problem 11: 2.0/2
Problem 12: 2.0/2
Problem EC 1: 1.0/1
Problem EC 2: 1.0/1

Score:
Total: 26.0

Cannot backup when running ok with --local.
---------------------------------------------------------------------
Final Score:27.0
Early Submission. Bonus Point Included
```

Submitted Files

ants.py Download

```
"""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
        if self.exit is not None:
            self.exit.entrance = self
        # END Problem 2

    def add_insect(self, insect):
        """
        Asks the insect to add itself to the current place. This method exists so
            it can be enhanced in subclasses.
        """
        insect.add_to(self)

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

    def __str__(self):
        return self.name

```

```
50
51  class Insect:
52      """An Insect, the base class of Ant and Bee, has health and a Place."""
53
54      damage = 0
55      is_waterproof = False #Prob 10
56      # ADD CLASS ATTRIBUTES HERE
57
58
59      def __init__(self, health, place=None):
60          """Create an Insect with a health amount and a starting PLACE."""
61          self.health = health
62          self.place = place  # set by Place.add_insect and Place.remove_insect
63
64      def reduce_health(self, amount):
65          """Reduce health by AMOUNT, and remove the insect from its place if it
66          has no health remaining.
67
68          >>> test_insect = Insect(5)
69          >>> test_insect.reduce_health(2)
70          >>> test_insect.health
71          3
72          """
73          self.health -= amount
74          if self.health <= 0:
75              self.death_callback()
76              self.place.remove_insect(self)
77
78
79
80
81      def action(self, gamestate):
82          """The action performed each turn.
83
84          gamestate -- The GameState, used to access game state information.
85          """
86
87      def death_callback(self):
88          # overriden by the gui
89          pass
90
91      def add_to(self, place):
92          """Add this Insect to the given Place
93
94          By default just sets the place attribute, but this should be overriden in the subclasses
95              to manipulate the relevant attributes of Place
96          """
97          self.place = place
98
99      def remove_from(self, place):
100         self.place = None
101
```

```
102     def __repr__(self):
103         cname = type(self).__name__
104         return '{0}({1}, {2})'.format(cname, self.health, self.place)
105 
106 
107 class Ant(Insect):
108     """An Ant occupies a place and does work for the colony."""
109 
110     implemented = False  # Only implemented Ant classes should be instantiated
111     food_cost = 0
112     is_container = False
113     # ADD CLASS ATTRIBUTES HERE
114 
115     def __init__(self, health=1):
116         """Create an Insect with a HEALTH quantity."""
117         super().__init__(health)
118         self.doubled = False
119 
120     @classmethod
121     def construct(cls, gamestate):
122         """Create an Ant for a given GameState, or return None if not possible."""
123         if cls.food_cost > gamestate.food:
124             print('Not enough food remains to place ' + cls.__name__)
125             return
126         return cls()
127 
128     def can_contain(self, other):
129         return False
130 
131     def store_ant(self, other):
132         assert False, "{0} cannot contain an ant".format(self)
133 
134     def remove_ant(self, other):
135         assert False, "{0} cannot contain an ant".format(self)
136 
137     def add_to(self, place):
138         if place.ant is None: #Empty
139             place.ant = self
140 
141         #Existing ant container and can contain new ant
142         elif place.ant.is_container and place.ant.can_contain(self):
143 
144             place.ant.store_ant(self)
145         # New ant is container ant and can contain exsisting ant
146         elif self.is_container and self.can_contain(place.ant):
147             old_ant = place.ant
148             place.ant = self
149             self.store_ant(old_ant)
150 
151         else:
152             # BEGIN Problem 8b
153             assert place.ant is None, 'Two ants in {0}'.format(place)
```

```
154             # END Problem 8b
155         Insect.add_to(self, place)
156 
157     def remove_from(self, place):
158         if place.ant is self:
159             place.ant = None
160         elif place.ant is None:
161             assert False, '{0} is not in {1}'.format(self, place)
162         else:
163             place.ant.remove_ant(self)
164         Insect.remove_from(self, place)
165 
166     def double(self):
167         # BEGIN Problem 12
168         "*** YOUR CODE HERE ***"
169         if not self.doubled:
170             self.damage *= 2
171             self.doubled = True # change to True is doubled
172 
173         # END Problem 12
174 
175 
176 class HarvesterAnt(Ant):
177     """HarvesterAnt produces 1 additional food per turn for the colony."""
178     name = 'Harvester'
179     implemented = True
180     # OVERRIDE CLASS ATTRIBUTES HERE
181     food_cost = 2 #######
182     def action(self, gamestate):
183         """Produce 1 additional food for the colony.
184         gamestate -- The GameState, used to access game state information.
185         """
186         # BEGIN Problem 1
187         gamestate.food+=1
188         # END Problem 1
189 
190 
191 class ThrowerAnt(Ant):
192     """ThrowerAnt throws a leaf each turn at the nearest Bee in its range."""
193 
194     name = 'Thrower'
195     implemented = True
196     damage = 1
197     # ADD/OVERRIDE CLASS ATTRIBUTES HERE
198     food_cost = 3 #########
199     lower_bound = 0
200     upper_bound=float('inf')
201     def nearest_bee(self):
202         """Return the nearest Bee in a Place that is not the HIVE, connected to
203         the ThrowerAnt's Place by following entrances.
204 
205         This method returns None if there is no such Bee (or none in range).
```

```
        """
        # BEGIN Problem 3 and 4
        distance = 0
        place = self.place
        while place:
            if (place.is_hive!=True) and (self.lower_bound<= distance <=self.upper_bound):
                bee = random_bee(place.bees)
                if bee:
                    return bee
            distance +=1
            place = place.entrance
        return None


        # 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."""
    # BEGIN Problem 4
    name = 'Short'
    food_cost = 2
    lower_bound = 0
    upper_bound=3
    #health = 1
    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."""
    # BEGIN Problem 4
    name = 'Long'
    food_cost = 2
    lower_bound = 5
    upper_bound = float('inf')
    #health = 1


    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
        total_damage = amount # Total damage to be refelcted
        if self.health - amount <= 0:  # when fire ant die
            total_damage += self.damage

        for bee in list(self.place.bees):  #Apply refelctive damage
            bee.reduce_health(total_damage)
        super().reduce_health(amount) #reduction and removal
        # END Problem 5

# BEGIN Problem 6
# The WallAnt class
class WallAnt(Ant):#############
    name = 'Wall'
    food_cost=4
    implemented=True

```

```
309     def __init__(self, health=4):
310         """Create an Ant with a HEALTH quantity."""
311         super().__init__(health)
312 
313     def action(self, gamestate):
314         """None."""
315         pass
316 
317 # END Problem 6
318 
319 # BEGIN Problem 7
320 # The HungryAnt Class
321 class HungryAnt(Ant):
322     name = 'Hungry'
323     food_cost=4
324     implemented=True
325     chewing_turns = 3
326 
327     def __init__(self, health=1):
328         """Create an Ant with a HEALTH quantity."""
329         super().__init__(health)
330         self.turns_to_chew=0
331 
332     def action(self, gamestate): ############
333         if self.turns_to_chew > 0:
334             self.turns_to_chew -= 1
335         else:
336             bees_in_place = self.place.bees
337 
338             if bees_in_place:
339                 eaten_bee = random.choice(bees_in_place)
340 
341                 eaten_bee.reduce_health(eaten_bee.health)
342 
343                 self.turns_to_chew = self.chewing_turns
344 
345 # END Problem 7
346 
347 
348 class ContainerAnt(Ant):
349     """
350     ContainerAnt can share a space with other ants by containing them.
351     """
352     is_container = True
353 
354 
355     def __init__(self, *args, **kwargs):
356         super().__init__(*args, **kwargs)
357         self.ant_contained = None
358 
359     def can_contain(self, other):
360         # BEGIN Problem 8a
```

```
361         "*** YOUR CODE HERE ***"
362         return (self.ant_contained is None) and (not other.is_container)#######
363         # END Problem 8a
364 
365     def store_ant(self, ant):
366         # BEGIN Problem 8a
367         "*** YOUR CODE HERE ***"
368         self.ant_contained = ant
369         # END Problem 8a
370 
371     def remove_ant(self, ant):
372         if self.ant_contained is not ant:
373             assert False, "{} does not contain {}".format(self, ant)
374         self.ant_contained = None
375 
376     def remove_from(self, place):
377 
378         if place.ant is self:
379             place.ant = place.ant.ant_contained
380             Insect.remove_from(self, place)
381         else:
382             # default to normal behavior
383             Ant.remove_from(self, place)
384 
385     def action(self, gamestate):
386         # BEGIN Problem 8a
387         "*** YOUR CODE HERE ***"
388         if self.ant_contained :
389             self.ant_contained.action(gamestate)
390         # END Problem 8a
391 
392 
393 class BodyguardAnt(ContainerAnt):
394     """BodyguardAnt provides protection to other Ants."""
395 
396     name = 'Bodyguard'
397     food_cost = 4
398 
399     # BEGIN Problem 8c
400     implemented = True   # Change to True to view in the GUI
401 
402     def __init__(self, health=2):
403         """Create an Ant with a HEALTH quantity."""
404         super().__init__(health)
405 
406     # END Problem 8c
407 
408 # BEGIN Problem 9
409 # The TankAnt class
410 
411 
412 class TankAnt(ContainerAnt):
```

```
    name = "Tank"
    implemented = True
    food_cost = 6
    damage =1
    def __init__(self, health=2):
        """Create an Ant with a HEALTH quantity."""
        super().__init__(health)

    def action(self, gamestate):
        if self.place.bees:
            for bee in list(self.place.bees):
                bee.reduce_health(self.damage)

        if self.ant_contained:
            self.ant_contained.action(gamestate)


# 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 ***"
        super().add_insect(insect)

        if insect.is_waterproof == False:
            insect.reduce_health(insect.health)

        # END Problem 10

# BEGIN Problem 11
# The ScubaThrower class
class ScubaThrower(ThrowerAnt):
    name ='ScubaThrower'
    is_waterproof =True
    food_cost = 6

# END Problem 11

# BEGIN Problem 12


class QueenAnt(ScubaThrower):  # You should change this line
# END Problem 12
    """QueenAnt is a ScubaThrower that boosts the damage of all ants behind her."""

    name = 'Queen'
```

```
    food_cost = 7

    # BEGIN Problem 12
    def __init__(self, health=1):
        """Create an Ant with a HEALTH quantity."""
        super().__init__(health)
        self.boosted_ants =[] # Damage *2 tracker

    implemented = True   # 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 ***"
        super().action(gamestate)


        curr_place = self.place


        while curr_place:
            if isinstance(curr_place.ant, ContainerAnt) and curr_place.ant.ant_contained == self:
             pass

            elif curr_place.ant and (curr_place.ant not in self.boosted_ants) and not
isinstance(curr_place.ant, QueenAnt):
                curr_place.ant.double()
            self.boosted_ants.append(curr_place.ant)

            if (isinstance(curr_place.ant, ContainerAnt) and curr_place.ant.ant_contained
                and (curr_place.ant.ant_contained not in self.boosted_ants)
                and (curr_place.ant.ant_contained != self)
                ):

                curr_place.ant.ant_contained.double()
                self.boosted_ants.append(curr_place.ant.ant_contained)

            curr_place = curr_place.exit




       # Track that we have boosted this ant


    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
        super().reduce_health(amount)

        self.health -= amount
        if self.health <= 0 and isinstance(self, QueenAnt):  # If the ant is a QueenAnt
            raise AntsLoseException("Ants have lost!")
        # END Problem 12

    def remove_from(self, place):
        # BEGIN Problem 12
        pass
        # END Problem 12



class SlowThrower(ThrowerAnt):
    """ThrowerAnt that causes Slow on Bees."""

    name = 'Slow'
    food_cost = 6
    # BEGIN Problem EC 1

    implemented = True   # Change to True to view in the GUI
    # END Problem EC 1

    def throw_at(self, target):
        # BEGIN Problem EC 1
        if target is None:
          return
        if not hasattr(target, 'slowed'):
            target.slowed = True
            target.original_action = target.action
            target.action = self.slowed_action_outer(target)
            target.slowed_count = 3
        else:
            target.slowed_count = 3
    def slowed_action_outer(self, target):
        def slowed_action(gamestate):
            if gamestate.time % 2 == 0:
                target.original_action(gamestate)
                target.slowed_count -= 1
            if target.slowed_count <= 0:
                target.action = target.original_action
                target.slowed = False
        return slowed_action

        # END Problem EC 1
```

```
568 class ScaryThrower(ThrowerAnt):
569     """ThrowerAnt that intimidates Bees, making them back away instead of advancing."""
570 
571     name = 'Scary'
572     food_cost = 6
573     # BEGIN Problem EC 2
574     implemented = True   # Change to True to view in the GUI
575     # END Problem EC 2
576 
577     def throw_at(self, target):
578         # BEGIN Problem EC 2
579         if target is None:
580             return
581         if not hasattr(target, "has_been_scared") or not target.has_been_scared:
582             target.scare(2)
583         # END Problem EC 2
584 
585 
586 class AntRemover(Ant):
587     """Allows the player to remove ants from the board in the GUI."""
588     name = 'Remover'
589     implemented = False
590     def __init__(self):
591         super().__init__(0)
592 
593 
594 class Bee(Insect):
595     """A Bee moves from place to place, following exits and stinging ants."""
596 
597     name = 'Bee'
598     damage = 1
599     is_waterproof = True #Prob 10
600 
601     # OVERRIDE CLASS ATTRIBUTES HERE
602 
603     def sting(self, ant):
604         """Attack an ANT, reducing its health by 1."""
605         ant.reduce_health(self.damage)
606 
607     def move_to(self, place):
608         """Move from the Bee's current Place to a new PLACE."""
609         self.place.remove_insect(self)
610         place.add_insect(self)
611 
612     def blocked(self):
613         """Return True if this Bee cannot advance to the next Place."""
614         # Special handling for NinjaAnt
615         # BEGIN Problem Optional 1
616         return self.place.ant is not None
617         # END Problem Optional 1
618 
619     def action(self, gamestate):
```

```
620         """A Bee's action stings the Ant that blocks its exit if it is blocked,
621         or moves to the exit of its current place otherwise.
622 
623         gamestate -- The GameState, used to access game state information.
624         """
625         destination = self.place.exit
626 
627         if self.blocked():
628             self.sting(self.place.ant)
629         elif self.health > 0 and destination is not None:
630             self.move_to(destination)
631 
632     def add_to(self, place):
633         place.bees.append(self)
634         Insect.add_to(self, place)
635 
636     def remove_from(self, place):
637         place.bees.remove(self)
638         Insect.remove_from(self, place)
639 
640     def scare(self, length):
641         """
642         If this Bee has not been scared before, cause it to attempt to
643         go backwards LENGTH times.
644         """
645         # BEGIN Problem EC 2
646         if not hasattr(self, "has_been_scared"):
647             self.has_been_scared = True
648             self.original_move_to = self.move_to
649 
650         def scared_move_to(place):
651             if hasattr(self, 'scared_turns') and (self.scared_turns > 0):
652                 self.scared_turns -= 1
653                 if self.place.entrance:
654                     self.original_move_to(self.place.entrance)
655             else:
656                 self.move_to = self.original_move_to
657                 self.move_to(place)
658         self.move_to = scared_move_to #backwards
659         self.scared_turns = length
660         # END Problem EC 2
661 
662 
663 ############
664 # Optional #
665 ############
666 
667 class NinjaAnt(Ant):
668     """NinjaAnt does not block the path and damages all bees in its place.
669     This class is optional.
670     """
671 
```

```
672     name = 'Ninja'
673     damage = 1
674     food_cost = 5
675     # OVERRIDE CLASS ATTRIBUTES HERE
676     # BEGIN Problem Optional 1
677     implemented = False   # Change to True to view in the GUI
678     # END Problem Optional 1
679 
680     def action(self, gamestate):
681         # BEGIN Problem Optional 1
682         "*** YOUR CODE HERE ***"
683         # END Problem Optional 1
684 
685 ############
686 # Statuses #
687 ############
688 
689 
690 class LaserAnt(ThrowerAnt):
691     # This class is optional. Only one test is provided for this class.
692 
693     name = 'Laser'
694     food_cost = 10
695     # OVERRIDE CLASS ATTRIBUTES HERE
696     # BEGIN Problem Optional 2
697     implemented = False   # Change to True to view in the GUI
698     # END Problem Optional 2
699 
700     def __init__(self, health=1):
701         super().__init__(health)
702         self.insects_shot = 0
703 
704     def insects_in_front(self):
705         # BEGIN Problem Optional 2
706         return {}
707         # END Problem Optional 2
708 
709     def calculate_damage(self, distance):
710         # BEGIN Problem Optional 2
711         return 0
712         # END Problem Optional 2
713 
714     def action(self, gamestate):
715         insects_and_distances = self.insects_in_front()
716         for insect, distance in insects_and_distances.items():
717             damage = self.calculate_damage(distance)
718             insect.reduce_health(damage)
719             if damage:
720                 self.insects_shot += 1
721 
722 
723 ##################
```

```
# Bees Extension #
##################

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


class Hornet(Bee):
    """Class of bee that is capable of taking two actions per turn, although
    its overall damage output is lower. Immune to statuses.
    """
    name = 'Hornet'
    damage = 0.25

    def action(self, gamestate):
        for i in range(2):
            if self.health > 0:
                super().action(gamestate)

    def __setattr__(self, name, value):
        if name != 'action':
            object.__setattr__(self, name, value)


class NinjaBee(Bee):
    """A Bee that cannot be blocked. Is capable of moving past all defenses to
    assassinate the Queen.
    """
    name = 'NinjaBee'

    def blocked(self):
        return False


class Boss(Wasp, Hornet):
    """The leader of the bees. Combines the high damage of the Wasp along with
    status immunity of Hornets. Damage to the boss is capped up to 8
    damage by a single attack.
    """
    name = 'Boss'
    damage_cap = 8
    action = Wasp.action

    def reduce_health(self, amount):
        super().reduce_health(self.damage_modifier(amount))

    def damage_modifier(self, amount):
        return amount * self.damage_cap / (self.damage_cap + amount)


```

```
776 class Hive(Place):
777     """The Place from which the Bees launch their assault.
778 
779     assault_plan -- An AssaultPlan; when & where bees enter the colony.
780     """
781     is_hive = True
782 
783     def __init__(self, assault_plan):
784         self.name = 'Hive'
785         self.assault_plan = assault_plan
786         self.bees = []
787         for bee in assault_plan.all_bees:
788             self.add_insect(bee)
789         # The following attributes are always None for a Hive
790         self.entrance = None
791         self.ant = None
792         self.exit = None
793 
794     def strategy(self, gamestate):
795         exits = [p for p in gamestate.places.values() if p.entrance is self]
796         for bee in self.assault_plan.get(gamestate.time, []):
797             bee.move_to(random.choice(exits))
798             gamestate.active_bees.append(bee)
799 
800 
801 class GameState:
802     """An ant collective that manages global game state and simulates time.
803 
804     Attributes:
805     time -- elapsed time
806     food -- the colony's available food total
807     places -- A list of all places in the colony (including a Hive)
808     bee_entrances -- A list of places that bees can enter
809     """
810 
811     def __init__(self, strategy, beehive, ant_types, create_places, dimensions, food=2):
812         """Create an GameState for simulating a game.
813 
814         Arguments:
815         strategy -- a function to deploy ants to places
816         beehive -- a Hive full of bees
817         ant_types -- a list of ant classes
818         create_places -- a function that creates the set of places
819         dimensions -- a pair containing the dimensions of the game layout
820         """
821         self.time = 0
822         self.food = food
823         self.strategy = strategy
824         self.beehive = beehive
825         self.ant_types = OrderedDict((a.name, a) for a in ant_types)
826         self.dimensions = dimensions
827         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 simulate(self):
        """Simulate an attack on the ant colony (i.e., play the game)."""
        num_bees = len(self.bees)
        try:
            while True:
                self.beehive.strategy(self)        # Bees invade
                self.strategy(self)                # Ants deploy
                for ant in self.ants:              # Ants take actions
                    if ant.health > 0:
                        ant.action(self)
                for bee in self.active_bees[:]:    # Bees take actions
                    if bee.health > 0:
                        bee.action(self)
                    if bee.health <= 0:
                        num_bees -= 1
                        self.active_bees.remove(bee)
                if num_bees == 0:
                    raise AntsWinException()
                self.time += 1
        except AntsWinException:
            print('All bees are vanquished. You win!')
            return True
        except AntsLoseException:
            print('The ant queen has perished. Please try again.')
            return 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]
        ant = ant_type.construct(self)
        if ant:
            self.places[place_name].add_insect(ant)
            self.food -= ant.food_cost
```

```
880             return ant
881
882     def remove_ant(self, place_name):
883         """Remove an Ant from the game."""
884         place = self.places[place_name]
885         if place.ant is not None:
886             place.remove_insect(place.ant)
887
888     @property
889     def ants(self):
890         return [p.ant for p in self.places.values() if p.ant is not None]
891
892     @property
893     def bees(self):
894         return [b for p in self.places.values() for b in p.bees]
895
896     @property
897     def insects(self):
898         return self.ants + self.bees
899
900     def __str__(self):
901         status = ' (Food: {0}, Time: {1})'.format(self.food, self.time)
902         return str([str(i) for i in self.ants + self.bees]) + status
903
904
905 class AntHomeBase(Place):
906     """AntHomeBase at the end of the tunnel, where the queen resides."""
907
908     def add_insect(self, insect):
909         """Add an Insect to this Place.
910
911         Can't actually add Ants to a AntHomeBase. However, if a Bee attempts to
912         enter the AntHomeBase, a AntsLoseException is raised, signaling the end
913         of a game.
914         """
915         assert isinstance(insect, Bee), 'Cannot add {0} to AntHomeBase'
916         raise AntsLoseException()
917
918
919 def ants_win():
920     """Signal that Ants win."""
921     raise AntsWinException()
922
923
924 def ants_lose():
925     """Signal that Ants lose."""
926     raise AntsLoseException()
927
928
929 def ant_types():
930     """Return a list of all implemented Ant classes."""
931     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]


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


def interactive_strategy(gamestate):
    """A strategy that starts an interactive session and lets the user make
    changes to the gamestate.

    For example, one might deploy a ThrowerAnt to the first tunnel by invoking
    gamestate.deploy_ant('tunnel_0_0', 'Thrower')
    """
    print('gamestate: ' + str(gamestate))
    msg = '<Control>-D (<Control>-Z <Enter> on Windows) completes a turn.\n'
    interact(msg)

############
# 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

    @property
    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]

```