Ants defense

AUTHOR
Christian Osendorfer

PUBLISHED
November 30, 2023

Introduction

Acknowledgements: This project is taken from the excellent CS61A course in Berkeley.

In this project, you will create a tower defense game called **Ants Vs. SomeBees**. As the ant queen, you populate your colony with the bravest ants you can muster. Your ants must protect their queen from the evil bees that invade your territory. Irritate the bees enough by throwing leaves at them, and they will be vanquished. Fail to pester the airborne intruders adequately, and your queen will succumb to the bees' wrath.

Logistics

The `proj02.zip` archive contains several files, but all of your changes will be made to `ants.py`.

- `ants.py`: The game logic of Ants Vs. SomeBees
- `ants_gui.py`: The original GUI for Ants Vs. SomeBees.
- `gui.py`: A new GUI for Ants Vs. SomeBees.
- `graphics.py`: Utilities for displaying simple two-dimensional animations.
- `utils.py`: Some functions to facilitate the game interface.
- `ucb.py`: Utility functions.
- `state.py`: Abstraction for gamestate for gui.py.
- `assets`: A directory of images and files used by gui.py.
- `img`: A directory of images used by ants_gui.py.

For the functions that we ask you to complete, there may be some initial code that we provide. If you would rather not use that code, feel free to delete it and start from scratch. You may also add new function definitions as you see fit.

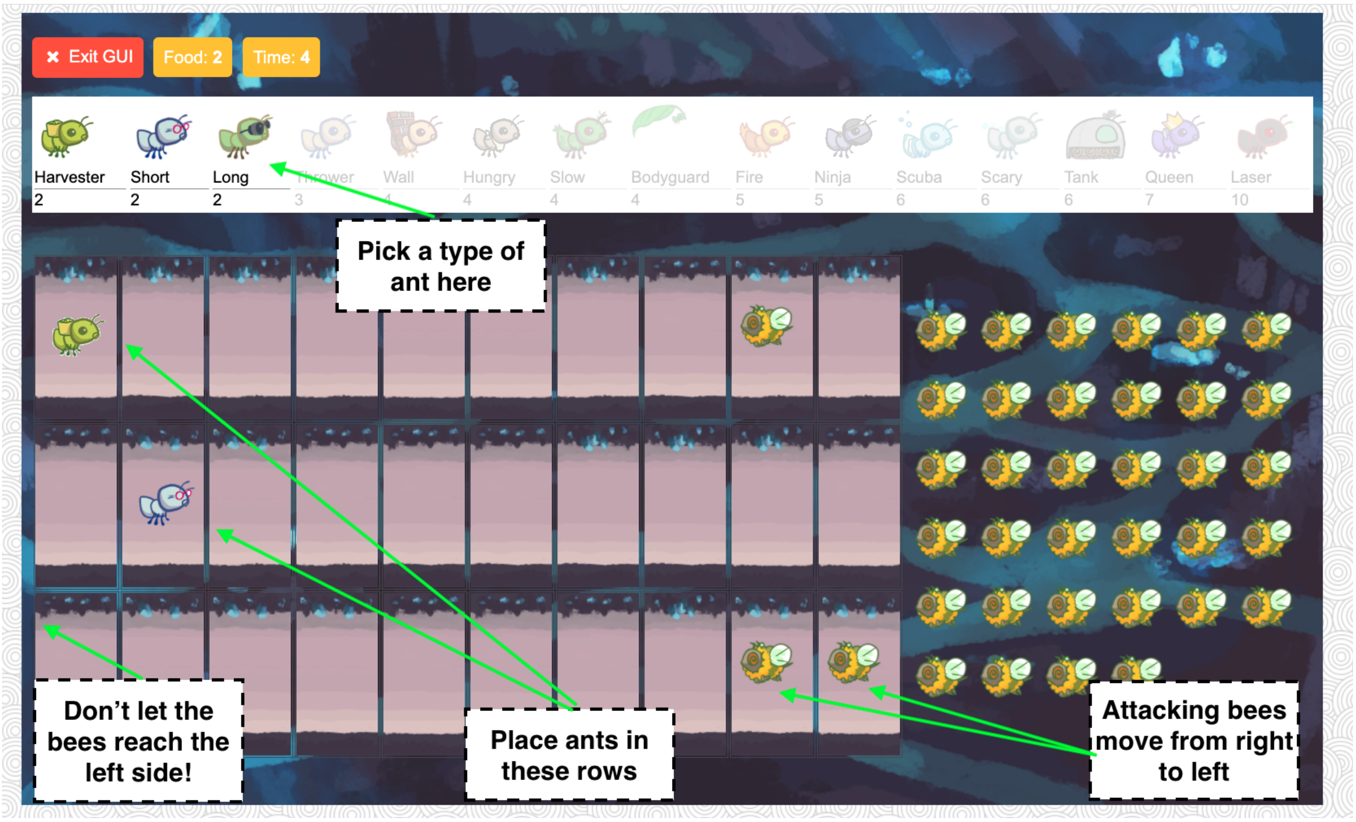
However, **please do not modify** any other functions, tests or edit any files not listed above. Doing so may result in your code failing e.g. the autograder tests. Also, please do not change any function signatures (names, argument order, or number of arguments).

Important

Add the full program (all included files) to a new repository on studilab, under your personal account. While programming, push regularly to this repo.

The Game

A game of Ants Vs. SomeBees consists of a series of turns. In each turn, new bees may enter the ant colony. Then, new ants are placed to defend their colony. Finally, all insects (ants, then bees) take individual actions. Bees either try to move toward the end of the tunnel or sting ants in their way. Ants perform a different action depending on their type, such as collecting more food or throwing leaves at the bees. The game ends either when a bee reaches the end of the tunnel (you lose), the bees destroy the QueenAnt if it exists (you lose), or the entire bee fleet has been vanquished (you win).



The Game GUI

Core concepts

The Colony. This is where the game takes place. The colony consists of several `Places` that are chained together to form a tunnel where bees can travel through. The colony also has some quantity of food which can be expended in order to place an ant in a tunnel.

Places. A place links to another place to form a tunnel. The player can put a single ant into each place. However, there can be many bees in a single place.

The Hive. This is the place where bees originate. Bees exit the beehive to enter the ant colony.

Ants. Players place an ant into the colony by selecting from the available ant types at the top of the screen. Each type of ant takes a different action and requires a different amount of colony food to place. The two most basic ant types are the `HarvesterAnt`, which adds one food to the colony during each turn, and the `ThrowerAnt`, which throws a leaf at a bee each turn. You will be implementing many more!

Bees. In this game, bees are the antagonistic forces that the player must defend the ant colony from. Each turn, a bee either advances to the next place in the tunnel if no ant is in its way, or it stings the ant in its way. Bees win when at least one bee reaches the end of a tunnel.

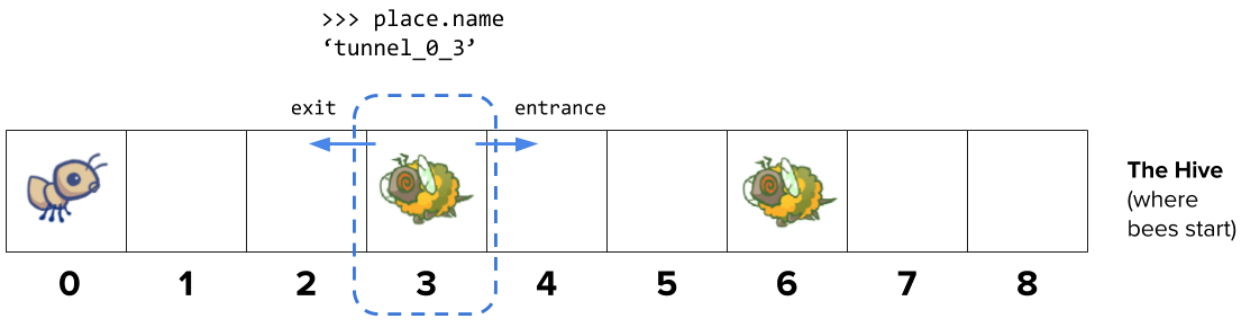
Core classes

The concepts described above each have a corresponding class that encapsulates the logic for that concept. Here is a summary of the main classes involved in this game:

- `GameState`: Represents the colony and some state information about the game, including how much food is available, how much time has elapsed, where the `AntHomeBase` is, and all the `Place`s in the game.
- `Place`: Represents a single place that holds insects. At most one `Ant` can be in a single place, but there can be many `Bee`s in a single place. `Place` objects have an exit to the left and an entrance to the right, which are also places. `Bee`s travel through a tunnel by moving to a `Place`'s exit.
- `Hive`: Represents the place where `Bee`s start out (on the right of the tunnel).
- `AntHomeBase`: Represents the place `Ant`s are defending (on the left of the tunnel). If `Bee`s get here, they win :(
- `Insect`: A superclass for `Ant` and `Bee`. All insects have health attribute, representing their remaining health, and a place attribute, representing the `Place` where they are currently located. Each turn, every active Insect in the game performs its action.
- `Ant`: Represents ants. Each `Ant` subclass has special attributes or a special action that distinguish it from other `Ant` types. For example, a `HarvesterAnt` gets food for the colony and a `ThrowerAnt` attacks Bees. Each ant type also has a food_cost attribute that indicates how much it costs to deploy one unit of that type of ant.
- `Bee`: Represents bees. Each turn, a bee either moves to the exit of its current `Place` if the `Place` is not blocked by an ant, or stings the ant occupying its same Place.

Game Layout

Below is a visualization of a `GameState`. As you work through the unlocking tests and problems, we recommend drawing out similar diagrams to help your understanding.



Example: AntColony with dimensions (1, 9)

Playing the game

The game can be run in two modes: as a text-based game or using a graphical user interface (GUI). The game logic is the same in either case, but the GUI enforces a turn time limit that makes playing the game more exciting. The text-based interface is provided for debugging and development.

The files are separated according to these two modes. `ants.py` knows nothing of graphics or turn time limits.

To start a text-based game, run

```
python3 ants_text.py
```

To start a graphical game, run

```
python3 gui.py
```

When you start the graphical version, a new browser window should appear. In the starter implementation, you have unlimited food and your ants can only throw leaves at bees in their current Place. Before you complete Problem 2, the GUI may crash since it doesn't have a full conception of what a `Place` is yet! Try playing the game anyway! You'll need to place a lot of ThrowerAnts (the second type) in order to keep the bees from reaching your queen.

The game has several options that you will use throughout the project, which you can view with `python3 ants_text.py --help`.

```
usage: ants_text.py [-h] [-d DIFFICULTY] [-w] [--food FOOD]

Play Ants vs. SomeBees

optional arguments:
  -h, --help      show this help message and exit
  -d DIFFICULTY   sets difficulty of game (test/easy/normal/hard/extra-hard)
  -w, --water     loads a full layout with water
  --food FOOD     number of food to start with when testing
```

Basic gameplay

In the first phase you will complete the implementation that will allow for basic gameplay with the two basic Ants: the `HarvesterAnt` and the `ThrowerAnt`.

Problem 0

Answer the following questions after you have read the entire ants.py file.

To submit your answers, run:

```
python3 ok -q 00 -u
```

Ignore the request to submit your *.edu* email address!

If you get stuck while answering these questions, you can try reading through `ants.py` again, consult the core concepts/classes sections above, or ask a question in Moodle.

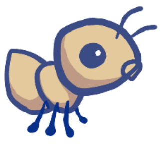
- What is the significance of an `Insect`'s health attribute? Does this value change? If so, how?
- Which of the following is a class attribute of the `Insect` class?
- Is the health attribute of the `Ant` class an instance attribute or a class attribute? Why?
- Is the damage attribute of an `Ant` subclass (such as `ThrowerAnt`) an instance attribute or class attribute? Why?
- Which class do both `Ant` and `Bee` inherit from?
- What do instances of `Ant` and instances of `Bee` have in common?
- How many insects can be in a single `Place` at any given time
- What does a `Bee` do during one of its turns?
- When is the game lost?

Note

A note on unlocking tests: If you'd like to review the unlocking questions after you have completed the unlocking test, you can navigate to (within the ants folder), the tests folder. For example, after unlocking Problem 0, you can review the unlocking test at `tests/00.py`.

Problem 1

Part A: Currently, there is no cost for placing any type of `Ant`, and so there is no challenge to the game. The base class `Ant` has a `food_cost` of zero. Override this class attribute for `HarvesterAnt` and `ThrowerAnt` according to the "Food Cost" column in the table below.

Class	Food Cost	Initial Health
	2	1
	2	1

Part B: Now that placing an `Ant` costs food, we need to be able to gather more food! To fix this issue, implement the `HarvesterAnt` class. A `HarvesterAnt` is a type of Ant that adds one food to the `gamestate.food` total as its action.

Before writing any code, unlock the tests to verify your understanding of the question:

```
python3 ok -q 01 -u
```

Once you are done unlocking, begin implementing your solution. You can check your correctness with:

```
python3 ok -q 01
```

Try playing the game by running `python3 gui.py`. Once you have placed a `HarvesterAnt`, you should accumulate food each turn. You can also place `ThrowerAnts`, but you'll see that they can only attack bees that are in their `Place`, making it a little difficult to win.

Problem 2

In this problem, you'll complete `Place.__init__` by adding code that tracks entrances. Right now, a `Place` keeps track only of its `exit`. We would like a `Place` to keep track of its entrance as well. A `Place` needs to track only one `entrance`. Tracking entrances will be useful when an `Ant` needs to see what `Bees` are in front of it in the tunnel.

However, simply passing an entrance to a `Place` constructor will be problematic; we would need to have both the exit and the entrance before creating a `Place`! To get around this problem, we will keep track of entrances in the following way instead. `Place.__init__` should use this logic:

- A newly created `Place` always starts with its `entrance` as `None`.
- If the `Place` has an `exit`, then the `exit`'s `entrance` is set to that `Place`.

Tip

Try drawing out two `Place`s next to each other if things get confusing. In the GUI, a place's `entrance` is to its right while the `exit` is to its left.

Tip

Remember that `Places` are not stored in a list, so you can't index into anything to access them. This means that you can't do something like `colony[index + 1]` to access an adjacent `Place`. How can you move from one place to another?

Problem 3

In order for a `ThrowerAnt` to throw a leaf, it must know which bee to hit. The provided implementation of the nearest_bee method in the `ThrowerAnt` class only allows them to hit bees in the same Place. Your job is to fix it so that a `ThrowerAnt` will throw_at the nearest bee in front of it that is not still in the Hive. This includes bees that are in the same `Place` as a `ThrowerAnt`.

Tip

All `Place`s have an `is_hive` attribute which is `True` when that place is the Hive.

Change `nearest_bee` so that it returns a random `Bee` from the nearest place that contains bees. Your implementation should follow this logic:

- Start from the current Place of the ThrowerAnt.
- For each place, return a random bee if there is any, and if not, inspect the place in front of it (stored as the current place's entrance).
- If there is no bee to attack, return None.

> **Tip**
>
> The `random_bee` function provided in `ants.py` returns a random bee from a list of bees or `None` if the list is empty.

> **Tip**
>
> As a reminder, if there are no bees present at a Place, then the bees attribute of that Place instance will be an empty list.

Before writing any code, unlock the tests to verify your understanding of the question:

```
python3 ok -q 03 -u
```

Once you are done unlocking, begin implementing your solution. You can check your correctness with:

```
python3 ok -q 03
```

After implementing `nearest_bee`, a `ThrowerAnt` should be able to `throw_at` a `Bee` in front of it that is not still in the Hive. Make sure that your ants do the right thing! To start a game with ten food (for easy testing):

```
python3 gui.py --food 10
```

Use

```
python3 ok --score
```

to check how well your solution for the first overall part of this project works (don't get confused by the many many tests that are running also!).

Ants!

Now that you've implemented basic gameplay with two types of `Ant`s, let's add some flavor to the ways ants can attack bees. In this phase, you'll be implementing several different `Ant`s with different attack strategies.

After you implement each `Ant` subclass in this section, you'll need to set its implemented class attribute to `True` so that that type of ant will show up in the GUI. Feel free to try out the game with each new ant to test the functionality!

With your new ants, try `python3 gui.py -d easy` to play against a full swarm of bees in a multi-tunnel layout and try `-d normal`, `-d hard`, or `-d extra-hard` if you want a real challenge! If the bees are too numerous to vanquish, you might need to create some new ants.

Problem 4

A `ThrowerAnt` is a powerful threat to the bees, but it has a high food cost. In this problem, you'll implement two subclasses of `ThrowerAnt` that are less costly but have constraints on the distance they can throw:

- The `LongThrower` can only `throw_at` a `Bee` that is found after following at least 5 entrance transitions. It cannot hit `Bee`s that are in the same `Place` as it or the first four `Place`s in front of it. If there are two `Bee`s, one too close to the `LongThrower` and the other within its range, the `LongThrower` should only throw at the farther `Bee`, which is within its range, instead of trying to hit the closer `Bee`.
- The `ShortThrower` can only throw_at a `Bee` that is found after following at most 3 entrance transitions. It cannot throw at any bees further than 3 `Place`s in front of it.

Class	Food Cost	Initial Health
	2	1
	2	1

Neither of these specialized throwers can `throw_at` a `Bee` that is exactly 4 Places away.

To implement these new throwing ants, your `ShortThrower` and `LongThrower` classes should inherit the `nearest_bee` method from the base `ThrowerAnt` class. The logic of choosing which bee a thrower ant will attack is the same, except the `ShortThrower` and `LongThrower` ants have a maximum and minimum range, respectively.

To do this, modify the `nearest_bee` method to reference `min_range` and `max_range` attributes, and only return a bee if it is within range.

Make sure to give these `min_range` and `max_range` attributes appropriate values in the `ThrowerAnt` class so that the behavior of `ThrowerAnt` is unchanged. Then, implement the subclasses `LongThrower` and `ShortThrower` with appropriately constrained ranges.

You should not need to repeat any code between `ThrowerAnt`, `ShortThrower`, and `LongThrower`.

> **Tip**
>
> `float('inf')` returns an infinite positive value represented as a float that can be compared with other numbers.

> **Tip**
>
> You can chain inequalities in Python: e.g. `2 < x < 6` will check if `x` is between 2 and 6. Also, `min_range` and `max_range` should mark an inclusive range.

Important

Make sure your class attributes are called `max_range` and `min_range`. The tests directly reference these attribute names, and will error if you use another name for these attributes.

Don't forget to set the implemented class attribute of `LongThrower` and `ShortThrower` to `True`.

Problem 5

Implement the `FireAnt`, which does damage when it receives damage. Specifically, if it is damaged by `amount` health units, it does a damage of amount to all bees in its place (this is called reflected damage). If it dies, it does an additional amount of damage, as specified by its `damage` attribute, which has a default value of 3 as defined in the `FireAnt` class.

To implement this, override `Ant`'s `reduce_health` method. Your overriden method should call the `reduce_health` method inherited from the superclass (`Ant`) to reduce the current `FireAnt` instance's health. Calling the inherited `reduce_health` method on a `FireAnt` instance reduces the insect's health by the given amount and removes the insect from its place if its health reaches zero or lower.

Do not call `self.reduce_health`, or you'll end up stuck in a recursive loop. (Can you see why?)

However, your method needs to also include the reflective damage logic:

- Determine the reflective damage amount: start with the amount inflicted on the ant, and then add damage if the ant's health has dropped to 0.
- For each bee in the place, damage them with the total amount by calling the appropriate reduce_health method for each bee.

Important

Remember that when any Ant loses all its health, it is removed from its place, so pay careful attention to the order of your logic in reduce_health.

Class	Food Cost	Initial Health
	5	3

Tip

Damaging a bee may cause it to be removed from its place. If you iterate over a list, but change the contents of that list at the same time, you may not visit all the elements. This can be prevented by making a copy of the list. You can either use a list slice, or use the built-in list function.

```
>>> lst = [1,2,3,4]
>>> lst[:]
[1, 2, 3, 4]
>>> list(lst)
[1, 2, 3, 4]
```

```
>>> lst[:] is not lst and list(lst) is not lst
True
```

Once you've finished implementing the `FireAnt`, give it a class attribute implemented with the value `True`.

Note

Even though you are overriding the superclass's `reduce_health` function (`Ant.reduce_health`), you can still use this method in your implementation by calling it. Note this is not recursion. (Why not?)

Before writing any code, unlock the tests to verify your understanding of the question:

```
python3 ok -q 05 -u
```

Once you are done unlocking, begin implementing your solution. You can check your correctness with:

```
python3 ok -q 05
```

You can also test your program by playing a game or two! A `FireAnt` should destroy all co-located Bees when it is stung. To start a game with ten food (for easy testing):

```
python3 gui.py --food 10
```

Overall there are much more problems (`Ant` types) available. For us, the tasks so far are enough!