Lab 9: 6.009 Zoo, Part 2

Table of Contents

1) Preparation

This lab assumes you have Python 3.5 or later installed on your machine.

The following file contains code and other resources as a starting point for this lab: `lab9.zip`

Most of your changes should be made to `lab.py`, which you will submit at the end of this lab. Importantly, you should not add any imports to the file.

This lab is worth a total of 4 points. Your score for the lab is based on:

- passing the test cases from `test.py` under the time limit (2 points), and
- a brief "checkoff" conversation with a staff member to discuss your code (2 points).

For this lab, you will only receive credit for a test case if it runs to completion under the time limit on the server.

Your code submission is due at 4pm on Friday, May 3.

2) Introduction

The 6.009 Zoo is setting new records, and the zoo board of directors suddenly has big aspirations. They've hired a new CEO versed in seizing market opportunities for scalable growth, to prepare for an IPO.

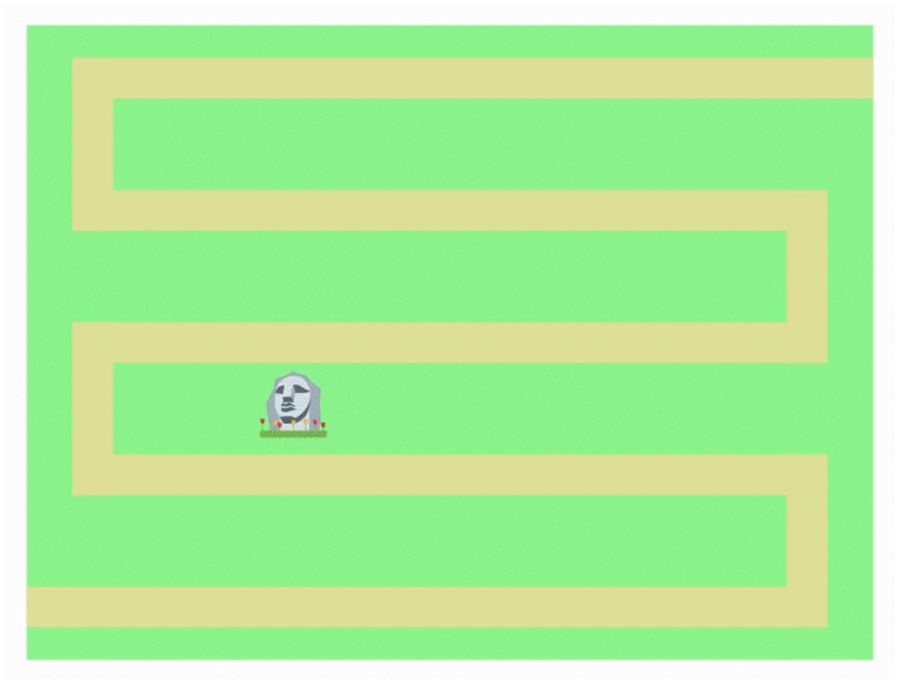
Yes, it's Santa Claus, who you helped in Labs 6 and 7, so of course he hires you as a consultant for the zoo expansions. Santa provides a list of several *required* extensions, documented below, and he also wants you to be creative and come up with an extension on your own.

This lab is your chance to show off how effectively your object-oriented design from Lab 8 enables agile experimentation with new zoo features. We're referring to the set of classes you've defined and how they relate to each other. Seemingly small decisions can have big consequences for how easy it is to add a particular new feature. Ideally, each new feature is confined to one or two new classes, with no need to change anything else! But, to achieve that ideal, you need to think ahead regarding which *dimensions of extensibility* your design should be prepared for.

Actually, we do expect you to make some changes in your Lab 8 architecture, as developing extensions in this lab reveals new design opportunities. Your checkoff conversation for this lab is a chance to explain your thinking behind the whole process, backed up by evidence of how different features are pleasantly confined to their own classes.

It may be helpful to refer back to the Lab 8 instructions, as we will be reusing most of the concepts and infrastructure. Your Lab 9 solution should start from your Lab 8 solution.

Here's what gameplay may look like after you implement this lab:



3) Required Features

Focus groups have revealed pent-up demand for a few different kinds of zookeepers or other ways of managing the animals. Here we describe the different new buyable formations that Santa requires you to implement. Like in the last lab, see the `Constants` class at the start of the starter `lab.py` file for important quantitative details about these new formations.

3.1) The Demon

Santa gets frustrated with how slowly the animals make their way through the feeding line, so he figures a little motivation is in order, and what better motivation than a huge scary Japanese mask?

When animals see this mask, they are frightened enough to hurry along -- doubling their speed! Only animals within `DEMON_RADIUS` of the mask's center are affected. A Demon has the following stats:

- `'Demon'` has price 100, sight range 150, and speed multiplier 2x.

3.2) The VHS Cassette

On the other hand, Santa believes in A/B testing and wants to make sure the zoo's bottom line couldn't be better improved by *slowing down* the animals. To that end, he asks you to deploy mysterious artifacts from the past.

What we have here is a VHS cassette, used by primitive civilizations to store video. The animals have never seen one before, so they stop to gawk as they pass. More specifically, their speeds are cut in half as they pass within `VHS_RADIUS` of a cassette. A VHS cassette has the following stats:

- `'VHS'` has price 20, sight range 150, and speed multiplier .5x.

The impacts of demon masks and VHS cassettes whose range contains an animal all apply cumulatively, multiplying their different factors (2 and 0.5) together to determine the final effect on an animal's speed.

If an animal speed is not an integer after applying multipliers, the speed is **rounded up** so that the speed remains an integer. For example, an after-multiplier animal speed of 2.3 will be rounded to 3. To do this rounding, you may use the `ceil` function we import for you. (No other imports are allowed, as usual.) This is the only time your implemetation should do any rounding.

VHS cassettes and demons cannot be placed such that they collide with other zookeepers, demons/VHS cassettes, rocks, or the path. In the same way, zookeepers cannot be placed such that they collide with demons/VHS cassettes.

3.3) The Trainee Zookeeper

Santa is aghast at how much of the budget is being spent on labor. Maybe we can provide on-the-job training to students from the local veterinary college, who will work for cheap? Or maybe the local daycare is even cheaper?

A trainee zookeeper is cheaper to purchase than the kinds from last lab, though he has pretty unimpressive throwing stats. However, every day is a learning experience! Once he has successfully used `TRAINEE_THRESHOLD` foods to feed animals, he upgrades into a speedy zookeeper, looking and acting just as if he had begun existence as speedy. Foods that were already thrown by the trainee before his upgrade should remain the same speed.

If two of trainee zookeeper's foods collide with the same animal, this will count for two ticks towards the `TRAINEE_THRESHOLD`. If one of trainee zookeeper's foods collides with two animals, this will only count for one tick towards the `TRAINEE_THRESHOLD`. Yes, this means that one piece of food can feed multiple animals! A trainee zookeeper initially has the following stats:

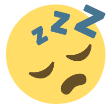
- `'TraineeZookeeper'` has price 50, sight range 100, and throw speed 5.

On the timestep when a trainee zookeeper reaches a total of `Constants.TRAINEE_THRESHOLD` foods used to feed animals, a trainee zookeeper upgrades to a speedy zooekeeper with the following stats:

- `'SpeedyZookeeper'` has sight range 50 and throw speed 20.

3.4) The Crazy Zookeeper

Santa worries that the zoo hasn't been tapping unconventional talent. In particular, a mysterious school of virtuoso food-throwers has come to light.



Though his appearance doesn't inspire confidence, the crazy zookeeper has immense throwing speed and range. However, he is easily exhausted. The crazy zookeeper starts awake and after *every throw*, he goes to sleep for `Constants.CRAZY_NAP_LENGTH` timesteps, not throwing again until he awakens. The first mug shot above applies when he is awake, while the second (unsurprisingly) is how he looks when asleep. Your code should keep track of the appropriate texture (that corresponding to `'CrazyZookeeper'` or `'SleepingZookeeper'` in the `TEXTURES` dictionary) for return in `render`.

A crazy zookeeper has the following stats and is only able to throw when he is awake:

- `'CrazyZookeeper'` has price 100, sight range 1000, and throw speed 50.

3.5) Game Timestep

In summary, your method `timestep(self, mouse)` should now make the following updates/changes **in the listed order**:

1. **Compute the new speed of animals based on the presence of nearby VHS cassettes or demons.**
2. Compute any changes in formation locations and remove any off-board formations.
3. Handle any food-animal collisions, and remove the fed animals and the eaten food.
4. **Upgrade trainee zookeeper if needed.**
5. Throw new food if possible, using the same algorithm as in lab 8. *The algorithm should use the speed of `aim_animal` computed in step 1.*
6. Spawn a new animal from the path's start if needed.
7. Handle `mouse` input, which is the integer tuple coordinate of a player's click, the string label of a particular zookeeper type, or `None`.
8. Redeem one dollar per animal fed this timestep.
9. Check for the losing condition.

4) Your Own Innovation

As the last requirement for the lab, show off your design chops to Santa by designing a new game feature of your choice. Ideally it will show off how much flexibility your object-oriented design provides, for adding new features with minimal tinkering with existing code. Unsurprisingly, there are no tests for your own feature (though we encourage you to write new tests to help test your feature!). Instead, your checkoff conversation is how we will evaluate how convincingly you have implemented this feature.

To add new characters on the board, you'll want to extend the `Constants.TEXTURES` dictionary. You can refer to `resources/emoji_table.txt` for a list of Emoji codes, any of which our game UI is prepared to render. (All along, we have been implementing the graphics via an emoji font, just as you may be used to from text messaging on phones!)

To make new zookeeper kinds buyable in the game UI, add them to the dictionary `TOWER_INFO` in file `ui/ui.js`. This file is written in the JavaScript programming language, but luckily the syntax for these dictionary values is the same as in Python, so you shouldn't need special JavaScript knowledge to edit the list. Note that the UI's zookeeper information is partly redundant with stats from your Python code, but the information in the UI is only used to generate the clickable menu of zookeepers and then has no effect on the game logic, behind determining which zookeeper kind string is passed to your timestep function.

By the way, it should be fun to experiment with extensions that go beyond new zookeeper kinds. You might consider new kinds of animals or food, or even completely new game mechanics (so long as you don't change the rules so much as to invalidate our test cases for required features).

5) Code Submission

if (ALLOW_STUDENT_SUBMISSIONS or cs_user_info.get('role', None) in {'UTA', 'TA', 'Instructor', 'Admin'}): print("""

Download Your Last Submission
Click to View Your Last Submission
Select File `No file selected`

Submit

You have submitted this assignment 2 times.

This question is due on Friday May 03, 2019 at 04:00:00 PM.

""") else: print(""" Submissions have closed""")

6) Checkoff

Once you are finished with the code, please come to a tutorial, lab session, or office hour and add yourself to the queue asking for a checkoff. **You must be ready to discuss your code in detail before asking for a checkoff.**

You should be prepared to demonstrate your code (which should be well-commented, should avoid repetition, and should make good use of helper functions). In particular, be prepared to discuss:

- The object-oriented design you came up with to make extension pleasant
- Code tour of your favorite required feature, whose implementation best shows off your high-level design
- Explanation and code tour for the new feature you came up with

Grade

Grading:

- Concept questions (0 points): 0
- Tests (2 points): 2
- Checkoff (2 points): 2

Total: 4 Points (of 4)