159.172 Computational Thinking

Programming Assignment 2: The Hungry Caterpillar

This assignment is worth 15% of your final mark. It will be marked out of a total of 20 marks. You are expected to work on this assignment individually and all work that you hand in is expected to be your *own* work.

In this assignment you will develop an animated application that uses **`linked lists`** as the basis of the implementation. This application builds somewhat on the work that you did for tutorial 4.

Go to Stream and download the files

`caterpillar.py`
`my_catclass.py`

Set up a new project and add these files to it.

caterpillar.py is a program that implements the beginnings of a simple graphical animation. When you run this program, you will see a screen, 1000 pixels wide by 400 pixels high, displaying a basic background scene with a 'caterpillar' anchored at random location. To begin with, the caterpillar just has a face, no body parts.

Similar to the application that you worked with in tutorial 4, on each iteration of the main program loop we check to see if the user wants to quit, or if he has pressed the spacebar, or one of the 'm', 'd' or 's' keys, each of which initiates a different response (full functionality is described later.) After that, we update the screen by redrawing the background scene and the caterpillar. The drawing methods for the background and the caterpillar have been provided for you.

The module *my_catclass.py*, which is imported into *caterpillar.py*, implements a linked list based implementation for a caterpillar object. The caterpillar class is expanded from the class that you developed in the tutorial.

Now, a caterpillar consists of the following components:

1. *face_xcoord* and *face_ycoord* - horizontal and vertical coordinates for the face. These coordinates are used by the draw_face(screen) method to give the location of the top left boundary of the graphic that represents the caterpillar's face.
2. *body* – a reference to a segment_queue object. A segment_queue is a linked list of body_segment nodes that makes up the caterpillar's body. A segment_queue has head and last references, just like the Improved Linked Queue class shown in lectures. A

body_segment(x, y) object consists of x and y coordinates, giving the location of the top left boundary of the graphic representing the segment; and a reference to the next body_segment in the list.

3. *travel_direction* – a string representing the current direction of travel for the caterpillar, either 'left' or 'right'.
4. *food* – a reference to a food_list object. A food_list is a linked list of food_item nodes that makes up the caterpillar's food. A food_item(x, y, kind) object consists of a string representing the foodtype, either 'nice' or 'nasty'; x and y coordinates, giving the location of the top left boundary of the graphic representing the item; and a reference to the next fooditem in the list.

We create an instance of a caterpillar object in our main program *caterpillar.py* with the code:

```
mycaterpillar = my_catclass.caterpillar()
```

Tasks

- You first need to program code for the following three methods in the caterpillar class:

1.
```
def grow(self):
    ...
```

   This method adds a body segment to the rear of a caterpillar. Pressing the spacebar will invoke this method on the *mycaterpillar* object.

   To implement this method, you first need to attend to the segment_queue class. Inside the segment_queue class, you need to provide a method to add a body_segment, with given x and y parameters, to a segment_queue. At the moment this function doesn't do anything.

```
def addSegment(self, x, y):
    return
```

   You should look at the lecture slides and pay particular attention to those that show how to insert a cargo item at the end of a queue, using the Improved Linked Queue implementation.

   Inside the caterpillar class, your grow() method will require a call to the addSegment(x, y) method for the caterpillar body, with correct location parameters. Note that the x coordinate for the new body segment will depend on the *travel direction* of the caterpillar and also on whether or not the segment is the first one to be added. When the body is empty the new segment should be placed relative to the caterpillar's face, otherwise it should be placed relative to the last body segment. Also note that the face graphic is 40 pixels wide, while a body segment graphic is only 35 pixels wide. The y coordinate for the new segment can be taken to be the same as the y coordinate of the caterpillar's face.

2.
```
def reverse(self):
    ...
```

   This method sets up the caterpillar to move in the opposite direction. This entails reversing the travel direction, reversing the linked queue that stores the body segment coordinates, and then "attaching" the face at the other end. Don't forget that the face graphic is 40 pixels wide, while a body segment graphic is only 35 pixels wide. Along with the travel direction, this will affect the calculation of the new x coordinate for the face.

3.
```
def move_forward(self):
    ...
```

   This method moves the caterpillar forward one step, either to the left or the right, depending on the travel direction. You will get a fairly smooth animation if you set a step to equal 2 pixels. You can presume that a caterpillar without a body can't move, so first test for an empty body. If one more step forward would send the caterpillar out of the scene, you need to instead have the caterpillar reverse. Note that the scene ranges from 0 to 1000 pixels, 0 at the left edge, 1000 at the right edge. Pressing the 'm' key will invoke this method on the *mycaterpillar* object.

Once you have these three methods completed, you can grow the caterpillar for a few segments, by pressing the space bar, and then press the 's' key; your caterpillar should run back and forth across the scene, reversing when it hits either edge of the scene. Press the 's' key again to stop the animation (the main loop will revert to its first conditional block, invoked when the Boolean variable *start_anim* is False.)

- When you have got this part of the program working properly, the next thing to do is to program code for the following method:

```
def drop_food(self):
    ...
```

  This method should drop some food at a random location in the scene, in other words, add a food_item to the food_list that makes up the caterpillar's food.

  First, generate a random x coordinate and a random food type for the new food item. The x coordinate for the new food item can be computed by the method random.randrange(0, 980). Note that the generator for the x coordinate should range from 0 to 980, rather than 0 to 1000, since the x coordinate gives the location of the top left corner of the food graphic. The y coordinate for the new food item should be calculated relative to the caterpillar's face_ycoord. The generator for the food type should range from 0 to 2, if the result is 0 the food type is 'nice', if the result is 1 the food type is 'nasty'.

  Once you have this method working, you can press the 'd' key to add some food to the caterpillar's food list, and have it displayed in the scene. You will also need to extend the move_forward() method so that once the caterpillar has moved forward, the method checks to see if his face has come within 5 pixels of any food item.

checks to see if his face has come within 5 pixels of any food item.

The move_forward() method should delete any food item in the food list that is within 5 pixels of the caterpillar's face. Use the abs() function, applied to the difference of the x coordinates of the caterpillar's face and each food item, to check distances. You should look at the lecture slides and pay particular attention to those that show how to remove 'bad' items from a linked list.

- Your final task is to program the code to make the caterpillar react to food.

  When you have got the caterpillar moving properly and eating his food, you need to have him react to the two different types of food. Add a 'wellbeing' attribute to the caterpillar class. This should be an integer, set to zero upon initialization. If he eats some 'nice' food the wellbeing property is increased, if he eats some 'nasty' food the wellbeing property is decreased. If 'wellbeing' is above 1, the caterpillar is blooming with health and his body should turn yellow, if the wellbeing property is below -1 the caterpillar is now 'poisoned' and so his body should become striped - alternate body segments should be coloured purple and black. In addition, he should 'shrink back'. You will need to implement the following method:

  ```
  def shrink_back(self):
      ...
  ```

  This method removes the first segment of the caterpillar body and repositions his face. Placement of the face will depend on the travel direction. The first body segment must be removed from the body segment_queue.

  The colour changes can be implemented by altering the provided methods draw_body() and draw_segment(). You will need to consider the wellbeing attribute when re-implementing the draw_body() method and you will need to alter the draw_segment() method so that it is passed a colour parameter. You may want to change your code so that the foodtype attribute of each food item is set to either 'nice' or 'nasty' absolutely, rather than randomly, while you test your colour change code.

  Once you have these methods completed, you can grow the caterpillar for a few segments, by pressing the space bar, and then drop some food for him, by pressing the 'd' key, and then press the 's' key; your caterpillar should run back and forth across the scene, reversing when it hits either edge of the scene, any food eaten should disappear and the caterpillar should react appropriately to the food eaten. Press the 's' key again to stop the animation.

Once you have the animation working properly, you may like to try one of the following optional extensions.

- Consider how real caterpillars move, in a wave-like motion from front to back.
  See You Tube Close Up Caterpillar Footage
  Can you get your caterpillar to move forward using a more natural motion? You will need to consider both the x and y coordinates of the face and each body segment.
-

- Another option is to consider how to alter the program so that it can sensibly deal with more than one caterpillar at a time. Your group (or pair) of caterpillars should share food and will need to be able to access the locations of one another, so the caterpillar class will need to be altered. One possibility - upon two caterpillars meeting, have the longer of the two climb over the shorter. No X-rated caterpillar activity!

Submission

Submit the assignment in Stream as a single zipped file containing:

1. Your completed code, contained in the file *my_catclass.py*.
2. A word document containing annotated screen shots demonstrating the behaviour of your program.
3. If you have attempted any extension to the project, submit your extended code as separate code modules, and include a separate word document demonstrating the extended program behavior. This will not (necessarily) contribute to your final mark, but imaginative solutions are welcomed!

Marking Scheme:

- Implementation of each of the first three methods: **`grow()`**, **`reverse()`** and **`move_forward()`** - 4 marks each
- Implementation of the **`drop_food()`** method - 2 marks
- Implementation of correct reaction to food - 4 marks
- Annotated screenshots demonstrating the behaviour of your program - 2 marks
- **Total – 20 marks**

Late submission:

Late assignments will be penalised 10% for each weekday past the deadline, for up to five (5) days; after this no marks will be gained. In special circumstances an extension may be obtained from the paper co-ordinator, and these penalties will not apply. Workload will not be considered a special circumstance – you must budget your time.