BIS 420 PROGRAMMING FOR DATA SCIENCE

PRAJAKTA POHARE

CHAPTER 18 EXERCISE 18.7

ILLINOIS STATE UNIVERSITY

This exercise uses TurtleWorld from Chapter 4. You will write code that makes Turtles play tag. If you are not familiar with the rules of tag, see http: // en. wikipedia. org/ wiki/ Tag_ (game) .

1. Download http: // thinkpython. com/ code/ Wobbler. py and run it. You should see a TurtleWorld with three Turtles. If you press the Run button, the Turtles wander at random.
2. Read the code and make sure you understand how it works. The Wobbler class inherits from Turtle, which means that the Turtle methods lt, rt, fd and bk work on Wobblers.
The step method gets invoked by TurtleWorld. It invokes steer, which turns the Turtle in the desired direction, wobble, which makes a random turn in proportion to the Turtle's clumsiness, and move, which moves forward a few pixels, depending on the Turtle's speed.
3. Create a file named Tagger.py. Import everything from Wobbler, then define a class named Tagger that inherits from Wobbler. Call make_world passing the Tagger class object as an argument.
4. Add a steer method to Tagger to override the one in Wobbler. As a starting place, write a version that always points the Turtle toward the origin. Hint: use the math function atan2 and the Turtle attributes x, y and heading.
5. Modify steer so that the Turtles stay in bounds. For debugging, you might want to use the Step button, which invokes step once on each Turtle.
6. Modify steer so that each Turtle points toward its nearest neighbor. Hint: Turtles have an attribute, world, that is a reference to the TurtleWorld they live in, and the TurtleWorld has an attribute, animals, that is a list of all Turtles in the world.
7. Modify steer so the Turtles play tag. You can add methods to Tagger and you can override

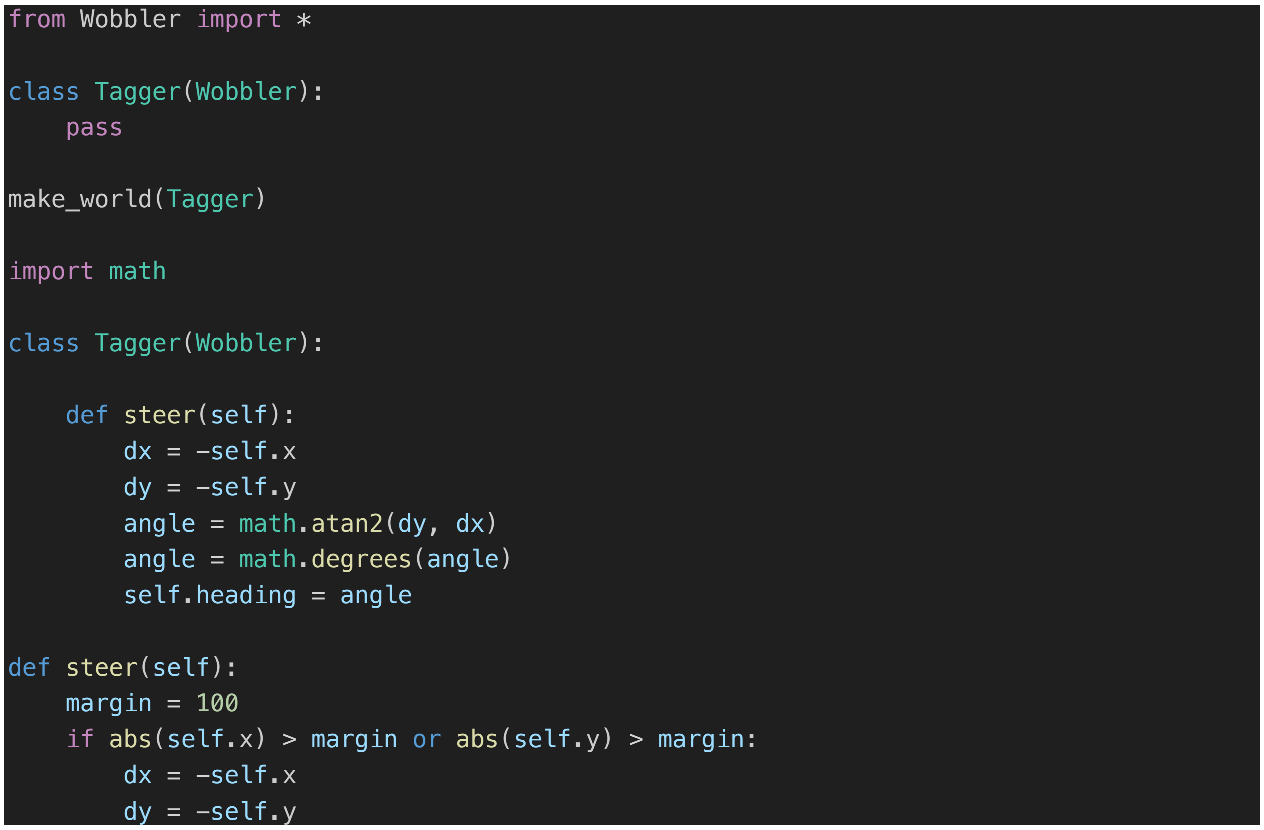
steer and __init__, but you may not modify or override step, wobble or move. Also, steer is allowed to change the heading of the Turtle but not the position.
Adjust the rules and your steer method for good quality play; for example, it should be possible for the slow Turtle to tag the faster Turtles eventually.

```
from Wobbler import *


class Tagger(Wobbler):
    pass


make_world(Tagger)


import math


class Tagger(Wobbler):


    def steer(self):
        dx = -self.x
        dy = -self.y
        angle = math.atan2(dy, dx)
        angle = math.degrees(angle)
        self.heading = angle


def steer(self):
    margin = 100
    if abs(self.x) > margin or abs(self.y) > margin:
        dx = -self.x
```

```
            dy = -self.y
        else:
            dx = -self.x
            dy = -self.y

        angle = math.atan2(dy, dx)
        self.heading = math.degrees(angle)

    def steer(self):
        nearest = None
        min_dist = float('inf')

        for other in self.world.animals:
            if other == self:
                continue
            dx = other.x - self.x
            dy = other.y - self.y
            dist = math.hypot(dx, dy)
            if dist < min_dist:
                min_dist = dist
                nearest = other

        if nearest:
            dx = nearest.x - self.x
            dy = nearest.y - self.y
            angle = math.degrees(math.atan2(dy, dx))
            self.heading = angle
```

```
class Tagger(Wobbler):

    def __init__(self):
        Wobbler.__init__(self)
        self.color = 'Black'

    def steer(self):
        # If I am "it", chase others
        if self.color == 'Red':
            nearest = self.find_nearest()
            if nearest and self.distance(nearest) < 10:
                nearest.color = 'Red'
                self.color = 'Black'
            else:
                self.face(nearest)
        else:
            # If not "it", run away from the one who is "it"
            chaser = self.find_chaser()
            if chaser:
                self.run_from(chaser)

    def find_nearest(self):
        nearest = None
        min_dist = float('inf')
        for other in self.world.animals:
            if other == self:
                continue
            d = self.distance(other)
            if d < min_dist:
                min_dist = d
                nearest = other
        return nearest

    def find_chaser(self):
        for other in self.world.animals:
            if other != self and other.color == 'Red':
                return other
        return None

    def run_from(self, other):
        dx = self.x - other.x
        dy = self.y - other.y
        angle = math.atan2(dy, dx)
        self.heading = math.degrees(angle)

    def face(self, other):
        dx = other.x - self.x
        dy = other.y - self.y
        angle = math.atan2(dy, dx)
        self.heading = math.degrees(angle)

from Wobbler import *
```

```
class Tagger(Wobbler):
    def steer(self):
        # Always steer toward the origin for now
        dx = -self.x
        dy = -self.y
        angle = math.atan2(dy, dx) * 180 / math.pi
        turn = angle - self.heading
        self.rt(turn)

world = make_world(Tagger)
world.animals[0].color = 'Red'  # First one is 'It'
wait_for_user()
```



```
from Wobbler import *

class Tagger(Wobbler):
    pass

make_world(Tagger)

import math

class Tagger(Wobbler):

    def steer(self):
        dx = -self.x
        dy = -self.y
        angle = math.atan2(dy, dx)
        angle = math.degrees(angle)
        self.heading = angle

def steer(self):
    margin = 100
    if abs(self.x) > margin or abs(self.y) > margin:
        dx = -self.x
        dy = -self.y
```

```python
    else:
        dx = -self.x
        dy = -self.y

    angle = math.atan2(dy, dx)
    self.heading = math.degrees(angle)

def steer(self):
    nearest = None
    min_dist = float('inf')

    for other in self.world.animals:
        if other == self:
            continue
        dx = other.x - self.x
        dy = other.y - self.y
        dist = math.hypot(dx, dy)
        if dist < min_dist:
            min_dist = dist
            nearest = other

    if nearest:
        dx = nearest.x - self.x
        dy = nearest.y - self.y
        angle = math.degrees(math.atan2(dy, dx))
        self.heading = angle

class Tagger(Wobbler):

    def __init__(self):
        Wobbler.__init__(self)
        self.color = 'Black'

    def steer(self):
        # If I am "it", chase others
        if self.color == 'Red':
            nearest = self.find_nearest()
            if nearest and self.distance(nearest) < 10:
                nearest.color = 'Red'
                self.color = 'Black'
            else:
                self.face(nearest)
        else:
            # If not "it", run away from the one who is "it"
            chaser = self.find_chaser()
            if chaser:
                self.run_from(chaser)
    def find_nearest(self):
        nearest = None
        min_dist = float('inf')
        for other in self.world.animals:
            if other == self:
                continue
            d = self.distance(other)
            if d < min_dist:
                min_dist = d
                nearest = other
        return nearest

    def find_chaser(self):
        for other in self.world.animals:
            if other != self and other.color == 'Red':
                return other
        return None

    def run_from(self, other):
        dx = self.x - other.x
        dy = self.y - other.y
        angle = math.atan2(dy, dx)
        self.heading = math.degrees(angle)

    def face(self, other):
        dx = other.x - self.x
        dy = other.y - self.y
        angle = math.atan2(dy, dx)
        self.heading = math.degrees(angle)

from Wobbler import *

class Tagger(Wobbler):
    def steer(self):
        # Always steer toward the origin for now
        dx = -self.x
        dy = -self.y
        angle = math.atan2(dy, dx) * 180 / math.pi
        turn = angle - self.heading
        self.rt(turn)

world = make_world(Tagger)
world.animals[0].color = 'Red'  # First one is 'It'
wait_for_user()
```