Homework Turnin

Name:	Riley H Taylor
Email:	rileytaylor@email.arizona.edu
Section:	2J
Course:	CS 110 17sp
Assignment:	hw12

Warning: The email ID you entered, 'rileytaylor@email.arizona.edu', is not found in this course's list of students. You may have entered your email address incorrectly. Please double-check your information to make sure you have the right address; if not, or if you don't get an email receipt of your turnin, you may want to turn in again. If you believe you have received this message in error, please contact your TA or instructor. (Our list of student email addresses was last updated on Thu 2017/04/27 08:42pm and contains 1 records. If you added the course after that, we may need to update our records.)

Receipt ID: f12cdae13de91c41901546f9ebaea8cf

Warning: Your turnin is 3 days late. Assignment hw12 was due Friday, April 28, 2017, 7:00 PM.

Turnin Successful!

The following file(s) were received:

ant.py (909 bytes)

```
# This class represents a Ant type of Critter
# Either goes south or north diagonally, always eats.
# it is represented by a red %

# HW 12 Hours spent: 5

from critter import *


class Ant(Critter):
    def __init__(self, walk_south):
        super(Ant, self).__init__()
        self.__walk_south = walk_south
        self.__moves = 0

    def eat(self):
        return True

    def fight(self, opponent):
        return ATTACK_SCRATCH

    def get_color(self):
        return "red"

    def get_move(self):
        self.__moves += 1
        if self.__walk_south:
            if self.__moves % 2 == 0:
                return DIRECTION_EAST
            else:
                return DIRECTION_SOUTH
        if not self.__walk_south:
            if self.__moves % 2 == 0:
                return DIRECTION_EAST
            else:
                return DIRECTION_NORTH

    def __str__(self):
        return "%"
```

bird.py (1302 bytes)

```
# This class represents a Bird type of Critter
# It moves in a square, and really hates ants.
# it is represented by a blue arrow.

from critter import *


class Bird(Critter):
    def __init__(self):
        super(Bird, self).__init__()
        self.__moves = 0
        self.__last_move = 'N'

    def eat(self):
        return False

    def fight(self, opponent):
        if opponent == "%":
            return ATTACK_ROAR
        else:
            return ATTACK_POUNCE

    def get_color(self):
        return "blue"

    def get_move(self):
        self.__moves += 1
        if self.__moves > 12:
            self.__moves = 1
        if self.__moves <= 3:
            self.__last_move = "N"
            return DIRECTION_NORTH
        elif self.__moves <= 6:
            self.__last_move = "E"
            return DIRECTION_EAST
        elif self.__moves <= 9:
            self.__last_move = "S"
            return DIRECTION_SOUTH
        elif self.__moves <= 12:
            self.__last_move = "W"
            return DIRECTION_WEST

    def __str__(self):
        if self.__last_move == "N":
            return "^"
        elif self.__last_move == "E":
            return ">"
        elif self.__last_move == "S":
            return "V"
        elif self.__last_move == "W":
            return "<"
```

hippo.py (1421 bytes)

```
# This class represents a Stone type of Critter
# It changes direction after 5 moves, and only eats X amount
# of times during its life.
# it is represented by a gray S

from critter import *
from random import *


class Hippo(Critter):
    def __init__(self, hunger):
        super(Hippo, self).__init__()
        self.__hunger = hunger
        self.__move_count = 0
        self.__move_dir = 0

    def eat(self):
        if self.__hunger != 0:
            self.__hunger -= 1
            return True
        return False

    def fight(self, opponent):
        if self.__hunger > 0:
            return ATTACK_SCRATCH
        return ATTACK_POUNCE

    def get_color(self):
        if self.__hunger > 0:
            return "gray"
        return "white"

    def get_move(self):
        # If we've moved 5 times, time to go another way
        if self.__move_count > 5:
            self.__move_count = 0
        # If the move count is reset, generate a new number and direction
        if self.__move_count == 0:
            self.__move_dir = randint(1, 4)
        if self.__move_dir == 1:
            return DIRECTION_NORTH
        elif self.__move_dir == 2:
            return DIRECTION_EAST
        elif self.__move_dir == 3:
            return DIRECTION_SOUTH
        elif self.__move_dir == 4:
            return DIRECTION_WEST
        self.__move_count += 1

    def __str__(self):
        return str(self.__hunger)
```

vulture.py (572 bytes)

```python
# This class represents a Vulture type of Critter
# This is a bird that is only hungry after fighting.
# it is represented by a black arrow.

from bird import *


class Vulture(Bird):
    def __init__(self):
        super(Vulture, self).__init__()
        self.__hungry = True

    def eat(self):
        hunger = self.__hungry
        if self.__hungry:
            self.__hungry = False
        return hunger

    def fight(self, opponent):
        self.__hungry = True
        return super(Vulture, self).fight(opponent)

    def get_color(self):
        return "black"
```

wildcat.py (1814 bytes)

```python
# A Wildcat. Yay.
# It's represented by a cyan &.
# I wanted to build in a learning mechanism for dealing with the other
# wildcat, but the time required without using 3rd-party libraries turned
# out to be not worth it. Oh well, it still cleans house of the other types.

from critter import *
from random import randint

directions = DIRECTION_EAST, DIRECTION_WEST, DIRECTION_SOUTH, DIRECTION_NORTH
attacks = ATTACK_POUNCE, ATTACK_SCRATCH, ATTACK_ROAR

class Wildcat(Critter):
    def __init__(self):
        super(Wildcat, self).__init__()

    def eat(self):
        neighbor = False
        for d in directions:
            if self.get_neighbor(directions[d]) != " ":
                neighor = True
        if neighbor:
            return False
        return True

    def fight(self, opponent):
        self.cat_was_last_opponent = False
        if opponent == "%":
            return ATTACK_ROAR
        elif opponent == "S":
            return ATTACK_POUNCE
        elif opponent == "^" or "<" or ">" or "V":
            return ATTACK_SCRATCH
        elif int(opponent) == 1 or 2 or 3 or 4 or 5:
            return ATTACK_ROAR
        elif int(opponent) == 0:
            return ATTACK_SCRATCH
        elif opponent == "a" or "A":
            # Best chance against the Aardvark
            return ATTACK_POUNCE
        else:
            r = randint(0,2)
            return attacks[r]

    def get_color(self):
        return "cyan"

    def get_move(self):
        # just move to the first empty space, this wildcats a loner
        move = False
        for d in directions:
            if self.get_neighbor(directions[d]) == " ":
                move = True
                return directions[d]
        if not move:
            return directions[randint(0, 3)]

    def __str__(self):
        return "&"
```