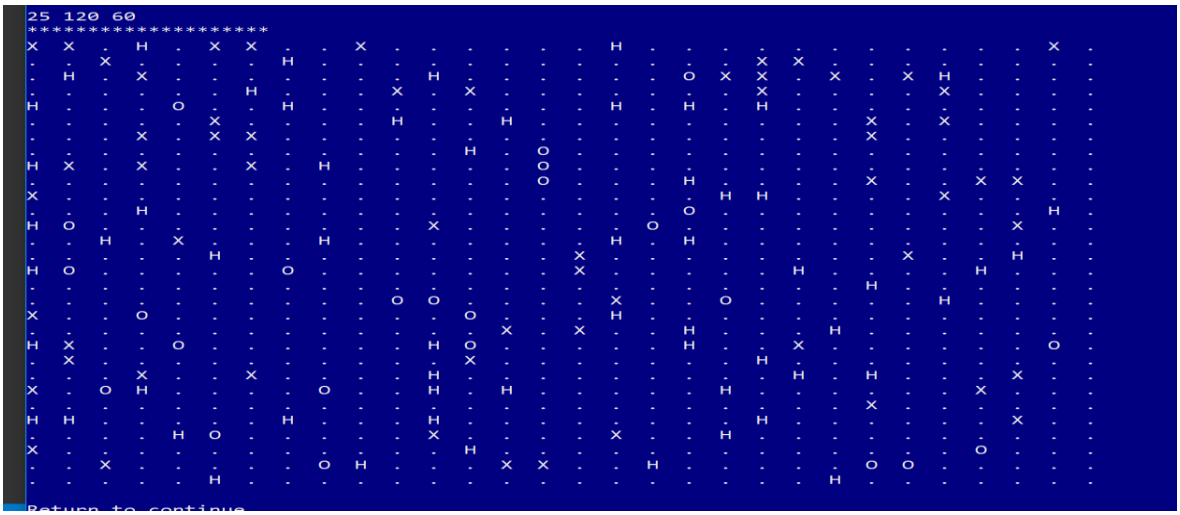
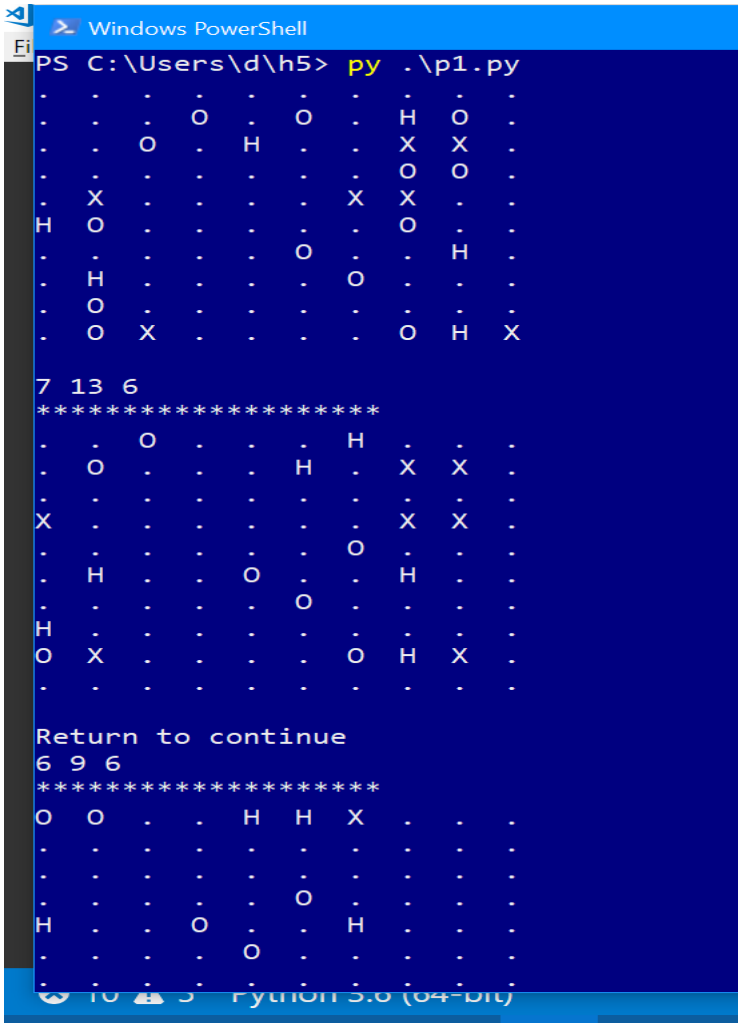
Homework 5 Problem-1

Pred-Prey-Humans





```
# Predator-Prey Simulation.
''' Adding to Island class for Human.
    Tested main: Grid and worked as well at start
'''

import random
import time
import pylab


class Island (object):
    """Island n X n grid where zero value indicates not occupied.
        For Human class: count_humans(), init_animals() adjustment
    """

    def __init__(self, n, prey_count=0, predator_count=0, human_count=0):
        '''Initialize grid to all 0's, then fill with animals'''
        # print(n,prey_count,predator_count)
        self.grid_size = n
        self.grid = []
        for i in range(n):
            row = [0]*n    # row is a list of n zeros
            self.grid.append(row)
        self.init_animals(prey_count, predator_count, human_count)

    def init_animals(self, prey_count, predator_count, human_count):
        '''Put some initial animals on the island'''
        # while loop continues until prey_count unoccupied positions are found
        count = 0
        while count < prey_count:
            x = random.randint(0, self.grid_size-1)
            y = random.randint(0, self.grid_size-1)
            if not self.animal(x, y):
                new_prey = Prey(island=self, x=x, y=y)
                count += 1
                self.register(new_prey)
        # predator_count
        count = 0
        while count < predator_count:
            x = random.randint(0, self.grid_size-1)
            y = random.randint(0, self.grid_size-1)
            if not self.animal(x, y):
                new_predator = Predator(island=self, x=x, y=y)
                count += 1
                self.register(new_predator)
        # Human_count
        count = 0
        while count < human_count:
            x = random.randint(0, self.grid_size-1)
            y = random.randint(0, self.grid_size-1)
```

```
            if not self.animal(x, y):
                new_human = Human(island=self, x=x, y=y)
                count += 1
                self.register(new_human)

    def clear_all_moved_flags(self):
        ''' Animals have a moved flag to indicated they moved this turn.
        Clear that so we can do the next turn
        '''
        for x in range(self.grid_size):
            for y in range(self.grid_size):
                if self.grid[x][y]:
                    self.grid[x][y].clear_moved_flag()

    def size(self):
        '''Return size of the island: one dimension.
        '''
        return self.grid_size

    def register(self, animal):
        '''Register animal with island, i.e. put it at the
        animal's coordinates
        '''
        x = animal.x
        y = animal.y
        self.grid[x][y] = animal

    def remove(self, animal):
        '''Remove animal from island.'''
        x = animal.x
        y = animal.y
        self.grid[x][y] = 0

    def animal(self, x, y):
        '''Return animal at location (x,y)'''
        if 0 <= x < self.grid_size and 0 <= y < self.grid_size:
            return self.grid[x][y]
        else:
            return -1  # outside island boundary

    def __str__(self):
        '''String representation for printing.
           (0,0) will be in the lower left corner.
        '''
        s = ""
        for j in range(self.grid_size-1, -1, -1):  # print row size-1 first
            for i in range(self.grid_size):      # each row starts at 0
                if not self.grid[i][j]:
                    # print a '.' for an empty space
                    s += "{:<2s}".format('.' + "  ")
```

```
                else:
                    s += "{:<2s}".format((str(self.grid[i][j])) + "  ")
            s += "\n"
        return s

    def count_prey(self):
        ''' count all the prey on the island'''
        count = 0
        for x in range(self.grid_size):
            for y in range(self.grid_size):
                animal = self.animal(x, y)
                if animal:
                    if isinstance(animal, Prey):
                        count += 1
        return count

    def count_predators(self):
        ''' count all the predators on the island'''
        count = 0
        for x in range(self.grid_size):
            for y in range(self.grid_size):
                animal = self.animal(x, y)
                if animal:
                    if isinstance(animal, Predator):
                        count += 1
        return count

    # Added this iteration for Humans_count
    def count_human(self):
        ''' count all the human on the island'''
        count = 0
        for x in range(self.grid_size):
            for y in range(self.grid_size):
                animal = self.animal(x, y)
                if animal:
                    if isinstance(animal, Human):
                        count += 1
        return count


class Animal(object):
    def __init__(self, island, x=0, y=0, s="A"):
        '''Initialize the animal's and their positions'''
        self.island = island
        self.name = s
        self.x = x
        self.y = y
        self.moved = False

    def position(self):
```

```
        '''Return coordinates of current position.
        '''
        return self.x, self.y

    def __str__(self):
        return self.name

    def check_grid(self, type_looking_for=int):
        ''' Look in the 8 directions from the animal's location
        and return the first location that presently has an object
        of the specified type. Return 0 if no such location exists
        '''
        # neighbor offsets
        offset = [(-1, 1), (0, 1), (1, 1), (-1, 0),
                  (1, 0), (-1, -1), (0, -1), (1, -1)]
        result = 0
        for i in range(len(offset)):
            x = self.x + offset[i][0]  # neighboring coordinates
            y = self.y + offset[i][1]
            if not 0 <= x < self.island.size() or \
               not 0 <= y < self.island.size():
                continue
            if type(self.island.animal(x, y)) == type_looking_for:
                result = (x, y)
                break
        return result

    def move(self):
        '''Move to an open, neighboring position '''
        if not self.moved:
            location = self.check_grid(int)
            if location:
                self.island.remove(self)   # remove from current spot
                self.x = location[0]       # new coordinates
                self.y = location[1]
                self.island.register(self)  # register new coordinates
                self.moved = True

    def breed(self):
        ''' Breed a new Animal.If there is room in one of the 8 locations
        place the new Prey there. Otherwise you have to wait.
        '''
        if self.breed_clock <= 0:
            location = self.check_grid(int)
            if location:
                self.breed_clock = self.breed_time
                # print('Breeding Prey {},{}'.format(self.x, self.y))
                the_class = self.__class__
                new_animal = the_class(
                    self.island, x=location[0], y=location[1])
```

```
                self.island.register(new_animal)

    def clear_moved_flag(self):
        self.moved = False


class Prey(Animal):
    def __init__(self, island, x=0, y=0, s="O"):
        Animal.__init__(self, island, x, y, s)
        self.breed_clock = self.breed_time

    def clock_tick(self):
        '''Prey only updates its local breed clock
        '''
        self.breed_clock -= 1


class Predator(Animal):
    def __init__(self, island, x=0, y=0, s="X"):
        Animal.__init__(self, island, x, y, s)
        self.starve_clock = self.starve_time
        self.breed_clock = self.breed_time

    def clock_tick(self):
        ''' Predator updates both breeding and starving
        '''
        self.breed_clock -= 1
        self.starve_clock -= 1
        if self.starve_clock <= 0:
            self.island.remove(self)

    def eat(self):
        ''' Predator looks for one of the 8 locations with Prey. If found
        moves to that location, updates the starve clock, removes the Prey
        '''
        if not self.moved:
            location = self.check_grid(Prey)
            if location:
                self.island.remove(self.island.animal(location[0], location[1]))
                self.island.remove(self)
                self.x = location[0]
                self.y = location[1]
                self.island.register(self)
                self.starve_clock = self.starve_time
                self.moved = True


class Human(Predator):
    '''Implements a hunting clock.
       Inherits eat(), move(), breed(), clock_tick()'''
```

```
    def __init__(self, island, x=0, y=0, s="H"):
        Predator.__init__(self, island, x, y, s)
        self.starve_clock = self.starve_time
        self.breed_clock = self.breed_time
        self.hunt_clock = self.hunt_time

    def hunt(self):
        '''Using functions in Animal like checkgrid/remove'''
        self.hunt_clock -= 1
        if self.hunt_clock <= 0:
            if not self.moved:
                location = self.check_grid(Predator)
                if location:
                    self.island.remove(self.island.animal(location[0], location[1]))
                    self.island.remove(self)
                    self.x = location[0]
                    self.y = location[1]
                    self.island.register(self)
                    self.hunt_clock = self.hunt_clock
                    self.moved = True


def main(predator_breed_time=6, predator_starve_time=3, initial_predators=7,
         prey_breed_time=3, initial_prey=13, size=10, ticks=300,
         human_breedtime=9, human_starve_time=5, hunt_time=5, init_humans=6):
    '''Event Loop, counting, and graphing in progress'''

    # Time Quantities for ClassVariables
    Predator.breed_time = predator_breed_time
    Predator.starve_time = predator_starve_time
    Prey.breed_time = prey_breed_time
    Human.breed_time = human_breedtime
    Human.starve_time = human_starve_time
    Human.hunt_time = hunt_time

    # Construct island
    isle = Island(size, initial_prey, initial_predators, init_humans)
    print(isle)

    # Defined outside the event loop for consistency
    predator_list = []
    prey_list = []
    human_list = []

    # Event loop manages the ticks, for every x,y location.
    # If there is an animal there, try eat, move, breed and clock_tick
    for i in range(ticks):
        isle.clear_all_moved_flags()
        for x in range(size):
            for y in range(size):
```

```
                animal = isle.animal(x, y)
                if animal:
                    if isinstance(animal, Human) or \
                       isinstance(animal, Predator):
                        animal.eat()
                    animal.move()
                    animal.breed()
                    animal.clock_tick()
                    if isinstance(animal, Human):
                        animal.hunt()

        # Display during Event-Loop
        prey_count = isle.count_prey()
        predator_count = isle.count_predators()
        human_count = isle.count_human()
        if prey_count == 0:
            print('Lost the Prey population. Quiting.')
            break
        prey_list.append(prey_count)
        predator_list.append(predator_count)
        human_list.append(human_count)
        # print out every 10th cycle
        # if not i % 10:
        print(prey_count, predator_count, human_count)
        # print the island
        print('*'*20)
        print(isle)
        ans = input("Return to continue")

    # Graphing after the Event-Loop
    ticks = 150
    pylab.plot(range(0, ticks, 2), tuple(predator_list), label="Predators")
    pylab.plot(range(0, ticks, 2), tuple(human_list), label="Humans")
    pylab.plot(range(0, ticks, 2), tuple(prey_list), label="Prey")
    pylab.legend(loc="best", shadow=True)
    pylab.show()

if __name__ == '__main__':
    main()
```