Appendix 1

Code for the simulation of prey predator dynamics with random topology

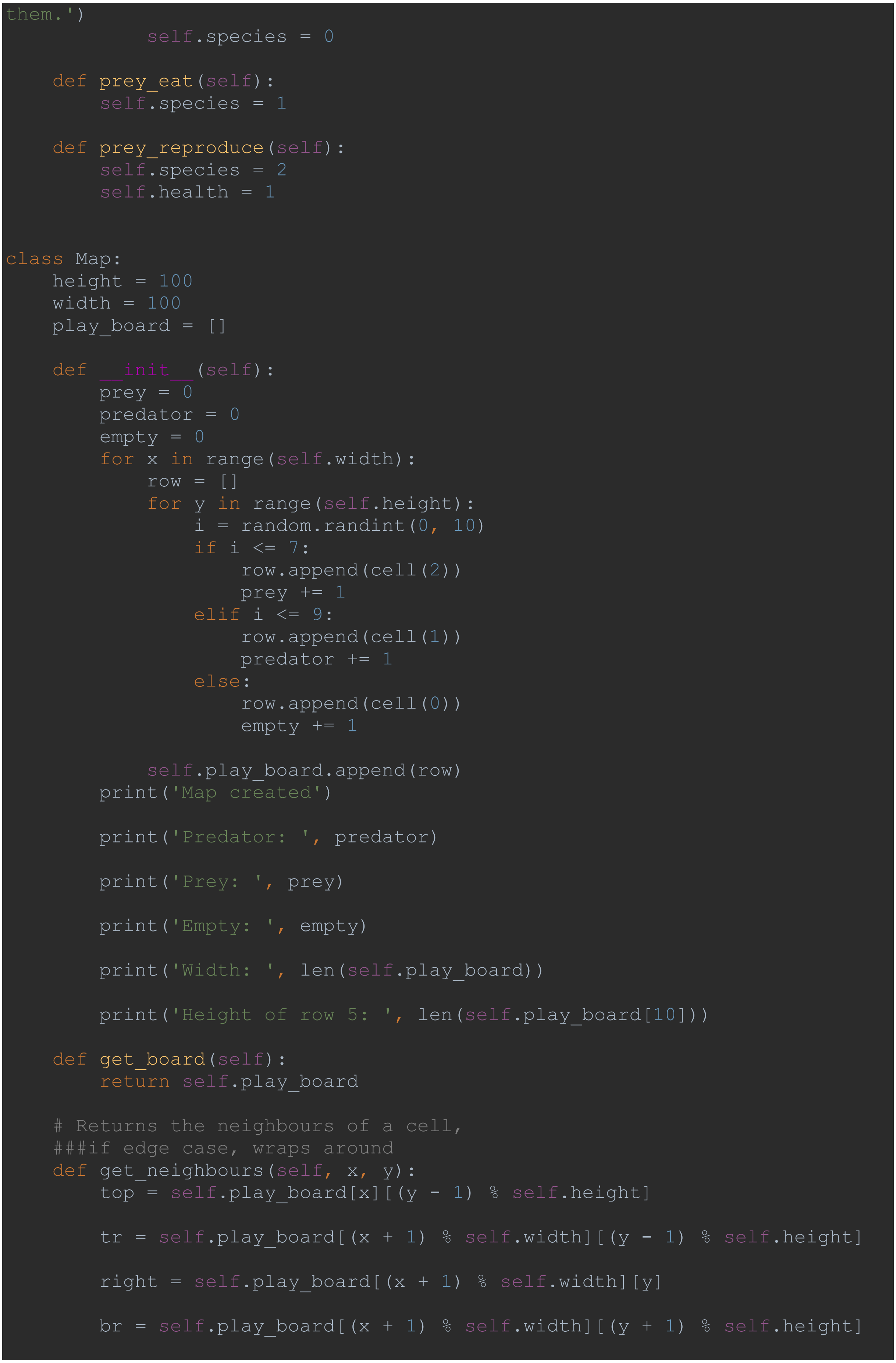
Authored by: Ishleen

Application.py

```
import random
import sys
import pygame
from threading import Thread
import Randompp

random.seed(None)
size = width, height = 500, 500
play_width = width / 5
play_height = height / 5
stoop = False
# Pygame codesfor for colours
red = 16386570
blue = 658170
black = 0
colour = [black, red, blue]
pygame.init()
grid = pygame.display.set_mode(size)
pp = Randompp.Map()


# Paints a 3x3 square for each node
def colour_ca(x, y, code):
    grid.set_at((x, y), colour[code])
    grid.set_at((x + 1, y), colour[code])
    grid.set_at((x, y + 1), colour[code])
    grid.set_at((x + 1, y + 1), colour[code])
    grid.set_at((x + 2, y), colour[code])
    grid.set_at((x, y + 2), colour[code])
    grid.set_at((x + 2, y + 2), colour[code])
    grid.set_at((x + 2, y + 1), colour[code])
    grid.set_at((x + 1, y + 2), colour[code])


# Colours the prey or predator species at each cell in the cellular
automata
def ca(grid_board):
    pa = colour_ca
    length = len(grid_board)
    for x in range(0, length):
        for y in range(0, length):
            pa(x * 5, y * 5, grid_board[x][y].species)


# Main code for random topology


while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            stoop = True
            sys.exit(0)
    ca(pp.get_board())
    pygame.display.flip()
    pp.turn()
```

Randompp.py

```
import random
import sys
import copy
random.seed(None)


class cell:

    species = None
    x = None
    y = None
    health = None

    def __init__(self, spec=None):
        if spec is not None:
            self.species = spec
        else:
            self.species = 0
        self.x = None
        self.y = None
        self.health = self.species*2

    def set_location(self, pos_x, pos_y):
        self.x = pos_x
        self.y = pos_y

    def set_species(self, num):
        if num == 0 or num == 1 or num == 2:
            self.species = num
            if num is 0:  # Blank
                self.health = None
            if num is 1:  # Predator
                self.health = 2
            if num is 2:  # Prey
                self.health = 4
        else:
            sys.stderr.write('Species not correct, do not do math with
them.')
            self.species = 0

    def move_here(self, species, health):
        if species == 0 or species == 1 or species == 2:
            self.species = species
            if species is 0:  # Blank
                self.health = health
            if species is 1:  # Predator
                self.health = health
            if species is 2:  # Prey
                self.health = health
        else:
            sys.stderr.write('Species not correct, do not do math with
```



```
them.')
            self.species = 0

    def prey_eat(self):
        self.species = 1

    def prey_reproduce(self):
        self.species = 2
        self.health = 1


class Map:
    height = 100
    width = 100
    play_board = []

    def __init__(self):
        prey = 0
        predator = 0
        empty = 0
        for x in range(self.width):
            row = []
            for y in range(self.height):
                i = random.randint(0, 10)
                if i <= 7:
                    row.append(cell(2))
                    prey += 1
                elif i <= 9:
                    row.append(cell(1))
                    predator += 1
                else:
                    row.append(cell(0))
                    empty += 1

            self.play_board.append(row)
        print('Map created')

        print('Predator: ', predator)

        print('Prey: ', prey)

        print('Empty: ', empty)

        print('Width: ', len(self.play_board))

        print('Height of row 5: ', len(self.play_board[10]))

    def get_board(self):
        return self.play_board

    # Returns the neighbours of a cell,
    ###if edge case, wraps around
    def get_neighbours(self, x, y):
        top = self.play_board[x][(y - 1) % self.height]

        tr = self.play_board[(x + 1) % self.width][(y - 1) % self.height]

        right = self.play_board[(x + 1) % self.width][y]

        br = self.play_board[(x + 1) % self.width][(y + 1) % self.height]
        bottom = self.play_board[x][(y + 1) % self.height]

        bl = self.play_board[(x - 1) % self.width][(y + 1) % self.height]

        left = self.play_board[(x - 1) % self.width][y]

        tl = self.play_board[(x - 1) % self.width][(y - 1) % self.height]

        neighbours = [top, tr, right, br, bottom, bl, left, tl]
        return neighbours

    def check_neighbors(self, x, y, cell1):
        # If the cell1 is empty no checks are performed
        if cell1.species is 0:
            return

        if cell1.species is 1:
            # This is a predator. Check for prey if no prey reduce
health/die
            neighbourhood = self.get_neighbours(x, y)
            prey = []
            open_ = []

            for n in neighbourhood:
                if n.species is 2:
                    prey.append(n)
                if n.species is 0:
                    open_.append(n)
            open_length = len(open_)

            if len(prey) is 0:
                if cell1.health > 0 and open_length > 0:
                    cell1.health -= 1
                    open_[random.randint(0, (open_length-1))].move_here(1,
cell1.health)
                    cell1.set_species(0)
                    return

            else:
                target = prey[random.randint(0, (len(prey)-1))]
                if cell1.health >= target.health:
                    target.prey_eat()

            if cell1.health is None or cell1.health <= 0:
                cell1.species = 0
                return

        if cell1.species is 2:
            # This is  checking the cellular automata for overpopulation
            neighbourhood = self.get_neighbours(x, y)

            empty_cell = []
            for m in neighbourhood:
                if m.species is 0:
                    empty_cell.append(m)
            count = len(empty_cell)
            if count >= 7 or count < 1:
                cell1.health -= 2
            if cell1.health is None or cell1.health is 0:
                cell1.species = 0
                return
            if cell1.health > 6 and len(empty_cell) >= 1:
                empty_cell[random.randint(0, (len(empty_cell))-
1)].prey_reproduce()
                cell1.health = 4
                return
            cell1.health += 1
    def turn(self):
        pb = self.play_board
        [[self.check_neighbors(x, y, pb[x][y]) for y in range(0,
self.height)] for x in range(0, self.width)]
```