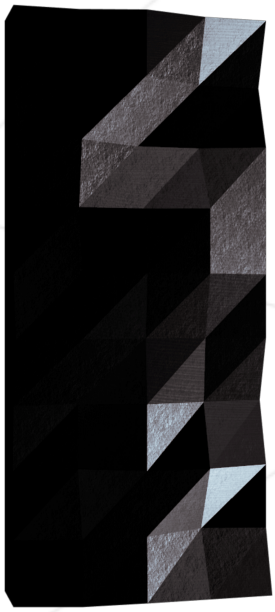

Plants vs. Non-Plants

p vs. np - any math geeks here who catch the pun?

Milan Ray raytmilan@gmail.com
Tapabrata Dey tapabratadey02@gmail.com
Wesley To wto@student.42.fr

Summary: A guide to creating a command-line style Plants vs. Zombies game in ***Python****.*

Contents

Chapter I

Introduction

This week you all learned how to use and apply the data structures of classes, queues, stacks, and linked lists. Today, your skills will be put to the tests in a one day final project that uses all the data structures you implemented over the week.

This final project is a command-line based clone of the popular Plants vs. Zombies game called, `Plants vs Non-Plants` (or PvNP for you math geeks out there).

For those of you who don't play many games, plants vs zombies is a turn-based strategy game where you need to stop zombies from taking over your house. PvNP is similar in the fact that you have Plants and Non-Plants (instead of zombies), and you must use the plants to protect your field from non-plants. The basic aim of the game to prevent any non-plants from reaching the left edge of your screen. The way that the non-plants are stopped is by buying plants from a virtual shop (automatically bought upon use) and blocking the non-plants (temporarily). Another extra feature is the ability to draw power-up cards to temporarily increase the strength of the plants surrounding the non-plants.

Chapter II

Project Outline

II.0.1 Game Rules

- Each turn, the player will be asked for keyboard input for actions of:
  - Placing a plant (at a `ROW` number and `COLUMN` number)
  - Drawing a powerup card (`C`) which temporarily increases damage by a random amount for all existing plants
  - Quitting the game (`Q`)
  - Doing Nothing (inputting nothing and pressing `ENTER`)
- Non-plants enter from the right and move forward one position each turn
- Non-plants are shown with a number, indicating how many occupy that position, or a # if more than 10 occupy that position
- Plants can be placed in any empty position on the field (placing a plant results in a loss of in-game cash)
- Plants are shown as `P` characters
- Empty spaces are shown with . characters
- Plants attack the first NonPlant anywhere in front of them (destroying a nonplant corresponds to a gain of in-game cash)
- NonPlants only attack plants one position in front of them
- If all waves are eradicated, the plants win
- If a non-plant reaches the far left of the field, the non-plants win

II.0.2 Game Input file

When the PvNP.py file is run (the file with your main python function), it accepts an argument of an extra file that holds data of the games properties.
Game files include:

- The amount of cash the player starts with
- The height and width of the playing field
- Any number of waves, each containing:
  - The turn number during which the wave of nonplants is released
  - The row in which the nonplants are to be released
  - The quantity of nonplants to be released
- Game files conform to the following format:
  Cash Height Width
  Turn Row Quantity
  Turn Row Quantity
  ...

We have provided example files that you can use when testing and to get an idea of what the format of the file will be.

II.0.3 Example game play

Here is an example of how your game should look:

```
$python3 PvNP.py ../examples/example_gamefile
Cash: $120
Waves: 3
   0  1  2  3  4  5  6
0  .  .  .  .  .  .  .

1  .  .  .  .  .  .  .

2  .  .  .  .  .  .  .

3  .  .  .  .  .  .  .

4  .  .  .  .  .  .  1


Action?
       [ROW COL] to place plant ($35)
       [C] to draw a powerup card ($5)
       [Q] to quit
       [ENTER] to do nothing?
```

There is a print-out of the field, how much cash the player has to buy plants from the virtual shop, and which wave is currently on the field. On the field, the numbers are the quantity of non-plants that are on the field, while the letter Ps are the plants that the player has the abiliy to place on the field.
The player is also prompted after each turn to input the row and col he wishes to place the plant, whether the player wants to draw a power-up card, quit the game or simply do nothing.

We have also provided a binary file that you can test out to see what the finished product should look like...if it's not provided to you, you probably were supposed to give your mentor some chocolate.

Chapter III

General instructions

Read and follow the exercises provided. Ask your mentor for help when needed.

Good luck!

Chapter IV

Exercise 00 : Create the `Organism` Class

	Create the Organism Class
Topics to study :	
Files to turn in : `organism.py`	
Notes : `n/a`	

1. Create a class called `Organism` that has instance variables of `hp` (hit points) and `dmg` (damage). When you set instance variables for a class, make sure to do something like this:

```
def __init__(self):
    self.hp = 35
    self.dmg = 10
```

   Here we're using the `self` attribute of the class to assign the instance variables. Make sure the `hp` is defined to start at 35 and the dmg to 10

2. Create a `take_damage` method, prototyped as:

```
def take_damage(self, damage)
```

   that takes away `damage` from the hp.

Chapter V

Exercise 01 : Create the `Plant` class

	Create the Plant Class
Topics to study :	
Files to turn in : `plant.py`	
Notes : `n/a`	

1. Create a class called `Plant` that inherites from the `Organism` class.

If you don't remember from the first day, inheritance for a Child to its Parent class, requires a class prototype like:

```
class Child(Parent)
```

2. Make sure the `Plant` properly inherites all the attributes of the `Organism` by adding:

```
super().__init__()
```

to the `__init__` method.

Also an important note. Python won't recongnize your Parent class unless you import it into the child's file. For example, if you have a class called Human and want to import it from homosapien.py then your import statement should be:

```
from homosapien import Human
```

3. Create a Class variable `cost` and set it to 35, this is the amount of cash that you need to buy a plant.

Just to remind you, here's an example of creating a Class variable (it's not the same as setting an instance of the class)

```
class ClassName(ParentsIfAny):
    classVariable = someNumber
    def other_methods:
```

If you want to know the reason for using a Class variable over an instance variable, ask your mentor.

4. Initialize the plant's `powerup` instance variable to 0.

5. Create a method called `attack`, prototyped as:

```
def attack(self, nonplant)
```

that applies a damage amount of the plants `dmg + powerup` to the nonplant (hint, use: `take_damage`, the nonplant wil be a child of the `Organism` class (see next exercise)).

6. Create a method called `apply_powerup`, prototyped as:

```
def apply_powerup(self, card)
```

that adds the card's `power` instance variable (will be created in a later exercise) to the plant's `powerup` instance variable.

7. Create a method called `weaken_powerup`, prototyped as:

```
def weaken_powerup(self)
```

that sets the plant's `powerup` instance variable to half of what it was.

Chapter VI

Exercise 02 : Create the `Non_Plant` class

	Create the Non_Plant Class
Topics to study :	
Files to turn in : `non_plant.py`	
Notes : `n/a`	

1. Create a class called `Non_Plant` that inherites from the `Organism` Class.
2. Create a class variable called `worth` and set it to 20.
3. Make sure the `Non_Plant` properly inherites all the attributes of the `Organism`.
4. Initialize the non-plant's `hp` instance variable to 80.
5. Initialize the non-plant's `dmg` instance variable to 5.
6. Create an `attack` method, prototyped as:

```
def attack(self, plant)
```

This method will apply the non-plant's `dmg` to the plant

Chapter VII

Exercise 03 : Create the `Card` class

	Create the Card Class
Topics to study :	
Files to turn in : `card.py`	
Notes : `n/a`	

1. Create a class called `Card`.
2. Create a class variable called `cost` and set it to 5.
3. Add an instance variable of `power` and set it to the value that is passed into the ___init___ function.

`This means your init function should be prototyped like so:`

```
def __init__(self, power)
```

Chapter VIII

Exercise 04 : Create the `Wave` class

	Create the Wave Class
Topics to study :	
Files to turn in : `wave.py`	
Notes : `n/a`	

1. Create a class called `Wave`.
2. Set the instance variables of the `Wave` class to be `wave_num`, `row`, and `num`, which should all be passed into the ___init___ function of the class.

`This means your init function should be prototyped like so:`

```
def __init__(self, wave_num, row, num)
```

Chapter IX

Exercise 05 : Create the `Game` class

	Create the Game Class
Topics to study :	
Files to turn in : `game.py`	
Notes : `n/a`	

This is the longest class we will make. It contains all the actual game logic, putting together all the classes and data structures learned this week.

Since we'll be using all the files you have made up till now, make sure you import all the required files and libraries that are needed to allow all the game logic to run easily. For ease of use, we'll include all the necessary import statements here, so you can easily paste it into your game.py file.

```
from linked_list import LinkedList
from queue import Queue
from stack import Stack
from wave import Wave
from non_plant import Non_Plant
from plant import Plant
from card import Card
import random
```

Make sure you have a linked_list.py, queue.py, and stack.py from earlier in the week!

1. Create a class called `Game`.

2. The init function for the `Game` class will need to be able to read input from a initialization file. Thus, to make it smoother for your implementation of the Game logic, we have provided the code that will read in input from a file (just copy and paste it into your `Game` class):

```
def __init__(self, file):
        with open(file, 'r') as f:
                self.cash, self.height, self.width = [int(x) for x in f.readline().split(' ')]
                self.waves = LinkedList()
                self.waves_num = 0
                for line in iter(f.readline, ''):
                        self.waves.add(Wave(*[int(x) for x in line.split(' ')]))
                        self.waves_num += 1
        # WRITE YOUR ADDITIONAL INIT CODE HERE
```

It's important to read over this function, and try to understand it. For example, the game's waves instance variable is initialized to a LinkedList.

If you need assistance understanding ask your mentor.

Where it says `#WRITE YOUR ADDITIONAL INIT CODE HERE` make sure to add the following functions to your init function for the `Game` class:

(a) Create an instance variable called `board` that will store and initialize an empty `Queue` of size of the `width` and `height` of the read-in file. These are stored in the respective instance variables of the `Game` class automatically handled in the code above.

(b) Create an instance variable to store:

i. Whether the game is over, and initialize it to `False`.

ii. The turn number, and initialize it to 0.

iii. The number of nonplants, and initialize it to 0.

iv. A deck of powerup cards, and initalize it to a `Stack`.

(c) For the initialized stack of deck powerup cards, loop through a `range` of 100 and push `Cards` initialized to a random `int` including and between 0 and 5 into the `Stack`.

3. Create a `draw` method in the `Game` class and copy, paste, and tweak the code to how you want your output to look. For now it will be better to simply copy and paste the code, and then later once everything is working, to actually tweak it.

```
    def draw(self):
            print("Cash: $", self.cash, "\nWaves: ", self.waves_num, sep = '')
            s = '  '.join([str(i) for i in range(self.width - 1)])
            print('  ', s)
            for row in range(self.height):
                    s = []
                    for col in range(self.width):
                            if self.is_plant(row, col):
                                    char = 'P'
                            elif self.is_nonplant(row, col):
                                    size = self.board[row][col].size()
                                    char = str(size) if size < 10 else "#"
                            else:
                                    char = '.'
                            s.append(char)
                    print(row, '  ', '  '.join(s), '\n', sep='')
            print()
```

Tweak at your own risk, this part will be graded by your peers.

Chapter X

Exercise 06 : Check the type of object in the board

	Create type checks for the Game class
Topics to study :	
Files to turn in : `game.py`	
Notes : `n/a`	

1. Create a method `is_nonplant`, prototyped as:

```
def is_nonplant(self, row, col)
```

that returns whether the element at the passed in `row` and `col` of the game's `board` is of type `Non_Plant`.

`Look up the built-in python method type(). Also, the peek() method or its equivalent in your Queue class will be helpful.`

2. Create a method `is_plant`, prototyped as:

```
def is_plant(self, row, col)
```

that returns whether the element at the passed in `row` and `col` of the game's `board` is of type `Plant`.

Chapter XI

Exercise 07 : Edit the placement of objects in the board

	Edit the placement of objects on the board
Topics to study :	
Files to turn in : `game.py`	
Notes : `n/a`	

1. Create a method `remove`, prototyped as:

```
def remove(self, row, col)
```

   that removes the entity at the corresponding `row` and `col` passed into the method from the board (hint, dequeue).

   If the entity is a `Non_Plant`, then add the `worth` to the game's `cash` instance variable (initialized in the init function of the `Game` class). Make sure to adjust the value of the game's instance variable for the number of nonplants.

2. Create a method `place_nonplant`, prototyped as:

```
def place_nonplant(self, row)
```

   that adds a nonplant to the game (again adjust the value of the game's instance variable for the number of nonplants). Make sure to:

   (a) Create a new `Non_Plant` object.

   (b) Then, add that new `Non_Plant` object to the row passed in at the last column of that row.

We are adding to the last column of the row, because the nonplants enter to attack from the last column, travelling to the left of the board.

3. Create a method `place_plant`, prototyped as:

```
def place_plant(self, row, col)
```

that adds a new `Plant` object to the passed in `row` and `col`.
Keep note that:

(a) Plants cannot be stacked on top of each other.

(b) Plants cannot be placed in the same location as a nonplant.

(c) Plants cannot be placed in the rightmost column (that is where nonplants enter).

Be careful with the logic here and don't forget, plant's cost money when they're placed!

4. Create a method `place_wave`, prototyped as:

```
def place_wave(self)
```

For each wave in the list of waves, check if it is equal to the current turn's number. If so, then:

(a) Release the appropriate number of non plants in the proper `Wave` class's instance variable `row`.

Remember the Wave is a linked list, so make sure to access the data properly.

(b) Remove that wave from the beginning of the list.

(c) Decrement the `wave_num` counter, which stores the number of waves that are left.

Chapter XII

Exercise 08 : Execute turns

	Execute turns
Topics to study :	
Files to turn in : `game.py`	
Notes : `n/a`	

1. Create a method `plant_turn`, prototyped as:

```
def plant_turn(self)
```

that, goes through the playing field, and for each plant in the field:

(a) Searches for the first nonplant infront of the plant and attacks it.

`You can print to the console a message that says the plant has attacked a nonplant, or something along these lines`

(b) If the nonplant's `hp` is less than or equal to 0, removes the nonplant from the playing field.

2. Create a method `nonplant_turn`, prototyped as:

```
def nonplant_turn(self)
```

that, goes through the playing field, and for each nonplant in the field:

(a) Checks if the nonplant has reached the leftmost column, this means the game is over, and you can display a game over message and exit the function (hint, `return`).

(b) Checks if the column to the left of the nonplant:

i. Has a plant, then the nonplant attacks the plant.

Because each spot in the board is a queue, you need to loop through the size of the queue for each nonplant attacking the plant that many times. Then check if the hp of the plant is less than or equal to zero, in that case, remove the plant from the board.

ii. After the attack this should be checked: If no plant is now in the spot to the left of the nonplant, then combine the queue on the left with the queue on the right, and set the spot where the nonplant was to an empty queue.

3. Create a method `run_turn`, prototyped as:

```
def run_turn(self)
```

(a) Increment the turn counter by 1

(b) Weaken all the plants' powerups (hint, loop throught all the plants in the `board`)

(c) Run the plants' turn

(d) Run the nonplants' turn

(e) Place the proper wave

(f) Check if the number of nonplants and waves are both zero.
If so, the game is over, draw the game and let the user know that they have won!

Chapter XIII

Exercise 09 : Finishing up the Game class

	Finishing up the Game class
Topics to study :	
Files to turn in : `game.py`	
Notes : `n/a`	

1. Create a method `draw_card`, prototyped as:

```
def draw_card(self)
```

that removes the top card from the stack. Then, it applies the card's power up to all the plants on the board.

`Cards cost money, so make sure to handle it accordingly`

2. Your last method for this huge class, is a `get_input` method that will get the input of the user, and execute accordingly. To make this last bit easier, we've got the whole method here, so just copy and paste it into your `Game` class.

```
def get_input(self):
                while True:
                        ui = input("Action?\n\t[ROW COL] to place plant ($" +
                                        str(Plant.cost) +
                                        ")\n\t[C] to draw a powerup card ($" +
                                        str(Card.cost) +
                                        ")\n\t[Q] to quit\n\t[ENTER] to do nothing?\n")
                        if (len(ui) > 0):
                                if (len(ui) == 1):
                                        if (ui.lower() == 'c'):
                                                self.draw_card()
                                                break
                                        elif (ui.lower() == 'q'):
                                                self.over = True
                                                break
                                        else:
                                                print("Invalid Input \"" + ui + "\"")
                                else:
                                        try:
                                                row, col = [int(x) for x in ui.split(' ')]
                                                self.place_plant(row, col)
                                                break
                                        except:
                                                print("Invalid Input \"" + ui + "\"")
                        else:
                                break
```

Of course tweak it how you want :)

Chapter XIV

Exercise 10 : Complete the Main method

	Complete the Main method
Topics to study :	
Files to turn in : `PvNP.py`	
Notes : `n/a`	

For the main method, we have provided you with some simple code for running the main program. This is what the user will be executing to start the game.

```
import sys
from game import Game

if __name__ == "__main__":
        if len(sys.argv) == 2:
                game = Game(sys.argv[1]);
                game.place_wave()
                game.draw()
                game.get_input()
                #ADD ADDITIONAL CODE HERE
                print("Game Over");
```

Where the `#ADD ADDITIONAL CODE HERE` is where you need to add:

1. While the game is not over, call the `run_turn` method of the `Game` class.
2. If after `run_turn` the game is not over, call the game's `draw` method and then the game's `get_input` method.

Chapter XV

Bonus part

Anything extra that your corrector deems a game-changer will be counted as a bonus. Game changers include but are not limited to:

1. Colored print statements.
2. Output of what happened between each turn.
3. Command line animations.

Chapter XVI

Turn-in and peer-evaluation

Turn your work in using your `GiT` repository, as usual. Only work present on your repository will be graded in defense.