Tutorial Seven: Classes

Imad Pasha
Chris Agostino

March 16, 2015

1 Introduction

This is probably the most fun and confined tutorial all semester. Today we are going to be working a bit with classes in order to program a simple video game, somewhat like pokemon. For any of you out there who may for some reason hate pokemon, you can easily adapt it to work for something like avatar the last airbender. If you don't like either, there's no hope for you.

Classes make programming a simple game like this (where you have pokemon/benders of different types, with hp and attacks, defenses, etc) more organized and easy to work with. Your task today is to create a game in which multiple players can attack each other in turn, lowering their opponent's hp until one player is left alive. How you do this we will leave largely up to you, though we will have a few basic stipulations and lots of advice and tips for how to accomplish different tasks.

2 The Setup: Classes

Note: From here on we will make references to pokemon for brevity, feel free to apply them to any fighter style game.

Your first step should probably be setting up a class called Pokemon, which pulls from the object class. Don't forgot to start inside the class by defining an init function, which takes self as the first argument. After that, you'll want it to take anything relevant to the pokemon's stats, such as name, type, initial hp, an attack stat, a defense stat, etc). Now you have the ability to initialize a pokemon by typing something like player1 = Pokemon('pikachu', 'electric', 100, 80, 75).

Inside your Pokemon class, you should also create a method for taking damage. It will probably take as inputs self, initial hp, your type, and the opponent type at minimum. (Feel free to make it more complex in the factors that determine an hp hit). Within this function you can return a new, lowered hp based on things like type match up (if your opponent is water and you are water you take less damage).

Hint: The function np.random.randint(-2000,2000)/10000. will spit out a small number between -.19 and .19, which you may want to use to modulate the hp hit a character takes to introduce a bit of unpredictability or randomness into the game.

3 Initializing: Picking Pokemon

At the start of the game, you will want your players to pick a pokemon to use, and maybe enter a name for it. You can do this by using the "raw input" function. It works like this:

name = raw_input('Enter a name for your Pokemon: ')

When python reaches that line of code, it will print what is in parenthesis to the screen and prompt for text entry; whatever is entered will be stored as a string in the variable name. Now when you initialize player1 = Pokemon(name, ...) you can put the name variable there and it becomes the name of the pokemon.

You might also have the player choose a type (like water or flying or electric) in the same way, or define it for them. You can also only have several preset options for them to choose from. It's up to you.

4 The Turn Function

Next you will probably want to create a function that includes all the code that defines a "turn". As inputs, it can just take attacker and receiver (or something of the like). Within it, you can do anything that happens during a turn, like prompting the user with raw input to choose an attack, calling your damage taking function to lower the receiver's hp, printing everyone's stats to the screen, etc. As a reminder for calling classes, here is an example of calling a damage method within the pokemon class, if we have initialized a pokemon as the variable player1:

player1.hp = player1.take_damage(player1.hp, player1.type, player2.type)

Basically, we set the hp of player1 to be the output of the method take_damage, which is a method that can be applied to any pokemon using dot notation. For inputs we just have the starting hp and the types of the two pokemon, you might be more complex). If you are doing this in your function, all the player1 would be replaced with attacker, all the player2 with receiver. This is nice because it allows you to code a whole set of turns by typing turn(player1, player2), followed by turn(player2, player1). (Assuming your function for a turn was called "turn"). Which in turn means that you could have just those two lines in a while loop that keeps alternating turns until some condition is broken (like both player's hp being over 0).

Hint: For printing something to the screen like "John's hp dropped to 30", where John and 30 are actually variables in your code, you can do the following:

```
print '%s \'s hp dropped to %s' %(player1.name, player1.hp)
```

Basically, typing %s in a print string, and following it with the parenthesis syntax allows python to fill in the %s sequentially with what is entered in the parenthesis after. the backslash 's is just to allow you to John's, since the ' would normally end the string.

5 Some Tips For Fun!

That's pretty much everything you need to cover for the basics. The following are some things you can do to make the game more complex:

- Have the damage function use "attack" and "defense" stats in addition to type matchup to determine how much hp should be taken.
- Have attacks of different strengths, and have the attacker expend some hp using an attack; more hp for a stronger attack, etc.
- Have moves whose sole purpose is to bolster your own stats or affect opponents stats (without affecting hp that turn).
- Have random "health packs" appear every once and a while at the start of a turn, allowing the player to take it and increase their hp, but lose the ability to attack that turn.
- Introduce a third player, and have each player decide at the start of their turn who they want to attack.

Basically, have fun with it! This type of experimentation and implementation of features is what we should see in your final projects, so it's good practice. The basics of writing any full program is to start simple (just have hp and hit each other till someone dies), and then piece by piece adding in more and more complexity. Working with classes and functions ensures that it is possible and fairly easy to implement such features later on.

6 Homework

The homework for this week is to finish up your game, and make it pretty, with an intro description of how to play, stats printed to screen, etc. We encourage you to spend the week tinkering with it and making it more complex and fun to play! When you're done, you'll be turning in your code. Be warned, we will be playing your games through to make sure they work!