**CS5001**

**Spring 2019**

**HW7 (evvvverything!)**

**Assigned:** March 28, 2019

**Deadline:** April 11, 2019 at 9:00am

To submit your solution, compress all files together into one .zip file. Login to handins.ccs.neu.edu with your CCIS account, click on the appropriate assignment, and upload that single zip file. You may submit multiple times right up until the deadline; we will grade only the most recent submission.

If you apply your late token, your new deadline is **April 15 at 9:00am**. Email laneys@northeastern.edu to notify us you’ll be cashing it in.

**The Assignment -- Space Invaders**

Have you ever played Space Invaders? It’s a classic Atari game, and you can still find it at many arcades. If you’re not familiar with this awesome ‘80s throwback, play it online [here](http://www.pacxon4u.com/space-invaders/). You can play it all day and it totally counts as work.

In the game, aliens are invading (from space) and the player has to shoot them down. The aliens are attacking the player, too, and the goal of the game is to knock off all the aliens before they crash into you.

Here is the general idea of what you’ll create: <https://youtu.be/dsKCwxL1x9Y>.

The only specific file you need to submit is design.txt (see below). Otherwise, submit whatever files you need for the game to work and we’ll use whatever you give us.

**Testing Requirements**

Remember to test as you go -- write a function, make sure it works, and then plug it into a larger program. For this homework, you’ll test your code using Python’s TestCase class, defined in the unittest module.

Some of your functions will involve graphics, which makes them near-impossible to unit-test. The requirement for this homework is to separate graphics from functionality. If a function does some drawing/Turtle, it does nothing else.

You must test any classes you define. Your test code should create at least one object from each class, and then verify that the attributes of the object are initialized correctly. If your class has a method that modifies the object’s attributes, test that too. H[ere’s an example of test code that works on objects](http://www.ccs.neu.edu/home/laney/cs5001/spring19/code/coffee_test.py).

You must test any file reading/writing in your code. If you read from a file, test that you read in the contents of the file correctly. If you write to a file, test that the function creates the expected file contents. We have some examples in [this code we went through in class](http://www.ccs.neu.edu/home/laney/cs5001/spring19/code/website_func_tests.py).

Your code must be thoroughly tested, and your tests must be well-documented, for full credit in **any** level of the grading rubric.

**Documentation Requirements**

* Filename: design.txt

You’ll submit a design file describing which level of game you developed. You’ll include:

* Which level you accomplish: C, B, A, or A+ (see below for descriptions of each)
* A bullet-point summary of your design, including:
  + What classes you defined, including the attributes and methods for those classes.
  + What data structures you used (lists, dictionaries, stacks, etc.).
  + Any non-class functions you defined.
* A one-paragraph summary of your approach to testing. You’ll write unit tests as well, but there’s some that can’t be unit-tested, what with the graphics and all. So, how did you determine that the game worked as expected? (For example, in our code we wanted to test ending the game -- so we created only one alien and shot it down -- game over; then made only one alien, increased the speed, and waited for it to smash us -- the other game over. When both worked, we considered that a successful test)

**Grading Rubric**

The grading rubric will be specific to this homework. Everyone **must turn in something that works, including test code,** even if it’s not the full functionality of the game; if you submit code that doesn’t run at all, or throws error messages as soon as we press a button, you will receive a zero for this homework.

We’ve provided ranges of scores below. Your solution must do at least what is specified in each range, but your score can vary depending on how understandable, efficient, and well-documented your code is.

**For a C**

To earn this grade, you’ll create a simple version of the game, including the following specs:

* There are 10 aliens on the screen.
* There is one player, at the bottom center.
* The user presses left/right arrow keys to move the player..

When your program kicks off, the 10 aliens are all placed in random locations, but all within the top third of the screen. The aliens all move right at the same speed (in the sample code video, we move each alien 10 pixels to the right each time they move. When an alien hits the right wall, it reappears on the left wall and moves down (again, for an example, we moved our aliens down 70 pixels).

When an alien reaches the bottom of the screen, it disappears. When there are no more aliens left, the game is over.

In our example code, we use a turtle shape for the player and triangles for the aliens, but you can use any shapes/colors you want.

**For a B**

To earn this grade, do everything specified for a C grade, and add the following functionality:

* The user presses the spacebar to fire the laser. If the laser collides with an alien, the alien disappears.
* The user earns 10 points for each alien they hit with a laser.
* When an alien floats down to the bottom of the screen, it might collide with the player, in which case the player dies.
* Each alien moves at a randomly-chosen speed.

In this version, the user can hit aliens to earn a score. When the user hits the spacebar, a laser is fired directly up from wherever they are. Every time the user hits an alien, they earn 10 points and their updated score is drawn on the screen.

There are now two ways for the game to end: (1) there are no more aliens, because they floated to the bottom of the screen and disappeared or got hit by lasers, or (2) an alien crashes into the player.

**For an A**

To earn this grade, do everything specified for a B grade, and add the following functionality:

* Track the high score of everyone who has ever played your game in a file. Draw the high score on the screen throughout the game, and if the user surpasses that score, prompt them for their name at the end and record it.
* Allow for multiple levels. Now, when all the aliens are gone (either shot by lasers or they float down and off the screen), clear the screen, replace the aliens and start again. The user’s score will accumulate; this is just moving on to the next level.

There is no need to keep any other scores, just the one high score. The user could enter anything for their name (initials, a full name, with or without spaces, etc.), but their score is whatever it was from lasering aliens.

**For an A+**

Extra points are on the line for this one. You could get an A+ or even above a 100%, depending on what you add to the game.

You can be creative with this part -- add any kind of feature(s) you like. The more elaborate you get, assuming it all still works, the higher your score. Some ideas:

* Add sound to the game (like a “pew! pew!” when the laser is fired).
* Instead of using built-in turtle shapes, find and use images instead (like [this](https://cdn1.itpro.co.uk/sites/itpro/files/images/dir_170/it_photo_85339.jpg) for aliens).
* With each new level, make the game more challenging -- increase the speed of the aliens.
* Add defense barriers. The original Space Invaders has this, barriers the player can hide behind. They get destroyed by the aliens over time, but they help to protect the player.