ENGR 102 - Project Part 1 (Lab Assignment #12)

TEXAS A&M UNIVERSITY

Fall 2020

General course learning outcomes:

- demonstrate the use of programming techniques in the construction of computer programs, including the use of: large data structures such as lists; control structures such as conditionals and loops; user-defined functions; data from and processed to an external file.
- decomposing a complicated task into manageable pieces.
- apply programming techniques to solve problems in engineering.
- complete a team programming assignment that ties together concepts learned in the class.

Team ⋆ Assignment 1

(Reminder: this assignment is worth 40 points)

For this assignment, your team will use bottom-up and top-down design to begin programming another, more complete, version of "Gotta Catch One of 'Em". The next pages lists several requirements the game must satisfy, but your team has significant freedom in the specifics of how to do so. For full credit your program should be well-designed, functional, and nice-looking.

A. Bottom-Up: As a team, brainstorm what functionality you are likely to need in this program.

- Your goal for Part A is not to generate a complete program design, but to identify a list of functions to create that will likely be helpful in your program (*e.g., pokeball_throw*)
- Consider functions that will call other functions only after you have designed the lower functions (*e.g., remember the trajectory_y( ) function from last week*).
- Skip parts you are uncertain about, and come back as necessary.
- For each basic task where all necessary pieces are already known, write a short function description, required input arguments (if any) and function return values (if any).

B. Top-Down:

- In a new section of your document, create and outline your team *top-down* design in a flow-chart as shown earlier this semester. Make use of the functions created in part A.
- Decide what data you will need to keep that will be used in more than one "node" of the design. Determine what variables you will use in the main code; write a list of these, along with a brief description of what that variable will store. *Note: you do not need to decide on variables used locally, e.g. a loop iterator.*
- Create the sequence of steps required to go from your plan into a working program.

C. Prepare your Python Project:

- Transfer your hybrid top-down / bottom-up plan into function stubs and comments in your program:
  i. The first part should be a list of function stubs, using the *pass* command, for the functions you will write, with the first ones being from the bottom of your hierarchy.

     For *every* function, provide a name, write a docstring, and indicate what input parameters and return values are required so the functions will interact correctly. Much of this should already be completed from Part A; copy the information to code.

  ii. The second part should be comments outlining the main code sequence of general steps that will be required for the program, including where calls to the above-listed functions are required.

(continued, next page)

Program Requirements:

Use functions frequently to simplify your main code.

- Consider: menu building, catching Pokemon, leveling up Pokemon, calculating CP, displaying current Pokemon information, etc.
- Minimize the number of functions needed by reusing functions where possible (i.e., if you find you are rewriting the same code in the program, it may be cleaner to utilize a function).

Menus: The game must provide several menus and submenus.

- A main menu should be created to allow several main functions of the game.
- Menus should be informative and intuitive
    - Game and Pokemon information must be used when populating the menu titles, options, and relevant information (e.g., displaying the current Pokemon information, and the number of candies available when viewing the level-up menu.)
- The Pokemon selection menu must show multiple options per row (see next page for an example).

Two Player: The game must provide a method to switch between player accounts.

- Each player must be able to access their separate list of Pokemons.
- Your program should create a new account for any new player, with no set limit of players possible.
- Only two players will be 'active' at a time. There must be some method of taking turns between the two players.

Selecting an Active Pokemon: A player must be able to select from all previously caught Pokemon in their game profile.

- A Pokemon is assigned to each player when they start the game the first time; either randomly, or by starting with a choice of low-level Pokemon.
- An external file should be used to load and store a list of Pokemon and their current CP and stored candies.
- Newly 'caught' Pokemon should be added to the file.
- The external file should update if a Pokemon's stats change.
- You may allow a player to have a single active Pokemon, or you may allow multiple Pokemon if you would like to add options to your version.

Catching: A player must 'catch' a new Pokemon through winning a type of mini-game.

- The specific game can be determined by your team, but make it interesting, and of appropriate time consumption for the player. (e.g., winning a round of hang-man, winning a game of Angry Birds as you designed recently within a certain number of guesses, etc….)
- If a player catches a new Pokemon, the player will be randomly awarded 1, 3, 5 or 10 candies. For our game, candy is generic and the same candy works for all Pokemon. You may add a weighting to the randomness if you'd like to favor the lower values, and make higher values more rare.
- A CSV file is available with many Pokemons specified that a player may catch. You may modify the file as desired, but it must contain at least the same amount of information, and remain a CSV file type.

Leveling: A player should be able to 'level up' Pokemon according to the following rules.

- Level 40 is the maximum level.
- One (1) candy is required to level the Pokemon by 1 level from levels 1–30, two (2) candies are required to level-up to levels 31–40.
- Candy must be awarded through catching Pokemon, rather than added by the user as we did in Week 6.
    - CP is calculated as CP * 0.0094 / (0.095 * sqrt(Current Level)) for levels 1–30
    - CP is calculated as CP * 0.0045 / (0.095 * sqrt(Current Level)) for levels 31–40

(continued, next page)

Battling: The two active players should be able to 'battle' their Pokemon.

- The outcome should be partially determined by a combination of randomness and the Pokemons comparative levels and CP.
- The specific battle format can be determined by your team. It should involve multiple 'moves' of some type, with user involvement.
- You may award candy, or level-ups to winning Pokemon.

Example menu structures:

These are not required to be used verbatim.

```
---------------------------          MAIN MENU         ---------------------------
1. View current Pokemon
2. Catch a new Pokemon
3.
```

```
---------------------------       Current Pokemon      ---------------------------
Pokemon Name

Current CP:    _______
Current Level: _______
Candies:       _______

1 - Use Candy to Level-Up
2 - Exit to Main Menu
```

```
---------------------------   Pokemon Selection Menu   ---------------------------
Pokemon Name

Current CP:    _______
Current Level: _______
Candies:       _______


-------------------------------------------------------------------------------
 1. Pokemon Name #1     2. Pokemon Name #2     3. Pokemon Name #3     4. Pokemon Name #4
    CP XXX                 CP XXX                 CP XXX                 CP XXX

 5. Pokemon Name #1     6. Pokemon Name #2     7. Pokemon Name #3     8. Pokemon Name #4
    CP XXX                 CP XXX                 CP XXX                 CP XXX

 9. Pokemon Name #1    10. Pokemon Name #2    11. Pokemon Name #3    12. Pokemon Name #4
    CP XXX                 CP XXX                 CP XXX                 CP XXX
-------------------------------------------------------------------------------
Select a new Pokemon (1-12):
```