1336 Programming Examination 3: Exam3

General Statement

You must write this examination using the **Template for Main Program** and **Template for Libraries** (or **Basic main** and **Basic Library**) files. All functions must be written in a library file named **warrior.py**. File **Exam3.py** contains only the main function. Consider starting with these posted solution files from Assignment 3: **combat3x.py** and **warrior.py**. Refer to the **Style Requirements.pdf** document posted in Blackboard for more requirements. These instructions will not specifically state requirements that you are expected to follow from that document.

The Problem

We're building a role-playing game.

Understand the Problem

The Task

In this fourth-level simulation, a warrior is defined as a set of 6 characteristics: name, energy, strength, speed, location, and direction. These characteristics are kept in a warrior profile list and are read from a file at the beginning of the simulation. **I will run your program with my own test file(s)**.

You will define two warriors and move them around. After each warrior moves you will display the game state (all the current values of each warrior's characteristics). The warriors keep moving until at least one warrior runs out of energy and dies.

Specifics

Change all the placeholders in the program header to be your name, program name, what the program does, and the date. All functions must be written in the library file **warrior.py**. Fill in the function headers with all appropriate values. Make sure that the header description and values match the code at all times.

Setup

Declare two list variables: warrior1, and warrior2. These are the warrior profile lists.

Declare global constants to use as indexes in the lists. They will allow you to refer to list items with words rather than numbers: instead of warrior1[0] you will write warrior1[NAME], etc.

The warrior characteristics are stored in a file named **warriors.txt**. There are two records in the file, six lines each. In order, the lines are [PAY ATTENTION!] the warrior's name, energy, strength, speed, location, and direction. These values fill a warrior's profile list.

Use these characteristic names as your global constants, so you have NAME = 0, ENERGY = 1, STRENGTH = 2 and so on for all six items in a profile list. Declare constant WARRIOR_RECORD_SIZE = 6. Declare these constants in *both* **Exam3.py** and **warrior.py**.

1336 Programming Examination 3: Exam3

Play (main program)

The four general program steps are: (1) build the warriors, (2) display the starting state, (3) while both warriors are alive (a) move and update each warrior, (b) check whether either warrior died, (c) display the current game state. After the loop finishes, (4) display who died.

Build the warriors

Call **get_warrior_profiles**() passing **warriors.txt** as a parameter. Assign the return value to a list named buffer.

Call **build_warrior_profile**() to create each warrior list. Pass buffer and the warrior's number as parameters; assign the return value to the warrior list (first warrior1, then warrior2).

Display the game state

Set time to 0.
Display the time, then call **display_warrior_status**() with each warrior list as parameter.

Do these things while both warriors are alive:

Increase time by 1.

Move and update the warriors

Call **move_warrior**() with warrior1 and warrior2 as parameters; assign the return value to warrior1. Essentially, you have updated warrior1 using this function call. Then call **move_warrior**() again with parameters reversed, assigning the return value to warrior2.

Display the new game state

Print the game state as described above.

Test the warrior's conditions

If either warrior's energy is 0 or less, set a flag that a warrior has died (this might happen to both).

Final report

Print the name of each warrior who is dead. If neither warrior is dead, print "ERROR" because you shouldn't get to this point in the program unless one of them died.

1336 Programming Examination 3: Exam3

The Functions

Be sure that each function header is complete: function name, description, the list of functions called, the list of parameters and their types, and the return value name and type.

Function get_warrior_profiles()

Function **get_warrior_profiles**() reads the data file and put the contents in a list.

The function has 1 parameter: a string that is the name of a file.

Open the file for reading and read the entire file contents into a list named buffer: this list will include all the values for all the warriors in the game.

Use an exception try for the file I/O – if there is an exception set buffer to a single item list containing "Error".

Return the buffer list.

Function build_warrior_profile()

Function **build_warrior_profile**() takes one warrior's description (6 items) from the buffer file and builds a new list representing that warrior.

The function has 2 parameters: a list named buffer and a warrior number (integer). Build a new warrior list by appending the six items in that warrior's record. Use the warrior number to determine where to start in the buffer, and WARRIOR_RECORD_SIZE to count through the items to append.

Convert the items for ENERGY, STRENGTH, SPEED, and LOCATION to integers.

Return the new warrior list.

Function display_warrior_status()

Function **display_warrior_status**() displays all the warrior's attributes.

The function has 1 parameter: a warrior profile list. Output is **one line** in this format:

<name>: (energy <val>, strength <val>, speed <val>, direction <val>) at location <val>.

where each <val> is the item in the warrior list corresponding to the printed name ([NAME], [ENERGY], etc.)

The function does not return a value.

1336 Programming Examination 3: Exam3

Function move_warrior()

Function **move_warrior**() relocates a warrior and adjusts its energy and strength.

The function has 2 parameters: moving_warrior and other_warrior; each is a warrior profile list. Only the moving warrior moves, and only its profile values are changed.

Move the warrior

The moving warrior's speed is the number of steps it will take. If the warrior's direction is left, each step subtracts 1 from its location; if the direction is right, each step adds 1.

With each step, if the new location will be the same as the location of the other warrior, reverse the moving warrior's direction and take the step in the new direction instead.

Continue until all steps have been taken.

Update the moving warriors' conditions

Subtract the square of the warrior's speed from its energy.
Subtract twice the warrior's speed from its strength.

Return the revised moving_warrior list.

Special Notes:

You earn up to 60% by correct operation and up to 40% with good style, readability, and documentation.

Submit (in **Blackboard**) the files **Exam3.py** and **warrior.py**.
Name the file(s) ***exactly*** *as given*. **Do not add anything to the program name**. **Not** your name, **not** your ID number, **not** words like "attempt 3" or "revised", or **any** additional characters as part of the file name.
The **Python** program is scored with a maximum value of **200 points**.