Computer Science I for majors by James Tam   Return to the course web page

CPSC 231: Assignment 5

New concepts to be applied for the assignment

1. Writing an Object-Oriented program
2. Using recursion to get a program to repeat

(Note: new design principles, e.g., Object-Orientation, as well design/style requirements from previous assignments such as good naming conventions must be applied in your work).

Overview

You are to write a dating simulator for fictional life forms called the "Tims" (Tam's simulated life forms) [1]. There are two types of Tims in this simulation: a 'pursuer' and a 'target'. The target is the object of the pursuer's desire. In the full version of the program, the goal of the pursuer is to have a successful date with the target. The target's behavior is purely random. The date consists of a number of social interactions. If by the end of the date the proportion of successful interactions is 50% or greater then the date is considered successful otherwise it's deemed a failure. Two types of reports will show statistics on various aspects of the date.

Detailed description

The program will consist of two classes: the `Pursuer` class and the `Target` class. Each class should be defined in its own module file.

`Class Pursuer`

Minimum attributes (you probably will define more)

Tally of the number of X-interactions generated by the pursuer

Tally of the number of Y-interactions generated by the pursuer

Probability of an X-interaction occurring (0 - 100%)

Probability of a Y-interaction occurring (0 - 100%)

Tally of the number of successful interactions with the `Target` (needed so that Pursuer can 'match' its behavior with that of the Target, even if you don't implement the intelligent matching behavior the information should be tracked here and not elsewhere).

Minimum number of methods (you will probably define more)

An "`init()`" constructor

A method that will display a report at the end of each interaction between the pursuer and target

A method that will display a report at the end of the simulation (when all interactions between the Pursuer and Target have finished)

One or more methods that will determine the type of behavior that will occur for a particular interaction with the target (among other things the random numbers must be generated).

`Class Target`

Minimum attributes (you probably will define more)

Tally of the number of X-interactions generated by the target

Tally of the number of Y-interactions generated by the target

Probability of an X-interaction occurring (0 - 100%)

Probability of a Y-interaction occurring (0 - 100%)

Minimum number of methods (you will probably define more)

An "`init()`" constructor

One or more methods that will determine the type of behavior that will occur during an interaction (among other things the random numbers must be generated).

In addition you will define a '`Manager`' module that contains the starting execution point for the your program (the `start()` or the `main()` function). The capabilities of this module can be written purely in a procedural manner (functions instead of classes and methods). At a minimum the functions in this module should be able to: 1) prompt the user for the number of interactions to occur when the program runs, 2) prompt the user for the probability of x-type interactions for the target 3) instantiate instances of the Pursuer and Target class inside of one of the Manager's functions. Finally the 'main loop' (that executes a number of times equal to the specified number of interactions) will be inside a function of the Manager. The appropriate methods of the pursuer and the target should be called within the body of this loop.

For an animated overview of the assignment see the following [presentation]

The total number of interactions for the date: it can be any integer >= 1. Your program should be able recover if the type of information entered by user is invalid (i.e., a non-numeric string) or if the range of information entered is invalid (zero or less). You can count floating point values as an invalid 'type' in this case. Whether the type or range of information is incorrect an appropriate error message should be displayed for each case and the program will prompt the user again for the number of interactions. **In order to get credit the re-prompting must be implemented using recursion and not a loop**.

Examples of determining the number of interactions that the simulation will run (program prompt and user input in blue, program response in red)

```
Enter the number of interactions (1 or greater): ab
Do not enter non-numeric values
Enter the number of interactions (1 or greater): 1.2
Do not enter non-numeric values
Enter the number of interactions (1 or greater): 1
...
```

(Input is OK, program continues)

The probability of the `Target` object exhibiting type 'X' interactions: It will be an integer value `0 <= (value) <= 100`. Similar to the total number of interactions your program should check the validity in the type and range of data entered and display appropriate error messages as necessary for each case. **Again recursion must be employed in order to get credit.**

Examples (program prompt and user input in blue, program response in red)

```
Enter the percentage # of 'X' interactions for target (whole numbers from 0 -
100): xyz
Do not enter non-numeric values
Enter the percentage # of 'X' interactions for target (whole numbers from 0 -
100): 1.5
Do not enter non-numeric values
Enter the percentage # of 'X' interactions for target (whole numbers from 0 -
100): 100
...
```

(Input is OK, program continues)

The program will then automatically determine the probability of type 'Y' interactions for the target object from the probability of X interactions. (The sum total of the two percentages must equal 100).

The probability of each type of interaction exhibited by the pursuer will depend upon the version of program that you implement. The simplistic approach will allow for an equal probability of type-X and type-Y behaviors (the "hope for the best" approach to relationships)[1].

The date will then run one interaction at a time. During each interaction the pursuer and the target will exhibit one of two behaviors: 'X' or 'Y'. If both the pursuer and target exhibit the same behavior during a particular interaction (i.e. both were 'X' or both were 'Y') then that particular interaction is deemed as successful (and is counted in a tally). Otherwise the interaction is deemed as unsuccessful. Next the *pursuer will display a report* describing the results of the interaction between the pursuer and target:

Format of report generated at the end of interaction round

```
<Behavior exhibited by the target><Behavior exhibited by the pursuer><Result of the interaction>
```

Example of report generated at the end of interaction round

```
Target behavior: x    Pursuer behavior: x     Matched behavior
```

Finally when the specified number of interactions have occurred, the final analysis of the date will be *displayed by a method of the Pursuer*. The report should include the total number of X-type interactions and Y-type interactions generated by the target and the pursuer. (Note: because the number of X's and Y's is already tracked by the target, the pursuer need not track this information again, it can simply get that information from the target). Next the program should show the number of and percentage proportion of successful interactions. (The percentage is equal to the number of successful interactions divided by the total number of interactions multiplied by 100). Finally the report should determine if the date was successful or if it failed (the former if the proportion of successful interactions was 50% or greater, the latter otherwise).

Examples end of date report
(10 rounds of interactions in this example):

```
ANALYSIS OF ALL THE INTERACTIONS (THE DATE)
Target: No. of X's=4     No. of Y's=6
Purser: No. of X's=8     No. of Y's=2
Number of successful matches: 4
Proportion of successful matches: 40.0%
Date was a dud :'(
```

(20 rounds of interactions in this example):

```
ANALYSIS OF ALL THE INTERACTIONS (THE DATE)
Target: No. of X's=20    No. of Y's=0
Purser: No. of X's=14    No. of Y's=6
Number of successful matches: 14
Proportion of successful matches: 70.0%
Date was successful <3
```

"Intelligent Pursuer"[1]
The 'intelligent' version of the pursuer will analyze the patterns of behaviors of the target and try to match that object's behavior (i.e., if many type-X behaviors were exhibited by the target then the pursuer would then have an increased probability of only exhibiting type-X behaviors).

Example of the pursuer adapting to the target (adapting after interaction #10)

```
Target behavior: x   Pursuer behavior: y   Mis-Matched behavior
Target behavior: x   Pursuer behavior: y   Mis-Matched behavior
Target behavior: x   Pursuer behavior: y   Mis-Matched behavior
Target behavior: x   Pursuer behavior: x   Matched behavior
Target behavior: x   Pursuer behavior: x   Matched behavior
Target behavior: x   Pursuer behavior: y   Mis-Matched behavior
Target behavior: x   Pursuer behavior: y   Mis-Matched behavior
Target behavior: x   Pursuer behavior: x   Matched behavior
Target behavior: x   Pursuer behavior: y   Mis-Matched behavior
Target behavior: x   Pursuer behavior: x   Matched behavior <== JT's annotation: the pursuer begins adapting to
the target's behavior
Target behavior: x   Pursuer behavior: x   Matched behavior
Target behavior: x   Pursuer behavior: x   Matched behavior
Target behavior: x   Pursuer behavior: x   Matched behavior
Target behavior: x   Pursuer behavior: x   Matched behavior
Target behavior: x   Pursuer behavior: x   Matched behavior
```

Exactly what constitutes 'intelligent' for this assignment? Because this isn't a formal class in Artificial Intelligence the definition is fairly broad. The pursuer demonstrates the ability to adapt to the pattern exhibited by the target and the marker can clearly see the pattern as the program executes (the display of statistics each round will be very useful). You can choose how many interactions will pass before the pursuer begins to adapt to the pattern (but it must be greater than zero interactions otherwise there is no 'pattern'). You can either come up with your own algorithm for the intelligent pursuer or you can freely research algorithms online. In the latter case make sure you clearly cite all your sources. (Failing to list a source may be regarded as academic misconduct).

If the need arises you can employ a fourth module called "`Globals.py`" which contains the definitions for global constants (these are global constants that are used in multiple modules - constants that are used in only one classes' methods can be defined in the same file module as the definition of the class). Global variables should not be employed in this assignment (unless you want to define/employ) another debugging tool.

D2L configuration:

- Multiple submissions are possible for each assignment: You can and should submit many times before the due date. F*or this assignment D2L will keep older versions of your submissions*.
- **Important**! Multiple files can be submitted for each assignment. I am allowing you to submit multiple files for each assignment so **don't archive/compress multiple files using a utility such as zip**. However, this means that **TAs will only mark the latest versions** of each file submitted via D2L. Even if the version of a document that you want marked has been uploaded into D2L if it isn't the latest version then you will only get marks for the latest version. (It's unfair to have the TAs check versions or to remark assignments because marking is enough work as-is).

Marking

- Assignments will be marked by your tutorial instructor (the "Teaching Assistant" or "TA"). When you have questions about marking this is the first person that you should be directing your questions towards. If you still have question after you have talked to your TA, then you can talk to your course (lecture) instructor.
- As well as being marked on whether "your program works" you will also be marked on non-functional requirements such as style and documentation. Consequently this assignment will include a separate [marking checklist]. Besides seeing your grade point in D2L you can also see the detailed feedback that your TA will enter for each student. You can access the grading sheet in D2L under `Assessments->Dropbox` and then clicking on the appropriate assignment link. If you still cannot find the grading sheet then here is a [help link]

Points to keep in mind:

1. **Due time:** All assignments are due at 5 **PM** on the due dates listed on the course web page. Late assignments or components of assignments will not be accepted for marking without approval for an extension beforehand. Alternate submission mechanisms (non exhaustive list of examples: email, uploads to cloud-based systems such as Google drive, time-stamps, TA memories) cannot be used as alternatives if you have forgotten to submit work or otherwise have not properly submitted into D2L. **Only files submitted into D2L by the due date is what will be marked**, everything else will be awarded no credit.
2. **Extensions** may be granted for reasonable cases by the course instructor with the receipt of the appropriate documentation (e.g., a doctor's note). Typical examples of reasonable cases for an extension include: illness or a death in the family. Example cases where extensions will not be granted include situations that are typical of student life: having multiple due dates, work commitments etc. Tutorial instructors (TA's) will not be able to provide extension on their own and must receive permission from the course instructor first.
3. **Method of submission:** You are to submit your assignment using D2L [help link]. Make sure that you [check the contents of your submitted files] (e.g., is the file okay or was it corrupted, is it the correct version etc.). It's your responsibility to do this! (Make sure that you submit your assignment with enough time before it comes due for you to do a check).
4. **Identifying information**: All assignments should include contact information (full name, student ID number and tutorial section) at the very top of your program in the class where the '`main()`' method resides (starting execution point). (Note other documentation is also required for most assignments).
5. **Collaboration**: Assignments must reflect individual work; group work is not allowed in this class nor can you copy the work of others. For more detailed information as to what constitutes academic misconduct (i.e., cheating) for this course please read the following [link].
6. **Execution**: programs must run on the computer science network running Python 3.x. If you write you code in the lab and work remotely using a remote login program such as Putty or SSH. If you choose to install Python on your own computer then it is your responsibility to ensure that your program will run properly here. It's not recommended that you use an IDE for writing your programs but if you use one then make sure that you submit your program in the form of text "`.py`" file or files.
7. **Use of pre-created Python libraries**: unless otherwise told you are to write the code yourself and not use any pre-created functions. For this assignment the usual acceptable functions include: `print()`, `input()` and the 'conversion' functions such as `int()`, `float()`, `str()`. Look at the particular assignment description for a list of other classes that you are allowed to use and still get credit in an assignment submission.

Complete sample output:

It can be found in UNIX under `/home/231/assignments/assignment5/output` [Shortcut]

[Attached file]

1 The 'theories' espoused in the assignment description are specific only to the Tims (actually just tongue in cheek). You are not guaranteed any success at dating or other social interactions if you apply any of the techniques specified in this document!