Mental Health and Wellness Screener
Project Plan

L. Vandecasteele (lv) N. Onofrei (no) A. Reichhold (ar) Q. Fetrow (qf) M. Gao (mg) -
2-14-2022 - v0.03

1. Programmer Documentation Revision History

Date	Author	Description
3-03-2022	Amy	Outlined the documentation based on the examples.
3-04-2022	Amy	Started filling in the outline.
3-05-2022	Michael	Added documentation of results.py files.
3-06-2022	Amy	Added introduction, program files, and conclusion.
3-06-2022	Nick	Added more documentation for modules
3-06-2022	Luke	UI.py documentation
3-06-2022	Quinn	Added SDQ documentation, updated Question module documentation

2. Introduction

This document covers all of the important information on the Mental Health & Wellness Screener by breaking down the source code and discussing the important classes and methods. This documentation is based on the EyeDraw v3.0 Programmer's Documentation [1] and World Tax Planner Programmer's Documentation [2] examples provided in the CIS 422 course, as taught by Professor Anthony Hornof at the University of Oregon.

All of the code is written in python and is written to be runnable on Python 3.7 through Python 3.10.

This documentation assumes an understanding in python and object-oriented programming.

3. Program Files

The UI.py is the main program entry point, it manages and displays the user interface. Upon start-up, the UI.py displays the start button, once the start button is clicked it displays the screener and the submit button. The SDQ.py communicates with the front-end module, the UI module, and the back-end modules, the Question and Result modules.
Each source file creates one class. The UI.py file implements displaying the screener, the SDQ.py implements generating the survey and submitting the survey by calling the Question and

Results modules, the question.py file creates an instance of a question, and the results.py file implements score calculations and storing the results.

3.1 question.py

The Question object stores every question's data, such as what the question message is, the category the question is in, the score array to get the questions value, and the questions value. The questions will be calculated by the results module, displayed by the Survey Interface module, and maintained by the SDQ module.

3.1(a) category → string

Used for storing which of the five categories the question belongs to.

3.1(b) message → string

Used for storing the message of the question (the question displayed to the user).

3.1(c) answer_array → [int]

The answer_array holds an array that represents how a question will be scored based on a value. For example, for a score_array of [2,1,0] and an answer_state of ID 0, the question value should be score_array[0] (= 2). This attribute is implemented because every question on the SDQ screener has options "Not True", "Somewhat True", and "Certainly True", but some questions can be scored differently.

3.1(d) answer_state -> tkinter.IntVal

The answer_state is the current state of the answer to the question. This value is bound to a set of three tkinter.RadioButtons in the UI module representing the options "Not True", "Somewhat True", and "Certainly True" as integers 0,1, and 2 respectively. The "get()" method is called on this object to get the integer representation in the self.set_value() function.

3.1(e) value → int

Used to store the "difficulty score" for the question answered. A lower value indicates less of a problem in the question's category. For example for the question "I am restless, I cannot stay still for long" in the "hyperactivity" category, a high value would mean the student has more of an issue with hyperactivity.

3.1(f) __init__(question_category, question_message, answer_array):

This method creates an instance of the Question object.

3.1(g) set_value():

This method is called from the SDQ module. It calls “self.answer_state.get()” which returns the current integer representation of the answer (“Not True”, “Somewhat True”, “Certainly True” as 0,1,2 respectively) and scores it based on the self.answer_array attribute. The score of the question can be represented by the value of self.answer_array[self.answer_state.get()], and question.value is set to this score.

3.1(h) set_answerstate(value->tkinter.IntVal):

This method assigns the attribute of self.answer_state to the variable “value”. “value” is bound to the three tkinter radio buttons assigned to each question representing the options “Not True”, “Somewhat True”, and “Certainly True” as integers 0,1, and 2 respectively.

3.1(i) value_to_plaintext() -> String:

This method is called by the results module to simplify the process of storing the student’s answers, it converts the answer’s integer value (0,1,2) to the plaintext version: (Not True, Somewhat True, Certainly True) and returns the plaintext string.

3.2 SDQ.py

The SDQ object is created and called upon by the UI module. It is created once the user presses the “begin screener” button. On startup, the user creates an instance of the Results module, and generates a series of question objects from a text file titled “questionnaire.txt”. These questions are accessed by the UI module.

3.2(a) questions -> [Question]

An array of Question objects that represent the survey. This array is to be accessed by the UI module to display the questions, and sent to the Results module to calculate and store the results.

3.2(b) result → Result

An instance of the Results module, used to calculate and store self.questions.

3.2(c) __init__():

Creates an instance of the Results module and calls the self method “generate_survey()”

3.2(d) generate_survey():

Parses a local file titled “questionnaire.txt” in which each line has the question text, the category of the question, and the method in which it is scored. The lines are in the following format: “[Question Text] ; [Category] ; [Question Order]”. This method uses the string function “.split(“;”)” on each line, which splits the line into an array whose elements are the sections of

the string between the delimiter ";". The method creates an array of questions with the parsed values.

3.2(e) submit():

Calls "set_value()" on all questions in the question array to score them, and calls "get_results(self.questions)" and "store_results(self.questions)" on the Results module to calculate the results and store them.

3.3 results.py

The Results object is created and called by the SDQ module. When get_results() is called, it will calculate all of the scores for the five categories and then sum the first four categories to calculate the total_difficulty_score. The store_results() function compares the scores to the ranges listed on the website which are: close to average, slightly raised, high, and very high and then saved to an external file.

3.3(a) emotional_score → int:

Used to store the emotional score, which is the sum of every answer from the questions in the emotional problems category.

3.3(b) conduct_score → int:

Used to store the conduct score, which is the sum of every answer from the questions in the conduct problems category.

3.3(c) hyper_score → int:

Used to store the hyperactivity score, which is the sum of every answer from the questions in the hyperactivity category.

3.3(d) peer_score → int:

Used to store the peer score, which is the sum of every answer from the questions in the peer problems category.

3.3(e) prosocial_score → int:

Used to store the prosocial score, which is the sum of every answer from the questions in the prosocial category.

3.3(f) total_difficulty_score → int:

Used to store the calculated total_difficulty_score, which is the sum of the emotional, conduct, hyper, and peer scores.

3.3(g) get_results([Question]):

get_results() loops through the list of questions passed in, checks which category the question belongs to, and adds the question's score to the score variable for the category that the question belongs to. After looping through the questions and calculating the scores for the five categories, the total_difficulty_score is calculated by summing the emotional, conduct, hyper, and peer scores.

3.3(h) create_file():

create_file() creates the output file, titled results.txt. If the file already exists, the file is overwritten to provide a clean file.

3.3(i) store_results([Question]):

store_results() creates the output file results.txt by calling the create_file() function, then calculates the users risk by comparing the user's scores for each category to predetermined ranges via the get_result() function. It then writes the user's risk and all questions and their respective user answers to the results.txt output file.

3.3(j) get_result(scoretype, ranges):

get_result() loops through an array (ranges) and checks if a specific number (scoretype) is the previously said array. It returns the loop index of the number.

3.4 UI.py

The user interface module is responsible for the GUI (graphical user interface) version of the screener. This module gives the user instructions on how to complete the screener, instructions about sharing the results, and the actual screener itself. This part of the system essentially turns the paper version of the SDQ into a slick graphical version. This module communicates with the SDQ module to get the questions and the information about the questions, and the results module to signal that the user has finished the screener and the system can now generate the results for the user. At the beginning of the file there are various global variables of large strings. These variables are all various instructions and tips that are in the system to assist the user. The only reason they have been placed as global variables is to clean up the code.

3.4(a) __init__():

The initializer method for the UI creates three different variables: each one for the various windows the user will see while taking the mental health screener.

1. First we have self.start. This is the variable that creates the window for the main menu screen via a tkinter object instance.
2. Then there is self.survey. This is a second variable for holding the window for the actual screener administration. This is a second tkinter object instance and is by far the most complex to account for all the different questions, RadioButtons for the answers, and allowing for scrolling on the screen.
3. Lastly we have the self.final variable which, again, is another tkinter window. This window informs the user when they have finished the screener, instructions for how to proceed, and provides options for them to take the screener again or close the application.

Lastly we also have a variable self.screener. This variable creates an instance of the SDQ class found in SDQ.py. In creating an instance of the SDQ class, this variable hold the questions for the screener which are in the form of Question class objects.

3.4(b) run():

This is simply the main driver function for the program. Its purpose is to do basic error checking to ensure that the questions for the screener were loaded in properly and that the application will work as intended for the rest of its lifecycle.

3.4(c) init_interface():

This method creates the first window for the program. It uses the self.start variable to hold the tkinter window and has rather basic functionality. It uses labels for the blocks of text instructions and a button to begin the screener. If you are familiar with tkinter, this method is rather straightforward.

3.4(d) display_survey():

Next we have the display_survey() method which creates a window in the self.survey variable and visualizes the actual screener. This method uses a grid placement for the question and answers which are placed upon a frame that is loaded onto the larger window. At the beginning of the method there are a whole bunch of extra frames and canvases which are used to create a window that can scroll. For more information on this code, please visit the link in the credits section of the file. However, the basic concept is that scrollbars in tkinter cannot be placed onto a frame. Therefore, we create a canvas that can be scrolled in both the x and y direction which we then attach to a frame that is on top of the main window. Then our grid frame with the questions is placed on top of the scrollable canvas to ensure that the entire window can scroll.

3.4(e) final_window():

This final method is for displaying the ending window after the user has completed the screener. Again it uses labels to provide some instructions for how to view your results and buttons for restarting the test and closing the application. This window is very straightforward for those with experience using tkinter and is rather self-explanatory similar to the initial menu display or the init_interface() function.

5. Conclusion

This documentation is intended to provide extended and additional information about the Mental Health & Wellness Screener Program. This documentation combined with the comments in the source code should provide a clear understanding of the program and how the modules work together.

References

[1] https://classes.cs.uoregon.edu/10W/cis422/Handouts/EyeDrawDocumentation.pdf
[2] https://classes.cs.uoregon.edu/22W/cis422/Handouts/CALCDOC_World_Tax_Planner.TXT