L.Vandecasteele (lv) 1/29/2022
N.Onofrei (no) 1/29/2022
Zacree Carroll (zc) 1/29/2022
Haoran Zhang (hz) 1/29/2022
Kenny Nguyen (kn) 1/29/2022

Cold Call Assist System
Programmer's Documentation

Table of Contents:

Program Files

ColdCallOperation.py

Global Variables - remove1, remove2, remove3, remove4. flag1, flag2, flag3, flag4, test_key

The remove keys are set to 1, 2, 3, 4 respectively and can be changed in order to change how students can be removed. flag1-4 are the variables used to store which keys flag students. test_key is the variable that keeps track of which key must be pressed to initiate testing

Class UserInterface(object):

__init__(self):

This initializer sets all the variables needed in the class UserInterface.

self.nameList records the student names currently displayed on the desk.
self.restList records the students don't in the desk but in the student data that is currently in the cold call system.
self.studentQueue is the randomized priority queue.
self.infoMap is the hashmap for storing student info.
self.classSize number of students in cold call system.
self.removedName record name of last removed student.
self.keypress and self.can_flag are components that ensures that the flag button can only be used within one second after the student is removed.
self.test_flag is an initialized variable that helps keep track of whether or not the key was pressed to initialize the testing.

firstInterface(self):

firstInterface is the first user interface(UI) that users see when they open the program, in which include 3 buttons - 'Cold Call Assist' button link with CCinterface, 'Import New File' button link with inputFile, 'Select Location to Export Student File' button link with outputFile.

CCinterface(self):

CCinterface is the UI that will be displayed in class.
It can read the student data stored in the current cold call system from the local directory, and then select four students in a random priority queue and put them into the desk, and have a back button, allowing the user back to the initial UI, that is, connect to firstInterface.
CCinterface monitors specific keyboard inputs, and the keys monitored depend on the global variables at the top of the file. The four keys to remove students from corresponding position of the desk and add new students are connected to four corresponding equations respectively, removeAndAddStudent_1, removeAndAddStudent_2, removeAndAddStudent_3 and removeAndAddStudent_4. The four keys to flagging last removed students both link with flag. The one key to run the test model link with test_flagging.
Create a blank daily log file.

warningInterface(self):

warningInterface only pops up when the user attempts to import students data by the ‘Import New File’ button in firstInterface and students data already exists in the cold call system.
warningInterface displays a risk warning text, and include two buttons - ‘continue’ link with confirmInput, redirecting the user to the file browsing window; ‘back’ button only destroy current warningInterface, so that user can back to firstInterface then save current student datas by export at first.

print_summary_line(self):

print_summary_line writes a line of log into the daily log depending on the information corresponding to self.removedName stored in self.InfoMap. If self.can_flag current is False, writing a ‘X’ in front of log as a mark of flagging.

removeAndAddStudent_1(self,event):

removeAndAddStudent_1 is an event in response to the corresponding binding key, here the key depends on the global variable, remove1.
After removeAndAddStudent_1 is called, which sets self.keypress_timer and changes self.can_flag to be True. Then, it will remove the corresponding student and add the student back to the priority queue, both getting the next student from the priority queue by updating self.removedName and self.studentQueue. Besides, removeAndAddStudent_1 updates students in the self.nameList, at the same time, reflecting this change in self.tempList, in this way to display changes in UI.
removeAndAddStudent_1 will also create a new thread, calling print_summary_line to record the remove or flagging log.

removeAndAddStudent_2(self,event):

With removeandaddstudent_1 is the same, but the binding key is the global variable remove2 and self.nameList and self.tempList

have different adjustments according to the students' position on the desk

removeAndAddStudent_3(self,event):

With removeandaddstudent_1 is the same, but the binding key is the global variable remove3 and self.nameList and self.tempList have different adjustments according to the students' position on the desk

removeAndAddStudent_4(self,event):

With removeandaddstudent_1 is the same, but the binding key is the global variable remove4 and self.nameList and self.tempList have different adjustments according to the students' position on the desk

flag(self,event):

flag is an event in response to the corresponding binding keys, here the keys are the global variables, flag1, flag2, flag3 and flag4. By the self.keypress_timer set and the change of self.can_flag in each removeAndAddStudent, flag judges whether the user has removed the student in the previous second, and limits the user to flag the corresponding student only once within the above second.

initiate_testing(self):

initiate_testing checks to see if the test_flag is true. If it is then it calls on testwarningInterface to start the process of the testing procedure.

test_flagging(self, event):

This method checks to see if self.test_flag is False. If it is then it sets it to True. If it is already true then initiate_testing is called.

inputFile(self):

inputFile judges whether there is student data in the system. If there is, call warningInterface to remind the user; If it does not exist, directly call ImportFile from ImportAndExport.py to open the file browsing window.

confirmInput(self):

When the user selects the 'continue' button from warningInterface, and ensures to replace existing students data in the system, confirmInput directly calls ImportFile from ImportAndExport.py.

outputFile(self):

outputFile call ExportFile from ImportAndExport.py.

Class Student():

__init__(self):

This initializer sets all the variables needed in the Class Student.
self.name storage name of student.
self.priority saves the priority of students being called.

__lt__(self,other):
__lt__compare priorities between two students.

main():
Start the program by establishing the UserInterface class then running firstInterface.

Testing.py

Global Variables - firstnames, phonetic, lastnames

firstnames stores a list of 70 first names and phonetic contains the appropriate phonetic spelling. lastnames contains 60 last names.

studentfile(num):

studentfile takes in an integer to create that many different students. First it creates and stores random students names, last names, and appropriate

phonetic spelling. Next the function creates a random uo id and makes sure its unique. The penultimate step is to create the reveal code which is unique to each student. The last thing is to write each students information into StudentFile.txt.

caller():

caller uses studentfile in order to generate 100 random students whose information is unique. The function then creates a queue that contains 4 students names. It then chooses randomly which one to pick and fills in the next student randomly. There is also a 25% chance of the student being flagged. Lastly, it writes the result into testoutput.txt.

testwarningInterface():

testwarningInterface uses tkinter to generate a window asking the user if they want to continue with the test. If the user presses continue then caller is called to run the test. If back is pressed than the window closes and nothing is executed.

ImportAndExport.py

A note on the use of Tkinter in ImportFile() and ExportFile():

Tkinter is used throughout the system to create user interface windows for our system however in these two cases its use has a slightly different functionality from other places. In both of these functions Tkinter calls are used to open a file/directory browser. These calls are not operating system specific and therefore increase the portability of our code.

ImportFile():

ImportFile spawns a file browsing window (using the Tkinter framework) to allow the user to select a file from their machine for importing. It calls ScanFile(userFile, False). This call effectively determines whether the file is in correct format (see ScanFile()). If an error is detected then FileError(windowName, errorMessage) is called to indicated to the user the nature of the error detected (see FileError()).

ScanFile(f, returnStudentList):

ScanFile will scan a file for the correct format. Each line in a file contains several pieces of information about an individual student which are all added to student []. After a line is finished being scanned and no errors were detected student [] is appended to studentInfoLists[]. If returnStudentList == True then that list is returned. If an error is detected then a list of strings is returned explaining what line the error exists on and which component contained the error.

ExportFile():

ExportFile spawns a directory browsing window (using Tkinter). It allows the user to select where the want the exported student roster file to go on their machine. It does not remove the current file from the system it simply copies it to another location.

FileError(error, buttonMessage, func=None):

FileError handles displaying to the user the specific errors that may have been encountered while attempting to import or export a file. It will spawn a window with the title saved in the error variable. This window will have a button with the title saved in buttonMessage. The third argument is used when there is an extra command that needs to be executed when calling FileError. ImportFile calls FileError(windowTitle, errorMessage, ImportFile) in order to allow the user to browse for the file once again when an error is encountered.