CIS 422 Project 1:
Classroom Cold-Call Assist Software
Technical Documentation

Mikayla Campbell (mc), Joseph Goh (jg), Olivia Pannell (op), Bethany Van Meter (bvm), and Ben Verney (bv)
Group: Cheetle

February 3, 2020 – v2.0

Table of Contents

1. Project Plan Revision History

Date	Author	Description
2-2-2020	op	Created the initial document and first draft.
2-3-2020	op	Finished sections 2,3,4, and started 6.
2-3-2020	jg	Added brief descriptions of each file in section 5
2-3-2020	jg	Finished initial draft of ui.py documentation
2-3-2020	mc	Added to save_data.py technical documentation
2-3-2020	bvm	Added descriptions and helped finish document
2-3-2020	jg	Unified document formatting

2. Installation Help

2.1 How to install

Note: This can also be found in Installation_instructions.txt.

1. Download the zip "G3ColdCall.zip"
2. Unzip the file "G3ColdCall.zip"
3. Double click ColdCall to open. *NOTE: ColdCall and Cold_call_files must be in the same directory.*
4. If ColdCall opens without error skip to step 8. If ColdCall does not open due to access rights you will have to change the access rights yourself. Please continue to step 5.
5. Open Terminal and cd to the directory that ColdCall is in.
6. Type "*chmod 777 ColdCall*"
7. Open ColdCall again by double-clicking
8. Click the "Import" button in the Cold Call window.
9. Find the class roster and open it. See 2.2 for the correct file formatting!
10. You are ready to use Cold Call for class!

2.2 Correct File Formatting for Import

Note: This can also be found in cold_call_files -> docs -> technical -> File-Formatting.md.

The student roster needs to be a ".txt" file that is TAB-delimited, ideally generated by Excel. The first line of the file needs to contain the following (also TAB-delimited) header: "FirstName LastName ID-Number Email ColdCallCount Flag FlagCount". The rows that follow should contain student information in the format according to the header above with zeros replacing each ColdCallCount, Flag, and FlagCount field. Items are to be updated in place to reflect the current information after each action. The Python Standard Library's CSV module contains all necessary functionality to simplify parsing and writing.

3. Software Dependencies

Cold Call depends on imports from Python standard libraries such as:

- ***Tkinter*:** used to create the graphical user interface of the project.
- ***OS:*** manipulates paths needed for file navigation.
- ***Random:*** shuffles the order of the queue for true randomization.
- ***CSV:*** used for necessary functionality to simplify parsing and writing
- ***Daytime:*** allows every time ColdCall is opened to be logged with the date.

4. Versions of Components

Cold Call is intended to be run on a MacOS version 10.14. To run this project, a version of Python 3 must be installed on the computer.

5. Relationship Between Files

- ***ColdCall*:** Executable script that runs *ui.py*.
- ***ui.py*:** Contains the main GUI. Loads and displays data according to user interaction using code from *create_queue.py* and *save_data.py*
- ***create_queue.py*:** Contains the class definitions, Student and Roster, which define the structure in which student information is stored, as well as the functionality for parsing input files, the deck, choosing which students will be on deck, and updating the student data in memory as necessary.
- ***save_data.py*:** Contains data saving/logging functions that take Roster objects (as defined in *create_queue.py*) and can save their states as files on the computer, as well as output user-readable logs and summaries. Usually invoked after each action by *ui.py*.
- ***log.txt:*** A log of actions, appended according to *save_data.py*
- ***order.txt:*** A text file used to save the execution state of Rosters, written according to *save_data.py*

- ***coldcall.ini:*** Also a config file used to save the execution state of Rosters, written according to *save_data.py*

6. Breakdown of Files

6.1 ColdCall

This is the executable script file that runs the program when double-clicked.

6.1.1 Code

1. ***cd "$(dirname "$0")":***
   changes to the current working directory
2. ***set backslash_quote="yes"***
   osascript -e "tell application \"System Events\" to\ set visible of some item of (get processes\ whose name=\"Terminal\") to false"
   Clear:
   Makes the terminal window disappear. Derived from: “10.4: Hide Terminal's Window during Launch at Login - Mac OS X Hints.” RSS, http://hints.macworld.com/article.php?story=20050512095145996
3. ***python3 cold_call_files/ui.py:***
   Runs the code

6.2 ui.py

This file contains the graphical user interfaces and keeps track of the user’s keystrokes.

6.2.1 Global Variables

1. ***Variable Name: listOfNames***
   Purpose: List of names of students currently on deck in “<first> <last initial>. Format.
2. ***Variable Name: listOfSlots***
   Purpose: List of four Tkinter text labels that represent the slots displaying the names of each student on deck.
3. ***Variable Name: currentSlot***
   Purpose: Index of the current slot.
4. ***Variable Name: roster***

Purpose: Roster object holding all of the student information and cold-call deck manipulation logic for this session.

6.2.2 Functions

update_ui()

Updates the name fields in ***listOfNames*** and updates the display elements accordingly for each slot in ***listOfSlots***.
This is achieved by getting the Student object corresponding to each slot using ***Roster.get_student()***.
To be called every import/dequeue.

imprt()

Function for the user to import a class roster file, called when the user clicks on the "Import" button.
Opens a dialog prompting the user to select an input file, then, if the path is confirmed to exist, attempts to initialize a new ***Roster*** object from the file and assign it to the global ***roster*** variable.

exprt()

Function for the user to export current session summary as TAB-delimited file to the custom path, called when the user clicks on the "Export" button.
Opens a dialog prompting the user to select an output file path and saves the current state of the global ***roster*** by calling ***save_data.save_roster()***.

reset_flags()

Function to reset all flags and flag counts recorded on ***Student***s in the ***Roster***, called when the user clicks on the "Reset Flags" button.
Iterates through each ***Student*** and calls ***Student.set_flag(reset=True)***.
save_roster() is also called to record this action.

leftPress() and rightPress()

Functions mapped to left and right cursor key presses, respectively, that changes the highlighted name by one slot in each direction.
currentSlot is decremented/incremented (checking to prevent going out of bounds) corresponding slots' visual elements are updated accordingly.

upPress()

Function mapped to the up cursor key press, dequeues the currently highlighted ***Student*** denoted by ***currentSlot***. This is achieved by calling ***roster.dequeue(currentSlot, False)***. ***update_ui(), save_roster()*** and ***log_call()*** are also called to display and record this action.

downPress()

Function mapped to the down cursor key press. Functionality is nearly identical to ***upPress()***, but this function also flags the dequeued ***Student***. The call to dequeue is as follows: ***roster.dequeue(currentSlot, True)***.

6.2.3 Code/Variables Not in Functions

1. ***Variable Name: win***
   Tkinter window object which is assigned its properties and serves as a basis for other display elements.
2. ***Variable Name(s): lbl, slot, b***
   Tkinter widgets displayed on win. ***lbl***s contain text with information or instructions for the user while using the program. ***slot***s are where the names on deck are displayed. ***b***s are the buttons mentioned above.

6.3 save_data.py

The file save_data.py saves the state of the class's queue, saves the state of the class's sorted roster list, creates the class's daily log, and adds to the class's term summary when the program is either paused or exited.

6.3.1 Functions

save_roster()

Parameters - roster: Roster, dest_path: str, export: bool

This function takes a ***Roster*** object and saves it to the designated path (***dest_path***) with a tab-delimited file format. A separate file with session-specific stats and another separate file of just flagged students are also saved with _daily and _flagged suffixes to the original path, respectively.

Returns None

log_call()

Parameters - student: Student, flagged: bool, log_path: str

This function adds the record of the "cold calling", the dequeuing, and the flagging of ***Student***s to the file ***log_path*** (***log.txt***).

Returns None

log_import()

Parameters - roster_path: str, log_path: str

This function adds the record of the ***Student*** list import action to the file ***log_path*** (***log.txt***).

Returns None

log_startup()

Parameters - log_path: str

This function adds the record of the application startup to the file ***log_path*** (***log.txt***).

Returns None

6.4 create_queue.py

The file create_queue.py creates the queue and handles the randomization and queue algorithm for the cold call system.

6.4.1 Functions

Student.__init__()

Used to create an instance of a class Student.

Includes first: str, last: str, id_num: str, email: str, cc_ct: int, flag: bool, flag_ct: int initializations.

Returns a Student class object.

Student.__str__()

Returns the string representation of a Student with as:

"first last [id_num, email]"

Student.__repr__()

Returns the string representation of Student as a string as:

Student(first, last, id_num, email, cc_ct, flag_clag_ct)

Student.set_flag()

Parameters - reset: bool

If reset is set to True, the instance of the Student's flag will be reset to False and flag_ct will be reset to 0.

If reset is set to False, the instance of the Student's flag will stay True and flag_ct will increase by 1.

Roster.__init__()

Used to create an instance of a class Roster.

Includes deck_size: int, filepath: str, use_conf: bool to initialize.

Returns a Roster class object.

Roster.__repr__()

Returns the string representation of Roster as a string as:

Roster(size)

Roster.shuffle()

Shuffles the order of the ***Student***s in the ***Roster*** to be queued. This function also resets the ***self._next*** index back to the head of ***self.order***.

Roster.get_next_idx()

Returns the next ***Student*** index relative to ***Roster.students*** to be put on deck and internally handles ***self._next*** incrementing and avoids putting duplicate students on deck.

If the last student on decklist is reached, the function shuffles the students again.

Returns the int of the index of the student to be used from the roster.

Roster.get_student()

Parameters - ***n***: int, ***from_deck***: bool

If the ***on_deck*** flag is set to True, the student on deck at the n index is returned. If ***on_deck*** is False, then the n index from the student list is returned.

Returns a ***Student*** object.

Roster.dequeue()

Parameters - ***n***: int, ***flag***: bool

Uses the ***n*** parameter to dequeue a student from the ***Roster.on_deck*** list at index ***n***. The ***cc_ct*** of the ***Student*** object dequeued is incremented. If the ***flag*** is set to True, the ***flag_ct*** is also incremented.

Returns the dequeued ***Student*** object.

6.4.2 Classes

class Student

This class creates a ***Student*** object that contains the attributes ***self.first*** (Student's first name), ***self.last*** (Student's last name), ***self.id_num*** (Student's id number), ***self.email*** (Student's email), ***self.cc_ct*** (Number of times a student has been "cold called" on), ***self.flag*** (If a student has been flagged), and ***self.flag_ct*** (Number of times a student has been flagged). Within this class there are 4 functions: ***self.__init__()***, ***self.__str__()***, ***self.__repr__()***, and ***self.set_flag()***.

class Roster

This class creates a ***Roster*** object that contains the attributes ***self.filepath*** (The path of the main roster summary file), ***self.students*** (The list containing ***Student*** objects that account for the entire class), ***self.flagged*** (The list containing the indices of ***Student***s who have been flagged), ***self.size*** (The total number of students in the ***Roster***), ***self.order*** (The list of indices that are randomly shuffled and saved to indicate the order in which the ***Student***s will be put into the queue), ***self._next*** (The index indicating which ***Student*** will be added to the queue next), ***self.on_deck*** (The list of indices that represent the order, in which, the ***Student***s are currently on deck), ***self.deck_size*** (The size of the deck), and ***self.error_num*** (Error code). Within this class there are 6 functions: ***self.__init__()***, ***self.__repr__()***, ***self.shuffle()***, ***self.get_next_idx***, ***self.get_student()***, and ***self.dequeue()***.

6.5 log.txt

log.txt contains a log of the startup of the ColdCall app with the date opened. It also has an import of a roster file, and the current student on deck, how many times each student was called on, and how many times they were flagged if they were.

6.6 order.txt

This is the first file that keeps track of the session state. Order.txt is a text file that is created to keep track of the roster order in a single, TAB-delimited line of numbers. Each number is an index of what position in the queue someone is when the Cold Call window is closed.

6.7 coldcall.ini

This is the second file that keeps track of the session state. Coldcall.ini documents the roster path, on decklist, and the next index. The roster path is the last opened roster CSV file. The on decklist is a queue of comma-separated indices of the students on deck according to Roster.students. The next index is the value of Roster.next indicating which position of the cold call queue the student that is on deck next.