DIABLOS 1.0
PROGRAMMER DOCUMENTATION

Jonathan Fujii, Brendon Kieser, Cameron McKeown, Alex Petzold, Rico Williams

Updated 4/29/19

Contents

1. Introduction

The following document explains and lists important components and aspects of the DIABLOS Cold Call System by listing what they do, and how they connect and interact with each other. Basic Python knowledge of syntax and functionalities will be assumed of the programmer.

DIABLOS is written completely using Python, specifically Python 3.7.1, and its intended working environment is within Macintosh OSX 10.13 or 10.14, or Ubuntu 18.04 LTS. It heavily utilizes tkinter and

sqlite3, a GUI library and SQL database library within the default python library respectively, and also relies on an external Python library – OpenCV – in order to properly function.

For further elaboration and nuance on how functions and attributes interact with their respective components beyond what is explained in this document, please see in-line comments.

Access to the source code can be found at:

```
git clone https://Brendon-K@bitbucket.org/Brendon-K/cis422-project1.git
```

2. Files, Components, Attributes, & Functionalities

2.1 Universal Widgets, & Functionalities

The following widgets and functionalities are to be found in every, or nearly every, window class.

Basic functions such as <widget>.pack() which are native to the tkinter library will not be covered within this document. To learn more about tkinter and its functionalities and components, please see its online documentation at https://docs.python.org/2/library/tkinter.html

2.1(a) createWidgets(self)

Every Window calls upon createWidgets() unique to itself. Upon doing so, it creates ALL widgets that the window will have and then packs them into the Window utilizing the following format:

```
self.<widgetname> = widget(self, text = <Widget display text>)
```

```
self.<widgetname>.pack({“side”: “<top, bottom, left, right>}”)
```

Other attributes can also be applied with to these widgets. Further documentation on these attributes and their usages can be found at tkinter’s online documentation:
https://docs.python.org/2/library/tkinter.html

2.1(b) Back Button Widget

The Back Button widget is functionally the exact same as the quit button and is on every window except the Course Selection Window. When activated by either clicking on it with the mouse cursor or using the assigned <q> hotkey, the Back Button will exit the current window and refocus on the most recent window to be made.

2.1(c) Header Label Widget

The Header Label Widget is responsible for showing in large font text at the top of the window exactly where in the application you are.

2.1(d) Window Exit Button

In each Window within the __init__ function, it overrides the WM_DELETE_WINDOW Protocol to instead have the same functionality as the Back Button/Quit Button Widget.

As a result, exiting any window the operating system’s built-in exit button will instead invoke self.quit, exiting the current window instead of exiting the entire application.

On Macintosh computers this is the Red circular button on the top-left hand corner of the window, while on Windows and Linux computers this is the X button on the top-right hand corner of the window.

2.1(e) Keybinds

In each Window within the __init__ function and after calling upon the createWidgets() function, certain button widgets will be assigned keys. As a result, upon pressing a keyboard key, it will invoke its bound function if the window is in focus.

2.2 Course Selection Window (CourseSelect.py)

The Course Selection Window is the first window that shows when the application is started and shows the available courses and provides access to them via their respective buttons. It will load information from the SQL database if applicable and show any initialized courses.

The Course Selection window has four main buttons total: three Course Select buttons, and a Quit button.

2.2(a) Course Select Button

By default, the course select buttons will not have any courses assigned to them and are empty.

When selecting an empty course button, it will do a one-time transfer to the Course/Roster Information Window.

AFTER a course has been made, the assigned course button will have the name of the made course and when activated, the button will bring you to the selected Course Window.

Each course select button, in ordering from top to bottom, can be activated by pressing the <1>, <2>, or <3> keys on the keyboard, or by clicking on them with the mouse cursor, regardless of whether a course has been made or not.

2.2(b) Quit Button

By either clicking the button with a mouse cursor or pressing the assigned hotkey <q> on the keyboard, the application will exit. This is functionally the same as the back buttons throughout the rest of the application.

2.2(c) course(self)

A simple helper function that gets called when selecting a course. It will either open the Course Info/Roster Window or the Course Home Window if either there is no saved course data or there is already course data respectively.

2.2(d) createRoster(self)

A helper function that sets up code to properly withdraw the current window and display a new Roster window. When the Roster window is quit, this function also ensures that it's properly destroyed.

2.2(e) createCourseHome(self)

A helper function that sets up code to properly withdraw the current window and display a new Course Home window. When the Roster window is quit, this function also ensures that it's properly destroyed.

2.3 Course Home Window (CourseHome.py)

The Course Window contains a variety of different functions to utilize for its respective course. It's from here that you can either access the course's Cold Call, Flashcards, and other information to edit. The window itself has three main buttons aside from the Back Button.

2.3(a) Cold Call Button

When activated, the Cold Call button will open the Cold Call Window utilizing the createColdCall() helper function. It can either be activated by pressing the <1> key on the keyboard or by clicking on it with the mouse cursor.

2.3(b) Flashcard Button

When activated, the Flashcard Button will open the Flashcard Window utilizing the createFlash() helper function. It can either be activated by pressing the <2> key on the keyboard or by clicking on it with the mouse cursor.

2.3(c) Update Button

When activated, the Update Button will open the Roster/Course Information Window utilizing the createRoster() helper function. It can either be activated by pressing the <3> key on the keyboard or by clicking on it with the mouse cursor.

2.3(d) createColdCall(self)

A helper function that sets up code to properly withdraw the current window and display a new Cold Call window. When the Cold Call window is quit, this function also ensures that it's properly destroyed.

2.3(e) createFlash(self)

A helper function that sets up code to properly withdraw the current window and display a new Flash Card window. When the Flash Card window is quit, this function also ensures that it's properly destroyed.

2.3(f) createRoster(self)

A helper function that sets up code to properly withdraw the current window and display a new Roster window. When the Roster window is quit, this function also ensures that it's properly destroyed.

2.4 Cold Call Window (ColdCall.py)

The Cold Call window is a compact and minimalistic window that shows three students from the course roster in priority of being called, ordered from left (first) to right (last), directly from the queue. These three shown queue positions are the "deck". The instructor can remove any of the three students shown by using any of the buttons assigned to the student respectively.

2.4(a) Student Queue

The student queue is imported from and stored in the SQL database. The student queue contains each student in the roster placed in order randomly. The queue exists until it becomes exhausted to only three remaining students; at this point, the queue will repopulate by appending a queue of each student in the roster ordered at random to the end of the existing queue. This ensures that all students will called upon in an egalitarian manner.

The "local" queue will always be synchronized with the stored queue in the SQL database, as it is imported and updated with each activation of the remove button.

2.4(b) Student Slot Button

There are three of these buttons in total, each labeled with a student from highest priority to be called (left) to lowest priority to be called(right). This button is functionally tied directly to the remove() function. When activated, it will remove its respective student from the queue and update the deck.

2.4(c) remove(self, n, event=None)

The remove function removes student “n” from the queue and calls upon its helper function update_names(). The event attribute is optionally used for debugging.

2.4(d) update_names(self)

The update_names() function helps update the queue upon a student being removed from the deck and also updates the Student Slot Button text.

2.5 Roster/Course Information Window (Roster.py)

2.5(a) Edit Roster

Upon activating the Edit Roster button the user will be prompted with dialogue boxes to both create/edit the course name and to import a roster to the course (both actions are optional).

2.5(b) Import Photos

The Import Photos button prompts the user to navigate to the directory in which they wish to store their class photos. Note that the photos in the directory must follow the naming convention of <UO ID NUMBER>.jpg to appear successfully.

2.5(c) Remove Course

When activated, the remove course button will prompt the user to confirm they wish to remove the course.

2.5(d) Export Roster

Upon activation, the export roster button will ask for the directory and export the course’s roster to a txt file if it exists

2.5(e) edit_roster(self, value, event = None)

Provides the functionality for the Edit Roster button. Message boxes indicating user error are also tied to this function to alert the user if they have made a mistake or if the action is invalid.

2.5(b) import_photos(self, value, event = None)

Provides the functionality for the Import Photos button.

2.5(c) remove_course(self, event = None)

Provides the functionality for the Remove Course Button.

2.5(d) export_roster(self, event = None)

Provides the functionality for the Export Roster Button. Message boxes indicating user error are also tied to this function to alert the user if they have made a mistake or if the action is invalid.

2.6 Table

2.6(a) self.conn

This table attribute is the table's connection to the actual SQL database file.

2.6(b) data_entry(self, data)

Inserts the inputted data into a row within the table.

2.6(c) import_roster(self, file)

Opens the inputted file and reads the data into the database.

2.6(d) export_roster(self, path)

Exports the stored roster data from the database into a text file.

2.6(e) build_queue(self, backup=False)

Builds a queue randomly from students within the database of the assigned course. If the backup attribute is set to false, it will move the returned queue to the backup queue.

2.6(f) raise_queue(self)

Returns a list of student names ordered by their queue position in the database.

2.6(g) remove_student(self, uoid)

Removes a student from the database using the given uoid as a designation for which student is removed.

2.7 Main File

2.7(a) main()

Main sets up the initial variables for the application, such as its name, its database, and sets up the initial Course Selection window for the user. When the Course Selection window is quit, this function also ensures that it's properly destroyed and that the application properly exits.

3. Issues and Known Bugs

3.1 Issues

3.1(a) Exiting the Program

Exiting the program properly without using terminating commands (such as Alt-F4 on windows or Command-Q on Macintosh) requires going back all the way to the Course Select window.

3.1(b) Course Name Limitations

Due to how SQL database injections work the following cannot be within a course name:

Apostrophe ‘

Two dashes --

Semicolon ;

3.2 Bugs

3.2(a) Photo Table Desync

If the user only wants to edit the course name without updating the roster, the photos table won't be updated to match the roster of the newly renamed course.

3.2(b) Flash Card Name Desync

The flash card cards sometimes displays differing names to the displayed photos.

3.2(c) Queue Initialization for Repeated Student

If a student is within the deck twice and a new queue is initialized, one instance of the student will be removed from the deck.

4. Program Testing & Sample Data

You can utilize sample_roster_gen.py to create sample rosters and photos to use in testing and debugging the software.