Programmer's manual

Team Darkspear

December 3, 2018

Team Members

Michael Thorsen
Andrew Thorsen
Jeffrey Heredia
Rachel Tidwell

Contents

1 Introduction

The main code was written in Python 3.7.1 that links to a MySQL server. the library used for the interface is Tkinter. The parent file is Main.py. Main initiates all of the pages used for the duration of the login session including: LoginPage, HomePage, FriendsPage, ClanPage and the Zombie minigame.

2 MySQL server

1. Daily use of the server

   The database is as of now locally hosted on the computer that is running the application until darkspear obtains a domain. until then all that is required is MySQL server and a up to date version of workbench. The files to create the tables are under the MySQL file named ds_design and there is some sample data you can get by running darkspear_populate. whenever testing be sure to open and run the localhost server and be sure to run the design and populate files at least the first time before running the darkspear application. whenever planning on using the application be sure to go to the server dropdown tab and select startup / shutdown. in this window click start server and your server is ready to be used. when the domain is obtained the server will need to be on a dedicated machine and will remain on unless scheduled for maintenance. However, until then be sure at the end of your session to bring it offline and then stop the server.

2. Future plans for the server

   What is left to do on the server for our live version we will setup MySQL server on a server computer. once the server is accessible remotely each line of code referencing host='localhost' will need to be changed to direct to the domain service. The server will be maintained by scheduled backups.

3 Main.py

This file sets up the initial frame, or window, and defines the library for each page. the last line in the init definition is to load the login page using show_frame(), an easy page pamipulation function that loads whatever page is in the function parameter.

4 LoginPage.py

The login function defined at the top is given two strings, expected to be a username and password, and performs a query of the database looking for a single row that matches both the username and the password given. this then returns the number of rows given.
The login_auth function is called when the Login button is clicked. It calls the login function and sets its return value as a variable then checks to see if that number is exactly 1, if it is not then it displays the error message stating that the username or password is not correct, then it lets the user try again, after given a correct login it then makes a call to the controller to display the home page.
The confirm_quit function is called when the quit button is clicked it then displays a new window asking the user if they are sure they want to quit. If the user clicks yes the page will shut down.

the create_account function is called when the create account button in clicked. it takes the information in the username and password field and attempts an insert into the database with the given information. however there is no error given if the information is unable to be inserted into the database.
login_sub = Button(<information here>) is where the button is what defines what the button does, it sets what the buttons says, what it looks like and it callslogin_auth, it gets the username from usr_name with the get() method and gets the password from the ps field with the get() method
quit_sub = Button(<information here>) is what defines the quit button. it calls the confirm_quit function and does not pass any information to it.
create_sub = Button(<information here>) defines what the create account button does. it passes the same information as the login button but inseat to the create_account function.

5 HomePage.py

The Homepage has all of the interface functions and the underlying logic for the interface and is loaded on application startup. file starts off initializing the frame and the grid. this page mainly consists of text fields, entries and buttons. the data in the text fields are filled with the users data stored in the MySQL database. the connection with the database is initiated with the conn, or con, variable which is utilised by the curs, cursor, variable. each text field is updated when the user presses a button of the corresponding theme of the button. the order of the file reads as bio entry area at the top with buttons that call the update_bio function that takes a text entry and updates the database with the new user generated bio. under the bi is the game list info with most of the same fields but instead the button calls select_new_game which opens a new window to select a new game that the user would like to add to their list of played games. similarly the clan section has similar buttons that open windows to add new clans.

6 FriendsPage.py

The friends page displays friends of the user, as well as a chat box that can switch between one-on-one friend chats. The friend page queries for the user, and then grabs all friends of the user (by querying player list) and creates buttons for each user. Then, for each chat, it queries the messages between the user and the friend in question. The function _init_ initializes the content for the page(layout, user, list of friends and chat box.) The function update_txt allows the chat box to be interactive, as it connects to sql to insert person 1, person 2, sender, and message into the database. The function get_txt allows the message to be viewed inside the chat box: It queries for the sender, message and message time to be displayed to the screen. Finally, confirm_quit allows the user to exit the page.

7 ClanPage.py

The clan page houses all the the information relative to each clan. At initial selection the template for the design of clan pages is displayed. This is because the page is originally rendered at application startup. The initial query at the start of the document generates and stores a list of all the clans the user is a member of. This query is re-executed every time the user navigates to the clan page, this ensures that any new clan joined after system start up gets

added to their clan list. This query is then used to dynamically populate a list of clans to the main nav bar while the user is on the clan page. This can be seen in the for loop within the navigation bar initialization. It is critical to use the lambda function to assign the name of the button to have the name of the clan it represents. Without this, every button will have the same name and would only link to the page of the last most clan listed.
The rest of the main content is initialized with basic values using Tkinter's Label object for the header, Text objects for announcements, events, bio, and chat. All of these text objects have a corresponding button that calls a function to update the information.
The update_event function handles updating the events field for the clan. The function first separates the entered text into, event name, event description, and event date. To do so, it grabs the first word and assigns that to the event name, it then grabs the last two words for the date and everything else. After that, an update statement is passed to the database to commit the changes.
The update_annon function simply grabs the info in the entry field and then performs an update statement on the database.
The update_txt function is used for the clan's chat functionality. It grabs the text the user has entered, then performs an update statement on the SQL database and finally removes all values from chat field and replaces it with the newest values in the database.
The get_clan function calls all the initial values of the fields on the page for a specific clan, it does make use of some of the other functions to do this, such as get_txt which simply grabs the original clan chat without the ability to update it as well.
The confirm_quit function behaves the same as it does on other pages and closes the application.
The function clan_announcement_query grabs the original announcements field from the database and updates the Text field.

8 Zombie.py

The Zombie.py page houses the Zombie text based survival game. It generates an initial setup using Tkinter's Text object, three button objects, and an entry field with a submit button. Either the submit, or any of the three choice buttons will call the function that generates the next situation. This function is essentially a giant switch that deletes the previous values in the Text field as well as the value of the text attribute for the buttons. It then uses a giant if, else if, else block that acts as a giant switch to insert new values into them. This means that every choice must be unique without any duplicates. If not, then the result would be the same as the first instance of that choice.