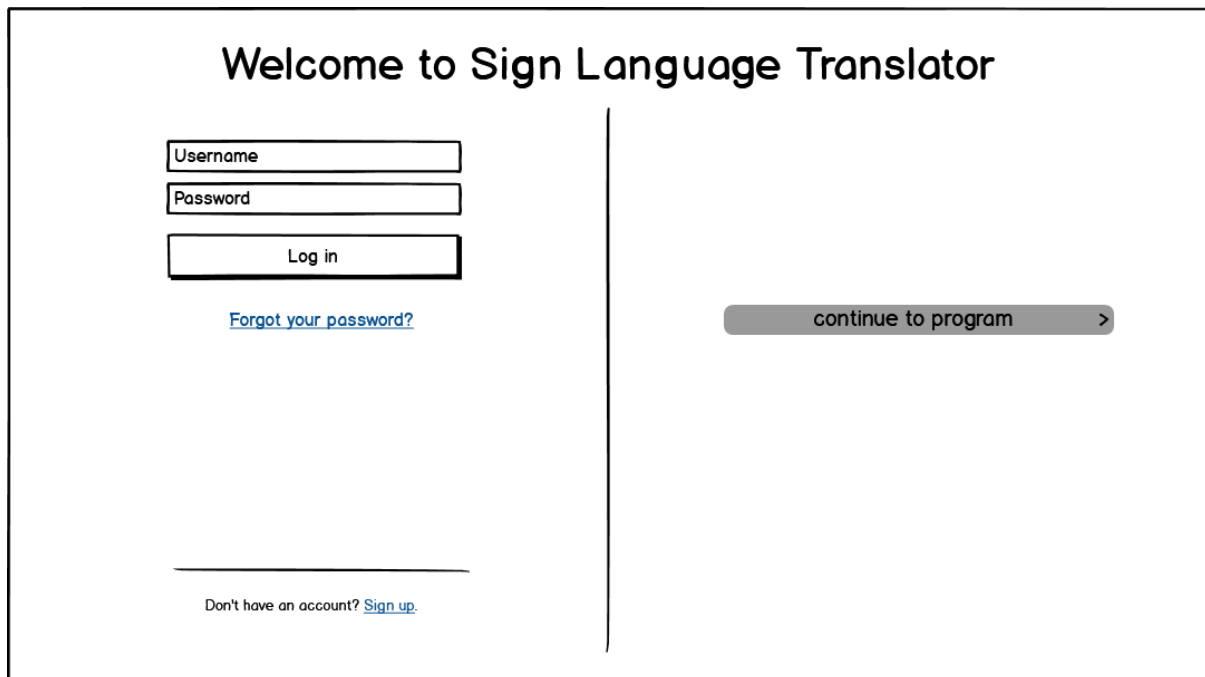


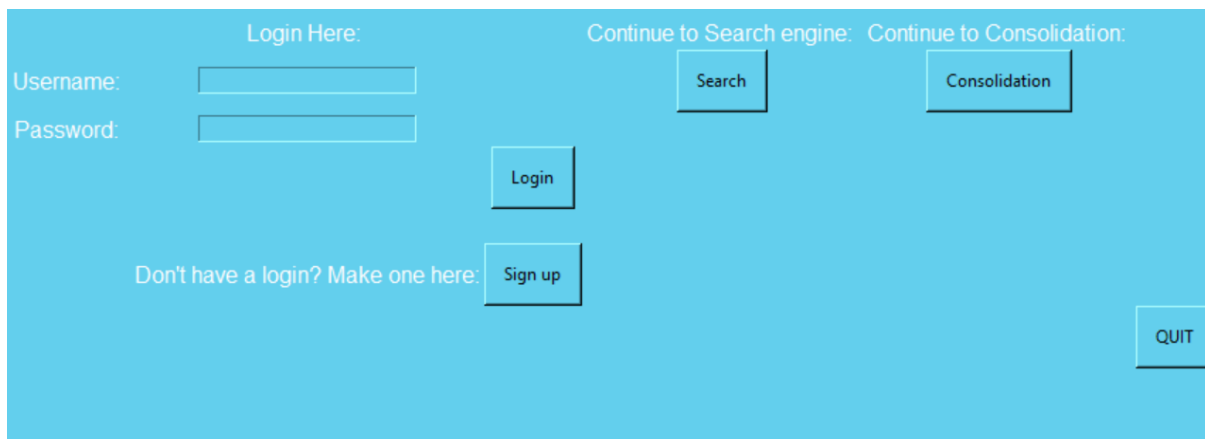
Home page

Initial Design:



The initial design is a clean, minimalist layout. At the top, the title "Welcome to Sign Language Translator" is centered. Below it, on the left, are three stacked input fields labeled "Username", "Password", and a "Log in" button. A link "Forgot your password?" is positioned below the password field. On the right side, there is a single button labeled "continue to program" with a right-pointing arrow. At the bottom left, a link "Don't have an account? Sign up." is displayed.

Actual Design:



The actual design is more complex and colorful, featuring a light blue background. It includes a "Login Here:" section with "Username:" and "Password:" labels and corresponding input fields, followed by a "Login" button. To the right, there are two buttons: "Search" and "Consolidation". A "Sign up" button is located below the login section, preceded by the text "Don't have a login? Make one here:". A "QUIT" button is positioned in the bottom right corner. The layout is more cluttered than the initial design.

Both designs boast a formal and tidy design, however, the initial design shows a more appealing simple design with access to different types of buttons and design aspects that guizero cannot provide.

Username and password:

The design between the initial and actual pages are quite similar here. Both have text boxes for the user to enter data. Both have a button to press in order to initiate the login. All of the objects are clearly labelled for the user allowing for ease of use in both designs.

Signup option:

Both initial and actual designs contain the signup button which allows the user to create a user account. Neither of the designs allow the user to input their details to sign up on the home page but instead provide a button to another page in which the signup would occur. This is effective as it saves there from being more clutter while offering the functionality.

Consolidation and search buttons:

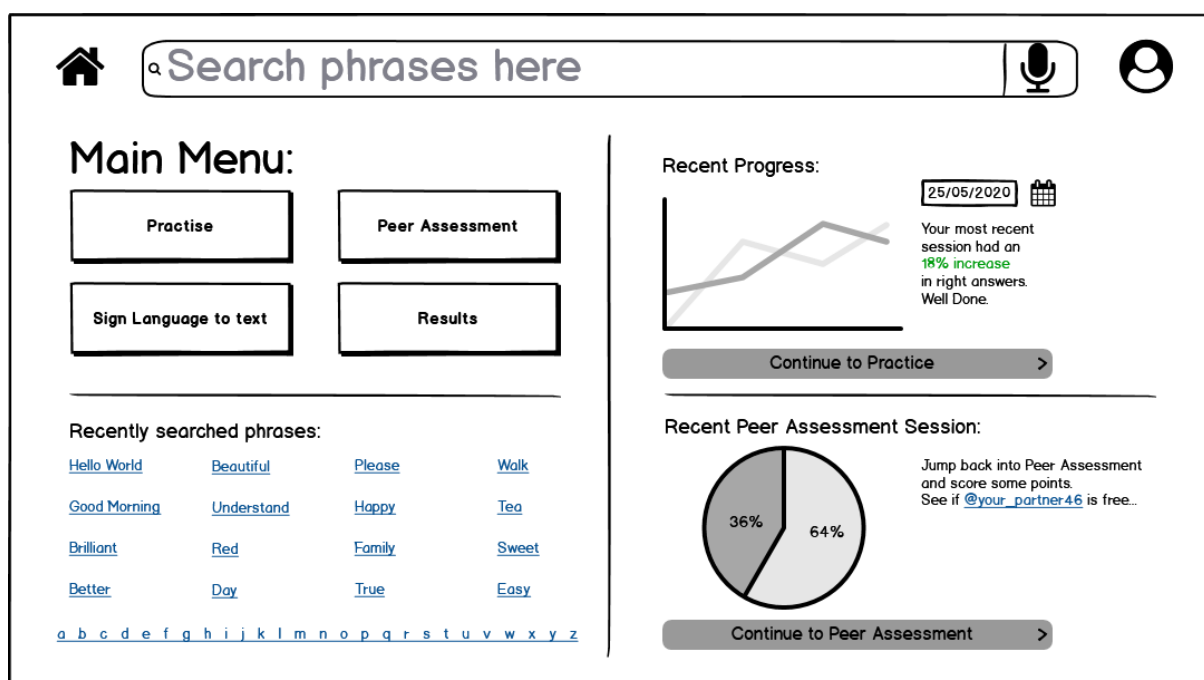
These buttons were placed on the home page with the knowledge that any user that has not signed in would be able to use the features. This effectively provides easier access to the functionality it removes a page of buffer from the initial design where the buttons would have been found on the next page.

Quit button:

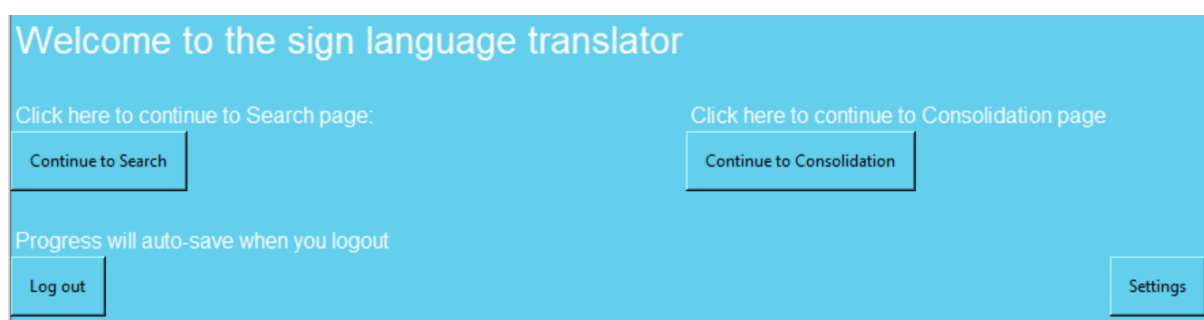
The actual design incorporates a quit button within the program for the purpose of exiting the program from the home page. This is effective as it establishes a base point for where the user starts and ends their journey in the program.

Logged in Home page

Initial Design:



Actual Design:



This page shows the biggest difference between the initial aesthetic and the actual design. This is mainly due to the lack of further functionality shown in the initial design.

Search button:

Button and description are simple and effective at directing user to the search page.

Consolidation:

Button and description are simple and effective at directing user to the consolidation page.

Logout:

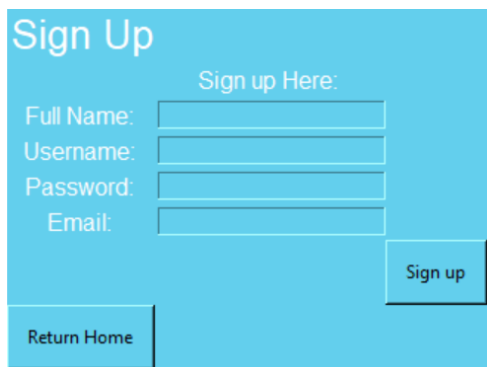
Logout button alongside information for the user saying their progress will be saved automatically when logging out is overall simple and effective.

Settings:

Button and description are simple and effective at directing user to the settings page.

Sign up page

Sign up - Actual Design

The image shows a 'Sign Up' form on a light blue background. At the top left, the text 'Sign Up' is displayed in a large, white, sans-serif font. Below this, the text 'Sign up Here:' is centered. There are four input fields stacked vertically, each with a label to its left: 'Full Name:', 'Username:', 'Password:', and 'Email:'. To the right of the 'Password:' field is a 'Sign up' button. At the bottom left of the form is a 'Return Home' button.

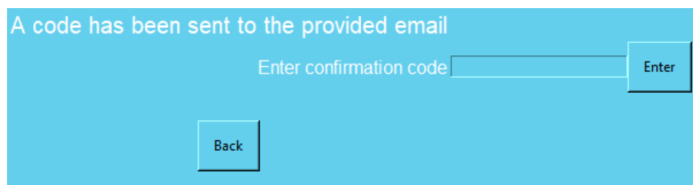
User input:

The 4 textboxes each have their own descriptor telling the user what value needs to go where. A simple and effective system that allows the user to input their data and sign up.

Home button:

A home button is added just in case a user mis-clicks and does not mean to create an account.

Confirmation - Actual Design

The image shows a 'Confirmation' form on a light blue background. At the top, the text 'A code has been sent to the provided email' is displayed in a white, sans-serif font. Below this, the text 'Enter confirmation code' is followed by an input field. To the right of the input field is an 'Enter' button. At the bottom left of the form is a 'Back' button.

Confirmation:

once the user has continued to sign up they are sent an email with a verification code. This code would be entered here as described by the instructions. on pressing enter the program will show the button to return to the homepage if the code is correct. This is effective as it only shows the exit button once the requirements are met.

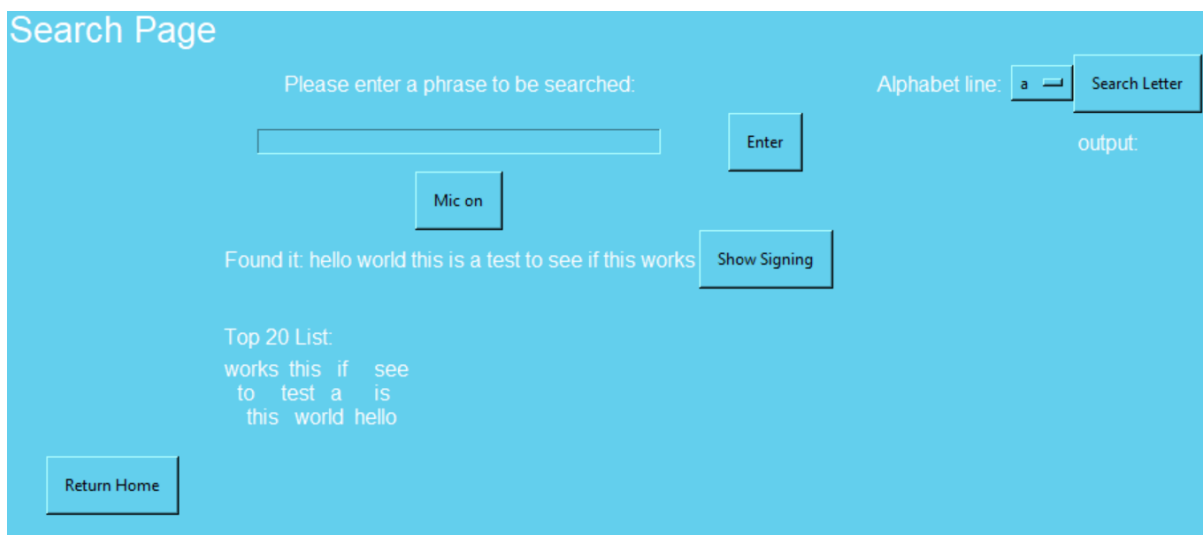
Search page

Initial Design



The initial design of the search page features a clean, minimalist layout. At the top left is a home icon. Next to it is a large search bar with the placeholder text "Search phrases here" and a magnifying glass icon. To the right of the search bar is a microphone icon and a user profile icon with a "Log in" button. Below the search bar, the text "Recently searched phrases:" is centered. Underneath, there is a grid of 16 hyperlinked phrases: "Hello World", "Beautiful", "Please", "Walk", "Good Morning", "Understand", "Happy", "Tea", "Brilliant", "Red", "Family", "Sweet", "Better", "Day", "True", "Easy", "Maybe", "Weekend", "School", and "Cool". Below this grid is a horizontal line of hyperlinked letters from "a" to "z". At the bottom, a message states: "To access the full functionality of Sign Language Translator you need to have an account." Below this message are two buttons: "Sign in Here" and "Sign up Here".

Final Design



The final design of the search page has a light blue background. At the top left, the title "Search Page" is displayed. Below it, the text "Please enter a phrase to be searched:" is followed by a text input field and an "Enter" button. To the right, the text "Alphabet line:" is followed by a dropdown menu showing the letter "a" and a "Search Letter" button. Below the input field is a "Mic on" button. Below the "Enter" button is a "Show Signing" button. The text "Found it: hello world this is a test to see if this works" is displayed. Below this, a "Top 20 List" is shown with the following text: "works this if see", "to test a is", and "this world hello". At the bottom left is a "Return Home" button.

Search options:

Both designs offer a search bar for the text search and a button for the microphone input. These are both simple and effective designs.

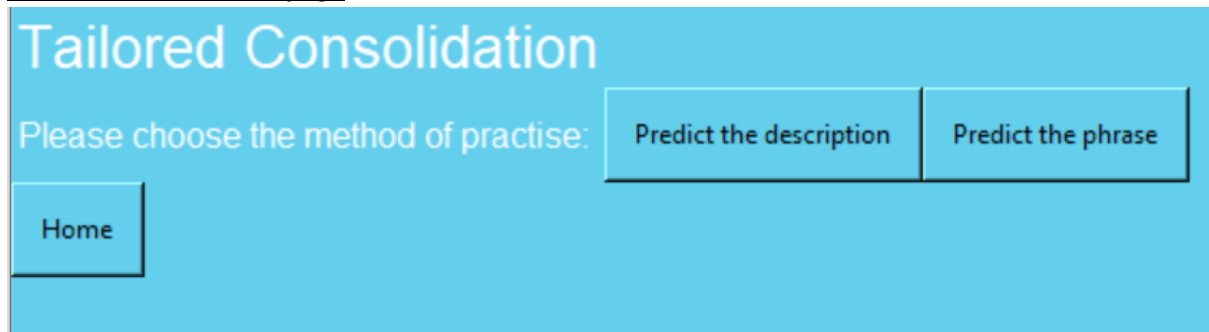
Alphabet line:

The format for the alphabet line is different for the final design as it does not incorporate a line of hyperlinked letters. Instead the alphabet is presented in a dropdown box which then presents the phrases that begin with the selected letter. While this method is effective it doesn't allow the user to click on the phrase and see it. The user must enter the desired phrase and search for it.

Top 20 list:

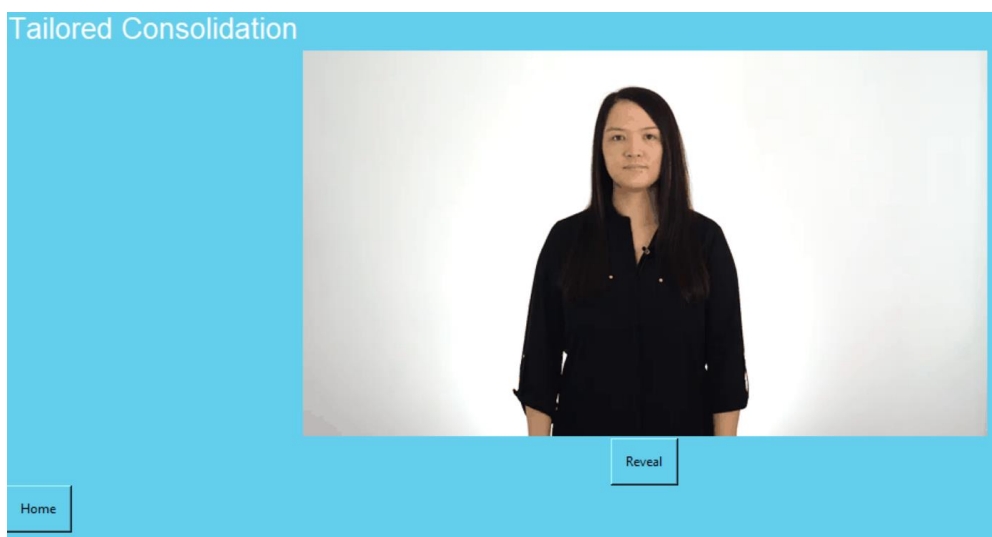
The top 20 list has a similar approach as the alphabet line where the phrases are presented identically however, they are not hyperlinked. The user must type the phrase which is a slightly less effective solution.

Tailored consolidation page



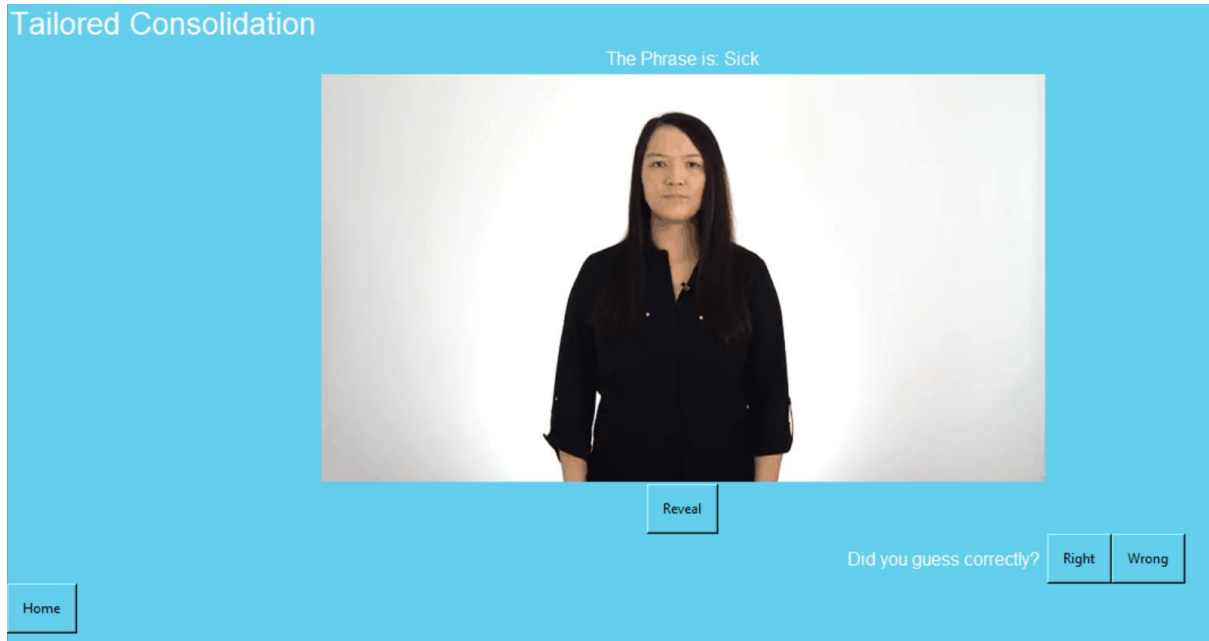
Methods of practise

The methods of practise are presented to the user by a choice of 2 buttons. Each of which would initiate the given method of learning. This is simple and effective.



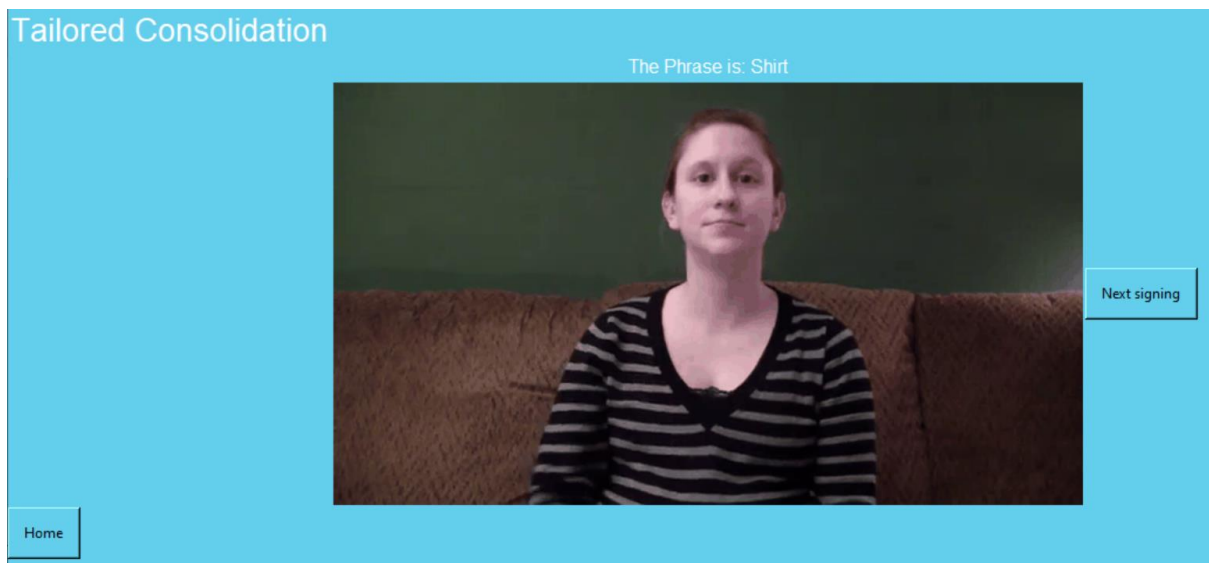
Reveal button:

A reveal button shows the description of the phrase if the "predict the description" method had been chosen. Conversely the reveal button would display the gif if the "predict the phrase" button had been clicked. Simple and effective.



Feedback buttons:

Two buttons are provided to the user to dictate whether they attempted the signing correctly or not. The lack of other options and apt descriptions makes this simple and effective.



Next signing button:

Serves as an iteration button to present the next phrase. This is the only option for the user as the other options are hidden after being pressed. This simplifies the options making it effective.

3.4.4 Maintenance and development

3.4.4a Discuss the maintainability of the solution

When designing the project, it was important to decompose each problem into its smaller chunks in order to better develop a solution, by first focusing on each individual chunk. Doing this throughout the project has created a modular design that is vital to the maintainability of the solution. Maintaining the solution includes keeping the libraries up to date, fixing errors or developing more features into the program.

Some of the methods I used to streamline these processes in the future for me or other developers are:

Having a modular approach:

As mentioned before in the project, the data manipulating parts of the program are all contained within separate functions. this modularity allowed the program to utilise any given function anywhere in the program and allowed the program to be more efficient with less bloat code. Each of these functions were kept within the same program for the sake of the project code listing, but could easily be split into separate files increasing its modularity for other users.

The project boasts 33 data manipulating functions, each of which could potentially be accessed and used by another project or person. This makes the access to the program much more available and understandable making the chances of major bug being solved much quicker.

Using external modules/data:

A total of 15 external modules are utilised in this programming project. using external modules means that tasks will be completed with greater efficiency alongside the ability to offer functionality that would otherwise take much more work and a lot more time to produce. External modules also mean that other developers looking at the code will better understand certain methods that they may already be familiar with, instead of trying to understand self-developed code that would do the same thing.

Saving data to an external file such as the online database for both BSL phrases and User profiles allows trusted developers to access this data which they would not be able to do if the data was stored locally. Having the data in a database also allows the data to be presented in a more understandable way than a long printed string.

Well commented/written code:

Having descriptive identifiers is essential to the ability of a developer to distinguish the processes within an algorithm. This, alongside descriptive comments and concise docstrings allow for a more streamlined approach to understand the processes created by a function and best way to adapt or fix it. This is vital to the maintainability of the program especially for the data manipulating functions as they will contain the manipulation of similar data sets.

```
> def add_user_to_database(user_object): ...  
> def updating_row(): ...  
> def existing_user_login(username, password): ...  
> def create_random_practice_list(): ...  
> def starting_consolidation(): ...
```

Here the functions are given descriptive identifiers to give the developer a brief understanding of what was contained within the function.

```
def add_user_to_database(user_object):
    """function abstracts all of the data from the user object and appends the
    data to the database. This data is added to the online database"""
    temp_list = []
    #values are added to an array
    temp_list.append(user_object.user_number)
    temp_list.append(user_object.name)
    temp_list.append(user_object.username)
    temp_list.append(user_object.password)
    temp_list.append(user_object.email)
    temp_list.append(user_object.date_joined)
    temp_list.append(user_object.started_TL)
    temp_list.append(str(user_object.learning_path))
    #user's consolidation data saved as a list within a string which is then extracted on retrieval
    #array added to the online user database in a row
    user_worksheet.append_row(temp_list)
```

Here you can see a docstring showing a description of what the entire function does as well as comments describing the processes within the function.

Parameterisation

Some aspects of the program are used multiple times for slightly different reasons. Parameters are passed through functions when they are called so that different pieces of data from different places can be manipulated in the same way.

```
def creating_user_object(name, username, password, email, user_number=None,
date_joined=None, started_TL=None, learning_path=None, total_right=None, total_wrong=None):
    """creates and returns a user object using values from signup page"""
    global user_object
    user_object = User(name, username, password, email, user_number, date_joined, started_TL,
learning_path, total_right, total_wrong) #creates user object
```

This function is used in order to create a user object for the signed in user. This can be used after signing up or when logging into the program. At different points in the program the user will have different pieces of data and therefore the use of parameterisation requires the essential data to be accepted and the extra data to be passed through if it is present.

3.4.4b Discuss potential further development of the solution

When beginning the project, it was important to have an idea that was easily expandable in what could be implemented into the program. With the sign language translator there were many places where more time could be spent to develop more pieces of functionality in order to bring about a broader and more intuitive program. Some of these extra pieces of functionality were incorporated into the final program such as email confirmation and multiple forms of tailored consolidation of learning. The final program serves a solution to the problem proposed at the beginning of the project as it provides a learning platform for people to learn sign language. However, it misses out on pieces of functionality that were mentioned which could still be implemented.

Without the time and money limitations on the project more features could be added to the program in order to make it more well-rounded.

During a second phase of development more time could be spent on developing the following features:

Peer assessment functionality:

This piece of functionality would use a partner system to gamify the learning of sign language with a friend. Using an emails system and storing the username of the buddy within the user profiles themselves, the peer assessment functionality was within arm's reach for this program since a lot of the framework had already been built. Due to time limitations however, I wasn't able to begin the integration and testing process.

Sign language to text functionality:

Originally the most interesting part of the project turned out to also be the most difficult. Sign language recognition requires very specific and complex algorithms that may include machine learning. This would require me to create an entire frame work with a module that allowed camera access by the program in order to obtain the relevant data from the user. one way of doing this better is with an inferred camera which are fairly expensive in order to create more accurate algorithms. In a second phase of development it was be vital to begin by researching existing modules and libraries that may help, given the overwhelming complexity of it.

More types of sign language:

Similar to the peer assessment functionality this extension to the project was within arm's reach also. With the online databases already set up in a standardised way it would be simple to implement a new database for a separate type of sign language. The problem I ran into with this was the acquirement of sign language videos to put into the database. Without creating your own videos, sign language videos are few and far between making them extremely hard to normalise into a database. I experienced this while creating my BSL database which has roughly 300 phrases currently. During a second phase I would spend time creating more databases with more phrases in order to accommodate a wider range of expressions.