# Testing to Inform Evaluation

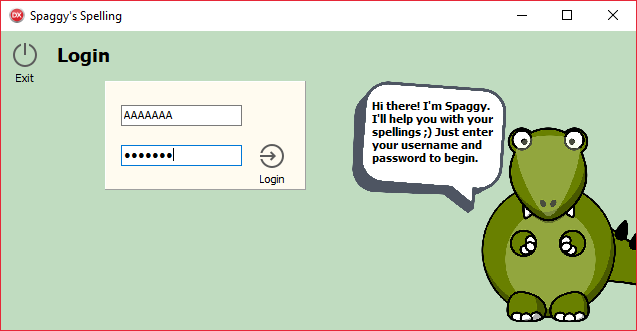
**3.4.1 Testing to inform evaluation**

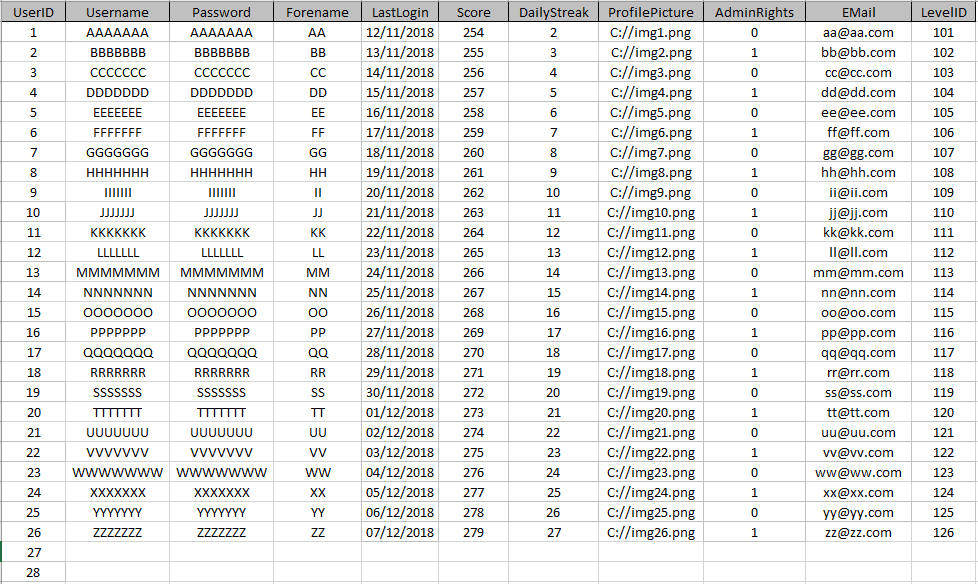
**(a) Provide annotated evidence of testing the solution of robustness at the end of the development process.**

Now that the development process has finished, the produced program must be tested as a solution to the original problem.

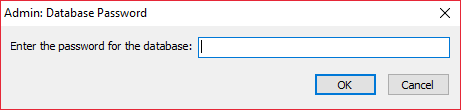
*‘The problem I am going to undertake is to design and produce a functioning prototype for an orthographic assistance program that my intended client would want to purchase.’*

As a whole, the developed program offers a working solution to this problem. Firstly, the program utilizes a user database to store user profiles, allowing for tailored learning experiences, as mentioned in the success criteria. When you first start up the program, you are met with a logon screen:

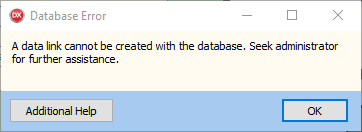


The user input is checked against the user database, containing all usernames and passwords.

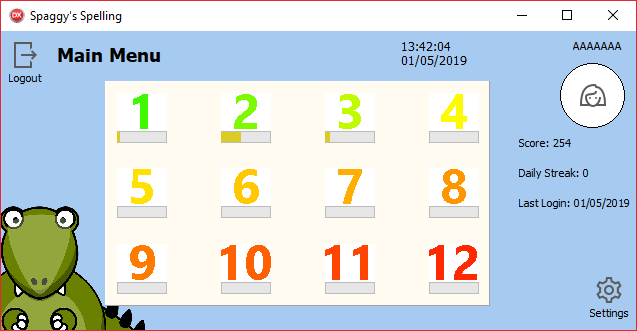
If the username or password is incorrect, the user will be denied access to the program. Thus, the log in function works as expected, and handles abnormal inputs with ease. There has been no tested method of entering the program without using a working username and password. Furthermore, the database is secured by a password, so sensitive user info is held safe.



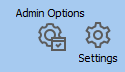
If the password is incorrect, the user is met with:



Different users have different administrator rights. Those with admin privileges are given extra options to view/add/remove users from the database.

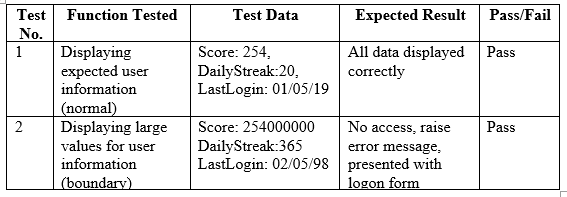


Normal users are shown the following:

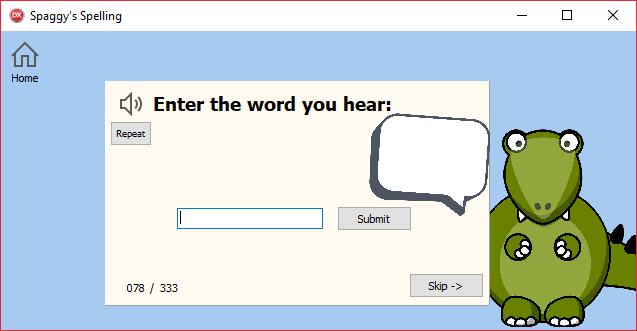


Administrators are shown the following:

Once logged in, the user is met with this main menu. In the centre, a panel shows all the game levels available to the user, along with bars denoting their progress into each. Buttons appear on the top left and bottom right of the screen, offering navigation back to the login screen and further into the settings/options. Finally, along the right hand side, the user is given a brief overview of their user profile. The score and daily streak offer incentives to play the program well and regularly. All these values function as expected. For example, the daily streak is only incremented when the last login was the previous day.

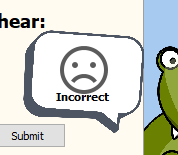


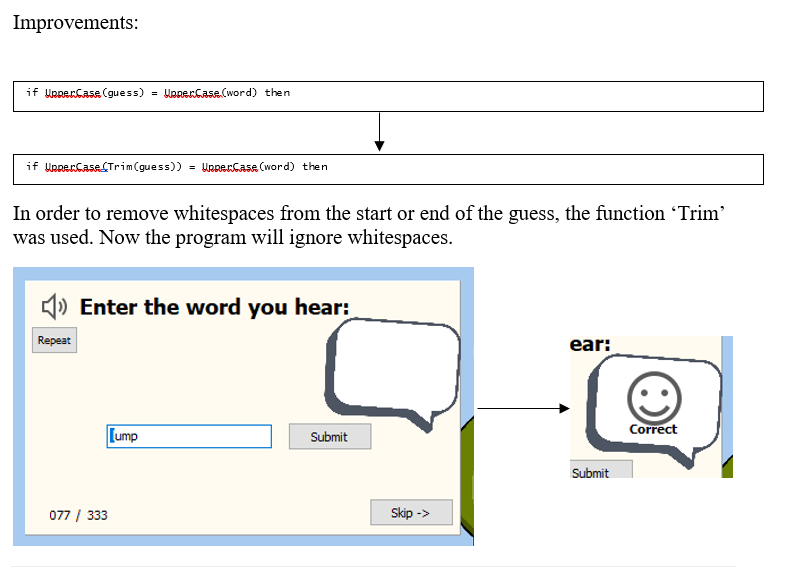
From the main menu, levels of varying difficulties may be accessed. This is where most of the coded solution resides. Upon clicking a level, the user is met with the following screen:

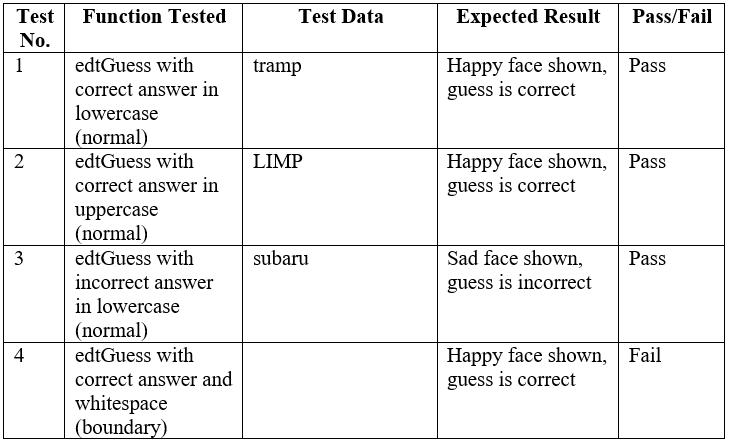


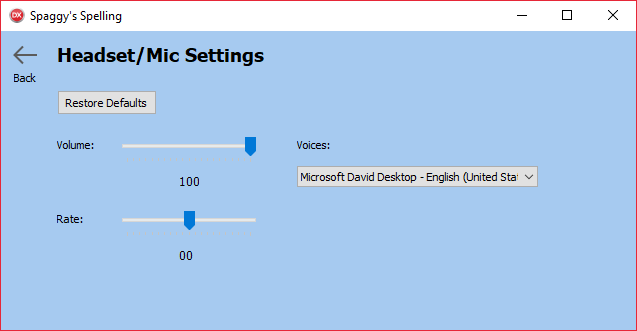
Again, this UI form is simple and clearly laid out, presenting only the main game and a button to go back to the main menu. When entering a level, the Speech API immediately says the word to be tested on the user. The user then enters their guess into the edit box and receives an immediate response to if they spelt the word correctly. The program provides orthographic assistance.

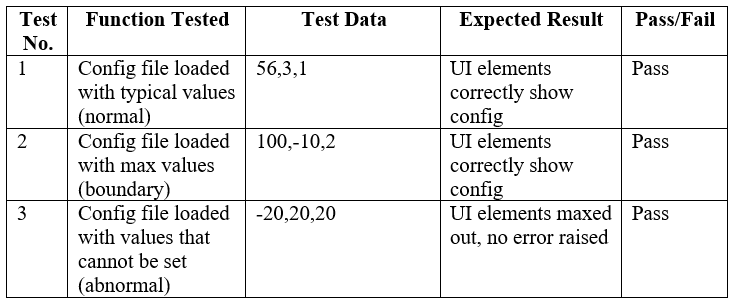


The test data below shows that this function worked as expected, and was able to handle most normal inputs. The program did stumble up on factually correct boundary inputs, though improvements were made to this issue and it has been mitigated.

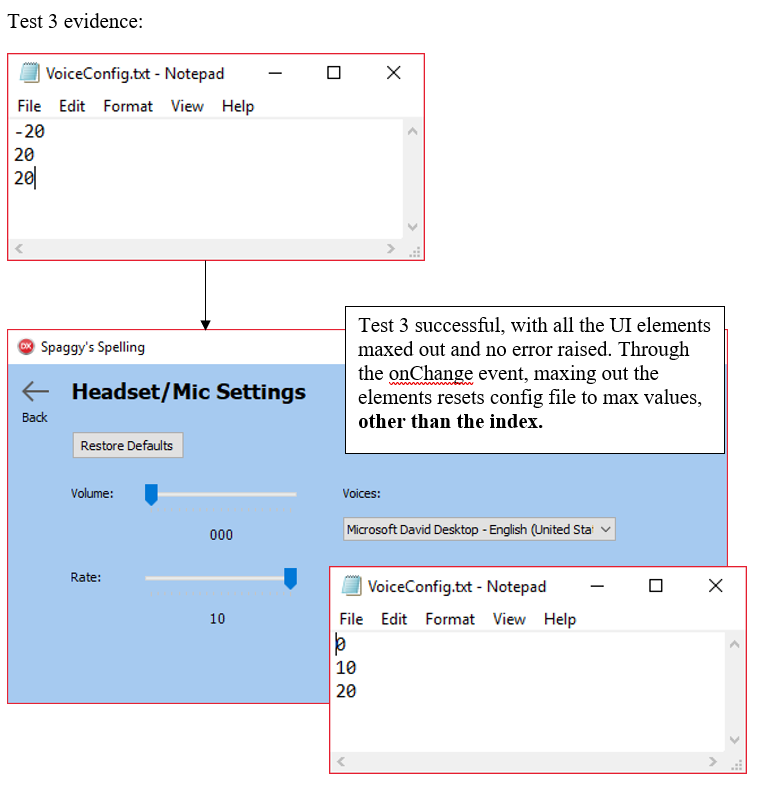
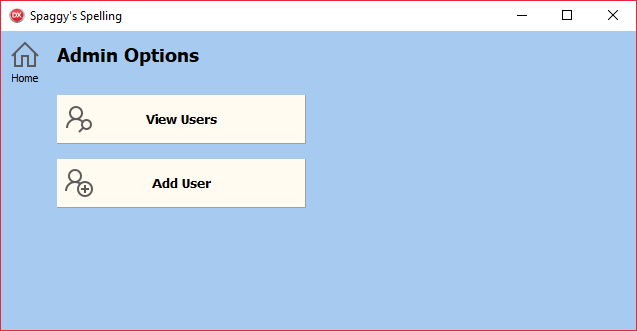


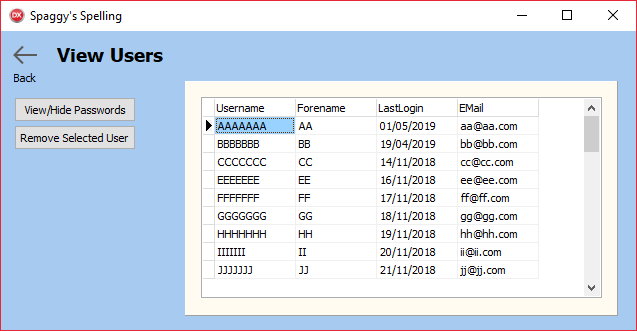
In use, it is found that the program can handle any abnormal inputs, such as numbers or special characters, and produce no errors. As long as the user’s guess is correct, the program will respond as intended. Furthermore, the Speech API encountered no problems in speaking the words contained in the flat-file database. Whilst the output of the speech engine cannot be documented on paper, it’s use in real-world testing showed that it was clear and easily understood. User testing confirmed this with the younger target audience.

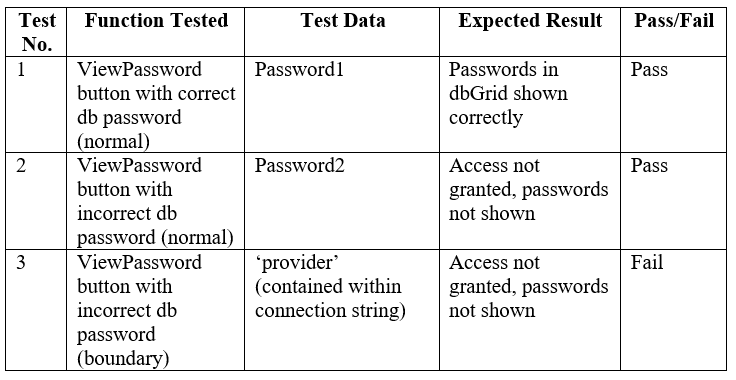
Other than the orthographic assistance game, the solution also provides configuration through the ‘headset settings’ form. It allows the user to customize the volume, rate and voice of the speech engine.

This form was also deemed robust through extensive testing of the configuration file.

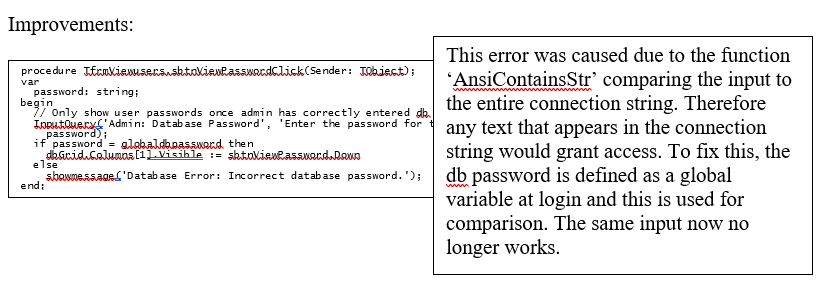
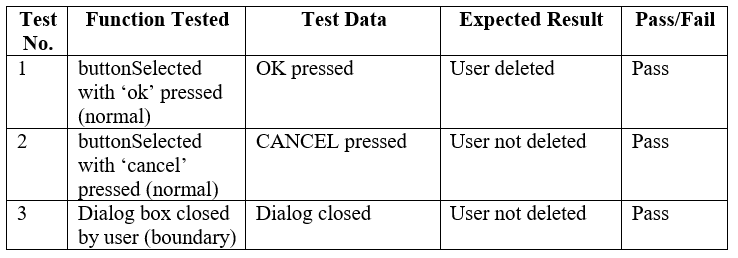
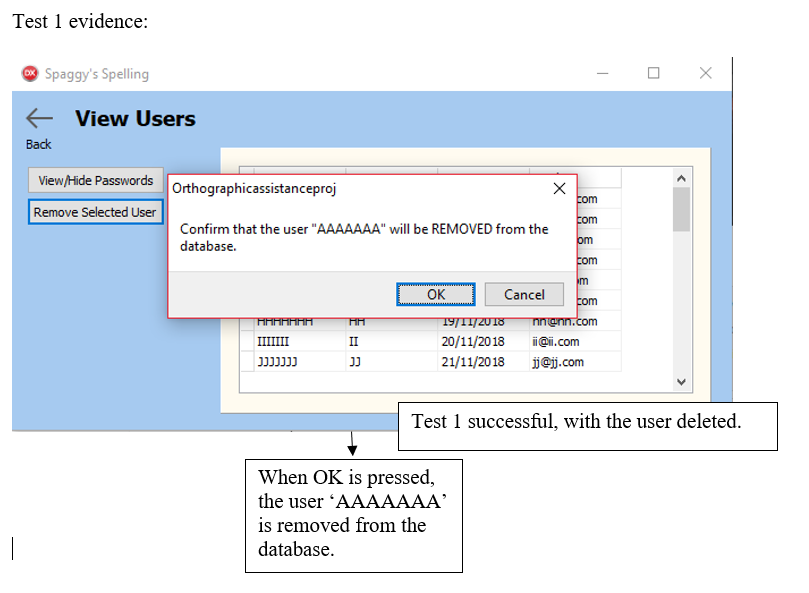
The user can configure the speech engine through any combination of values for volume, rate and voice. Even if the config file is changed outside of the program to include abnormal values, the program can still handle the input. Changes made to the config file are reliably applied to the speech engine contained within the level form.

Finally, the solution provides options for administrators to maintain the user database. As part of CRUD, the solution meets the updating of data with the updating of game progress. Creating, reading and destroying data are all available functions in the admin options pane.

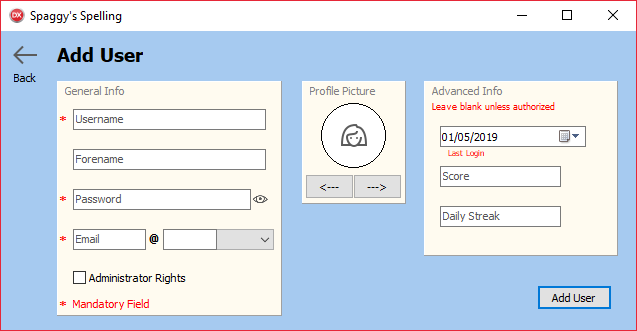
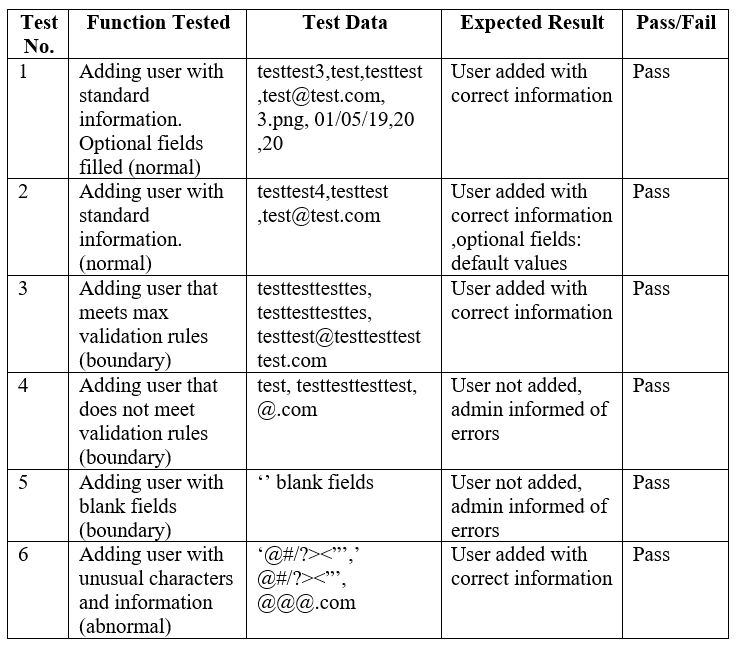


From the view users form, the admin can read through all currently registered users. The view/hide passwords button provides an extra layer of security to view sensitive information, by asking for the database password.

Although test 3 failed, improvements were made to secure this function behind the password. In the current version, the admin must know the database password to view passwords and remove users from the database.

The remove user button also passed testing for robustness. The verification and confirmation of this sensitive action are coded well enough to prevent an admin from unintentionally removing a user.

From the add user form, the admin can add a new record to the user database. In order to conform to the validation rules setup in the database, this form contains code to validate all data entry. This validation was tested for robustness.

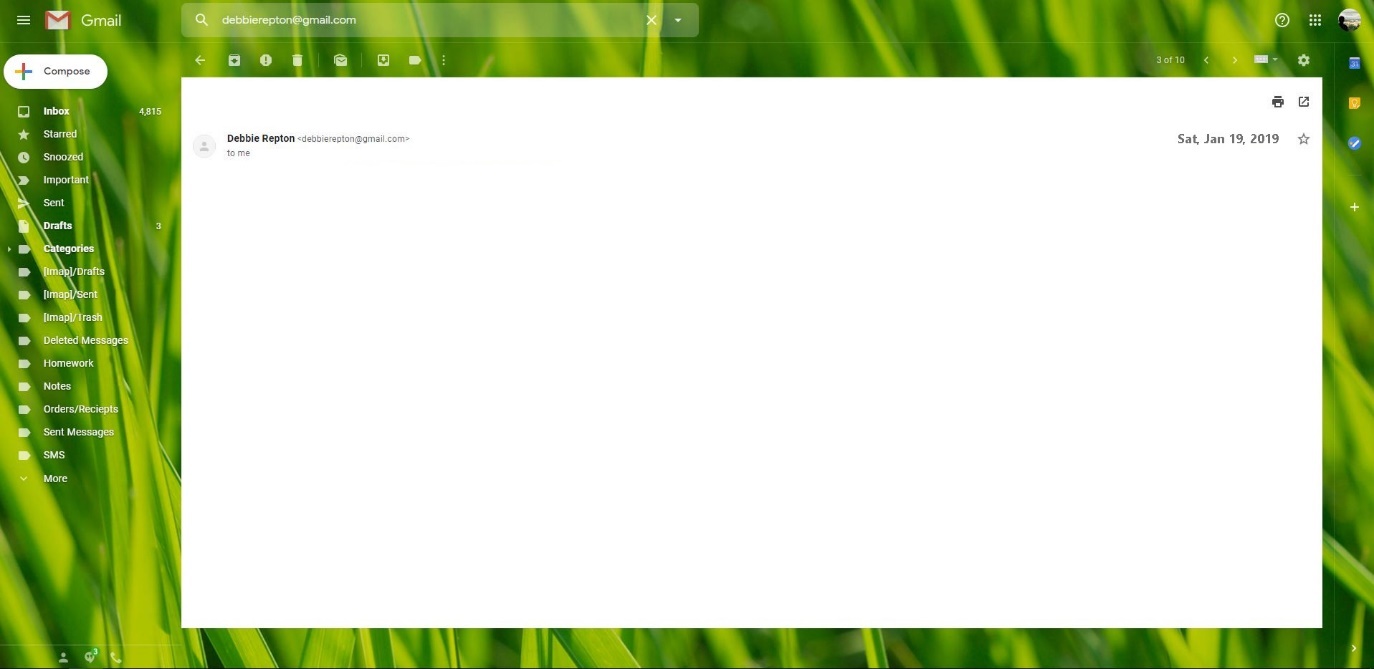


This testing shows that the program can handle normal, boundary and abnormal data. Therefore, the add user form is robust enough to correctly add a user to the database without encountering any validation errors.

Overall, the solution can be deemed robust. When the program was unable to handle boundary and abnormal inputs, improvements were made to mitigate this. In its current form, the program can handle all the tested scenarios without producing an error. Any errors that are displayed are the ones to inform the user of their failed data entry.

**(b) Provide annotated evidence of usability testing (user feedback).**

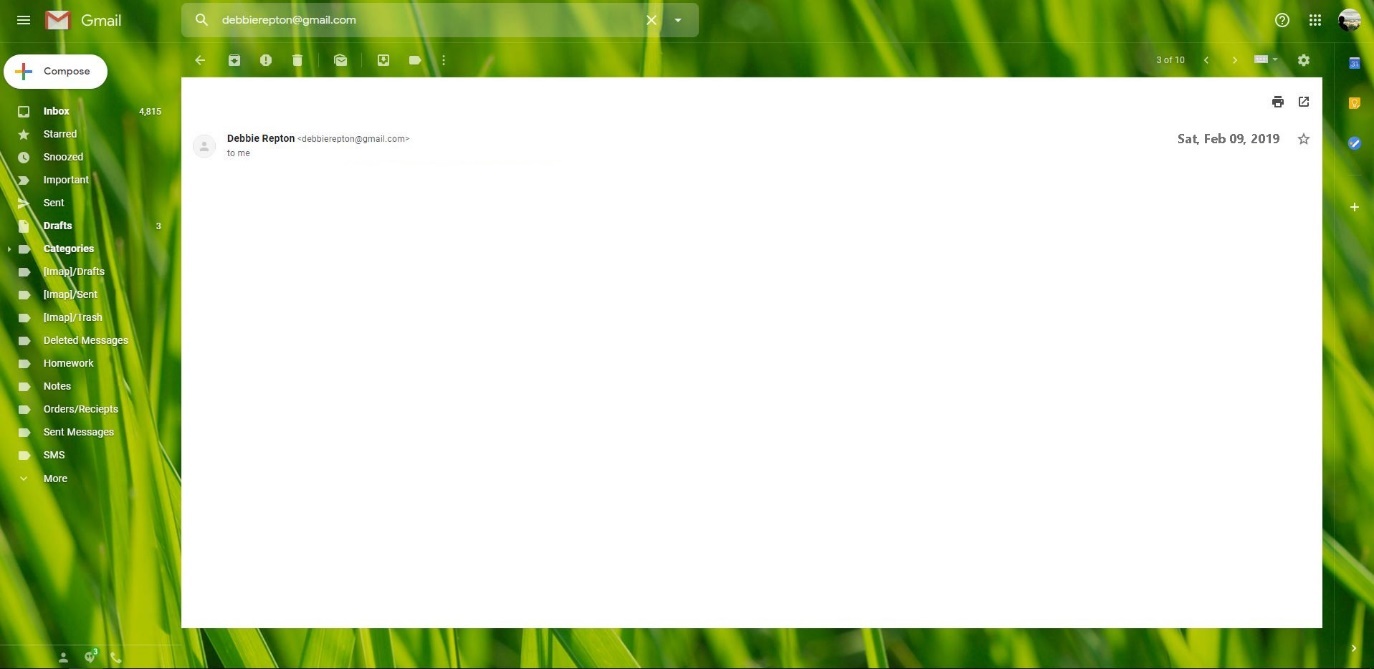
**During Mid-Development**



Dear Michael Repton,

Concerning the Orthographic Assistance program, I have looked over the designs you have sent me and have come to conclusions on the following:

1. The colours used are aesthetically pleasing. The light blue and cream are bright and contrasting, being complimentary of each other, and are a vast improvement to the design from the dull grey. The children are particularly fond of the chosen colours. No further changes need to be made to this.
2. Navigation through the program is fairly simple. The older users of the program (ages around 8 years old), are able to read the button captions and titles to understand the various screens and what they can do in them. Whilst the younger children struggle to read some of the captions, they are still able to get a grasp on what to click on. The only thing I would ask for would be a short caption on the login screen, with brief instructions on what to do. Otherwise, the instructions are clear enough for the program to be usable.
3. In general, the graphical interface is very consistent. I particularly like the use of panels to group objects together. They do a lot to simplify the program and make it less overwhelming. The icons are also effective at pulling the whole design together. The fact that they all look the same makes the program appear more professional. My request for visual buttons has been successfully met, no further action required.
4. I cannot make any comments on the usability of the program, since you have not given a working prototype …

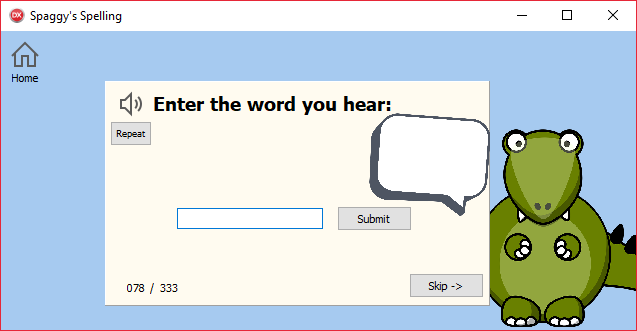


**During Post-Development**

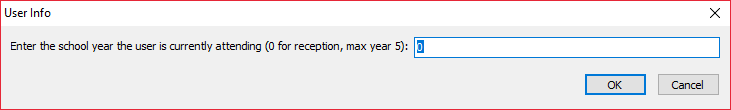
# Success of the Solution

**3.4.2 Success of the solution**

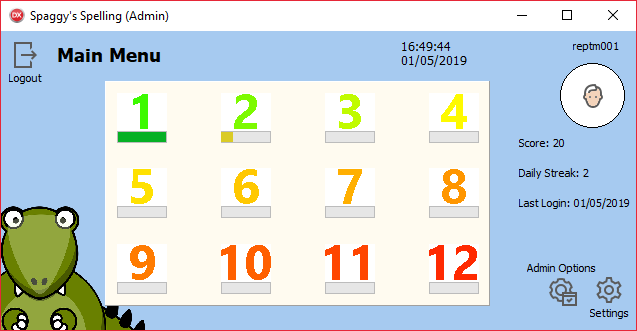
**(a) Use the test evidence from the development and post development process to evaluate the solution against the success criteria from the analysis.**

1. The solution will provide orthographic assistance for children aged between 4 and 8. The implementation of the solution will offer users the ability to improve the accuracy of their spelling. By using the program, users will be able to accurately spell at least 10 words that they could not previously spell.

This criterion is fully met. By utilizing the speech engine to speak the word that is tested on the user, the user is able to test the accuracy of their spelling. With the way that the game is set out, it offers users 3 attempts at spelling the word correctly, and then shows the correct spelling if this is not achieved. This allows the user to learn the actual spelling. When replaying the level, they will remember the word they could not previously spell and enter it in correctly. Through this progression, the user is given orthographic assistance. Through user testing, it was found that every individual in the sample set were able to spell 10 new words after 2 days of use.

1. The solution will provide a tailored learning experience which will be unique to each individual user. The experience will be tailored to current level of knowledge and the user’s ability/confidence to learn new words.

This criterion is partially met. Through this dialog box, the admin is able to remove levels containing words that the user has already learned, or should’ve already learned (according to school years). This tailors the learning experience between users belonging to different school years. The skip button also gives users the ability to skip words they find difficult, which is another way to separate the experience from user to user. However, users within the same school year will find that they learn exactly the same words, as the word banks for each level are kept the same. This could be addressed in further development by randomizing and showing a selection of words from word banks of varying difficulty. This would further vary the experience felt by different users.

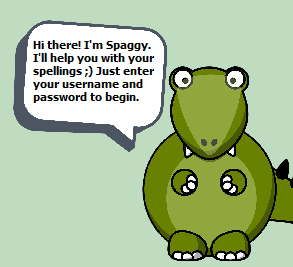
1. The solution will contain a graphical user interface, which is aesthetically pleasing (according to a consensus). The design will be consistent and easy to follow, appealing to the intended audience of children aged between 4 and 8. Visually, the interface will contain bright colours and contrast.

This criterion is fully met. The use of bright colours has been deemed to be aesthetically pleasing through user testing. They specifically point out the simplicity in the UI design, which confirms that the program is easy to follow. The use of the icon package also adds consistency to the graphical interface. Finally, the bright colours are engaging to the young audience, since they are heavily contrasting.

1. The solution will contain a graphical user interface, which is functional (according to a consensus). The intended audience of children aged between 4 and 8 will experience little difficulty in manoeuvring through the program, with the use of large, tiled icons and symbols.

This criterion is fully met. Since the program is only divided into 5 forms (for a standard user), it is very simple to navigate through. Icons and captions are made clear as to where they lead and through user testing, it was confirmed that program could be easily understood by the target audience. The icons were particularly useful since most 4 and 5 year olds cannot read or understand the navigation captions.

1. The solution will output instructions which are easy to interpret and follow for the intended audience of children aged between 4 and 8. A user will be aware of all the features the program has to offer.

This criterion is partially met. The mascot was used as a means of explaining a lot of the programs functions and operations to the young target audience. Although the program is easy to navigate through, the instructions are still too complex to be read by 4 or 5 year olds. Most of the users in the user testing required additional help from an adult, and this would probably be the same in real-world testing. This could be addressed in further development by utilizing the speech engine to speak the words that can’t be read. The instructions could also be further simplified for the young target audience to understand better.

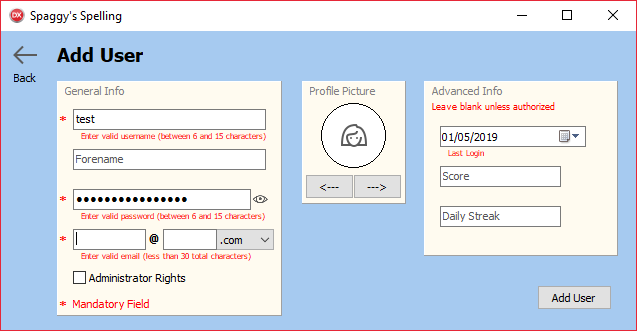
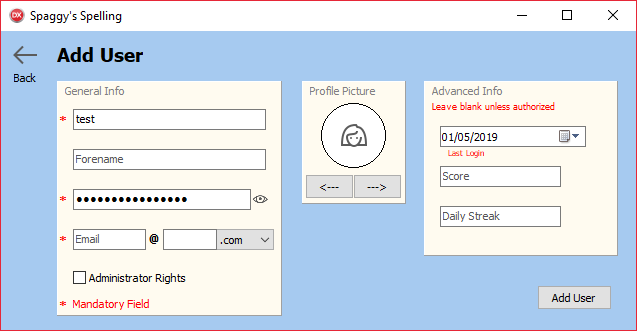
1. The solution will contain user-defined profiles which are unique to each individual user, containing information about words learnt and general progress. Data collected will not be personal to the user, in accordance to data privacy laws.

This criterion is partially met. The user profiles do store data about the user, including the progress into each level and the last login, with no private data (location, gender) being collected. The usernames are unique to each user, being a primary key field. However, the data collected on game progress is limited to just the amount of words learned. The database doesn’t store information about skipped words or words that required several attempts. This could be addressed in further development by creating a separate table, linked with a database relationship, which specifically contains data about the progress into each level. Doing it this way could allow for more useful information to be gathered.

1. The solution will utilize the Microsoft Speech API Ver 5.4 to provide a text-to-speech output which provides legible instructions.

This criterion is fully met. The solution uses the in-built Microsoft Speech API to provide a text-to-speech output. User testing has confirmed that the voice outputted is clear and legible.

1. The solution will be able to handle input errors without risking the program’s stability. These errors will be made apparent to the user and may be corrected by the user with ease. The program’s stability won't be compromised by boundary-value inputs.



This criterion is partially met. As mentioned in testing, this program is able to withstand most boundary or abnormal inputs, through the use of continued validation and verification of data. The above example shows how validation is used in the add user form to conform to the validation rules set in the user database. By incorporating validation, the program is able to identify input errors and inform the user of their mistake. Entering in an incorrect database password, for example, produces a prompt informing the user that an incorrect password has been entered. Without such error trapping, the program may be caused to shut down or the user won’t be informed of their mistake, causing confusion. That being said, some input errors still persist within the program that haven’t been covered by testing and development. This could be addressed in further development by continuing to identify errors and writing code to trap the error.

1. The solution will be developed and coded so that later maintenance may be completed with ease. In this way, additions to the program’s functionality are feasible and the solution has the foundations for future-proofing.

This criterion is partially met. The code is written into clear, defined sub-procedures which other programmers will be able to read and understand. Variables and UI elements are labelled clearly using Hungarian notation, a standard which most programmers use. Finally, the code is formatted and annotated to read clearly, and inform programmers when complex or unconventional code is used. With these, programmers can easily perform maintenance to the program and perform additions in future development. However, more work can be done to split up larger procedures into smaller, more understandable sub-procedures. This would also contain errors for easier debugging. Improvements could also be made to provide more thorough documentation where annotation is non-existent or unclear for those unfamiliar with the programming language.

1. The solution will meet the predefined budget of £20 total for hardware, testing or other external costs.

This criterion is fully met. No external costs were required to develop or test the solution.

1. The solution will be fully developed, tested and evaluated by 12th February 2019.

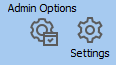
This criterion is fully met. The solution was user tested by 9th February and evaluation was completed soon after. All deadlines were met.

1. The main client/stakeholder will be satisfied with the provided solution and will continue to implement the solution into their work.

This criterion is fully met. The user-testing proved successful, prompting the stakeholder to implement the solution into their work. They were pleased with how the program looked and performed, meeting all the required criteria they set out. The current version of the program will now be set up on a computer at the stakeholder’s residence, to be used by young children on a daily basis to improve their spelling. Maintenance will be provided into the future if the solution encounters any problems.

# Describing the Final Product

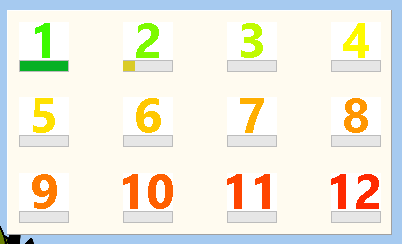
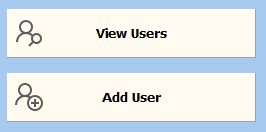
**3.4.3 Describe the final product**

**Icon buttons**

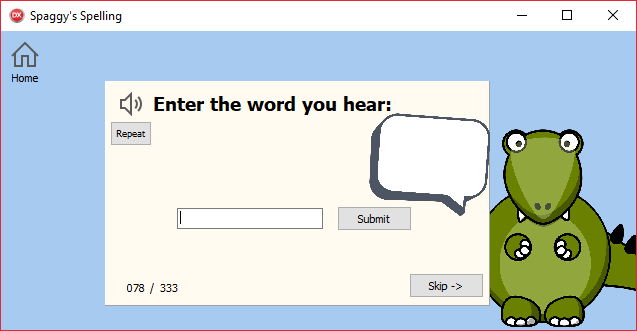
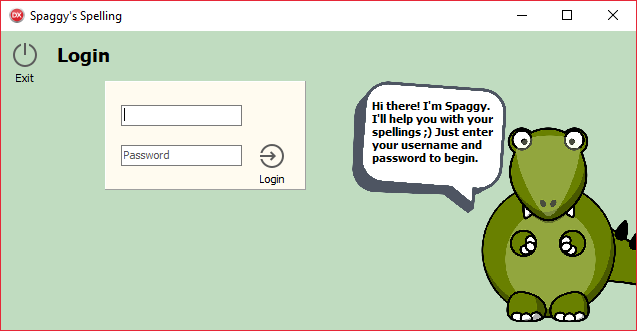


Icon buttons were used in the program as a means of navigating through the different forms. This was a deliberate choice over standard buttons as it would make the UI more unique in style and more interesting to look at. The inclusion of a custom icon package also allowed the UI design to be more consistent. In practice, this usability feature was s success in that they looked more aesthetically pleasing than the standard grey buttons, especially with the variation in the icons themselves. The UI design also became more consistent, and allowed for those who can’t read the button captions (young target audience). However, these icons didn’t do much to improve the overall look of the program. Further development would see a change in icons from straight lines to brightly coloured squiggly lines, which are more favoured by a young target audience.

**Panels**

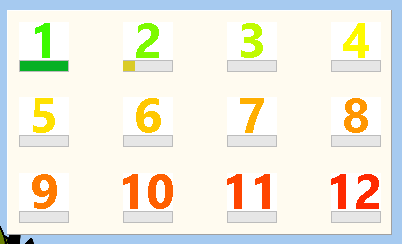


Panels were used in the program to group similar objects together and make the overall design easier to interpret and use. In practice, these panels were a success. They managed to clean up the UI and group similar items, such as the game levels, which made the entire UI easier to comprehend. They also provided a background for greater contrast to labels, captions etc, making them easier to read. Finally, the panels produced a two-colour design scheme that was aesthetically pleasing.

**Colours**

As previously mentioned, a two-colour design scheme was used in the program to make it more aesthetically pleasing. Sky blue, Money green and Cream were the colours used. Overall, the choice of colours was a partial success. While the reduced three-colour template made the UI appear brighter and nicer to look at, it left a lot to be desired. Many of the testers in user testing commented that it looked too simple, and they ‘ran out of things to look at’.

**Progress bars**



Progress bars were used to show the user’s progress at a glance within the main menu. They were successful in meeting this purpose, as they are easy to understand and stick out. Though they lack the accuracy and detail of numbers, this progress can be found within the game level anyway.

**Audio Output/Speech API**

The use of an audio output, through the speech API, had the purpose of making the program more usable in providing orthographic assistance. It allowed users to hear the word they were required to spell, and tested them on their spelling. In the end, this feature was a success, as user testing found this feature to be exciting and engaging for the young audience, and allowed them to be tested on their spellings as they would at school. This feature means that users are able to use this program by themselves, which follows the most useful way to learn spellings: regularly.

# Maintenance and Development

**3.4.4 Maintenance and development**

**(a) Discuss the maintainability of the solution and any limitations of the system.**

Maintainability is a measure of ease and speed in restoring a program to operational status or upgrading an outdated feature. In regards to making repairs, this solution is built upon the latest version of object pascal, a programming language with widescale support and continued updates. If a failure were to happen to the solution, it would be easy to gain support online. This support is likely to be upheld for years to come, so the maintainability of this solution is strong. The Delphi IDE also has several debugging features, such as a log output, that would help to reduce debugging time.

Furthermore, the code to the solution is clearly laid out for other programmers to interpret and make changes to. Detailed annotations describe functions whose purpose are not abundantly clear, and procedures separate long sections of code into smaller, more manageable chunks. In regards to how the code is written, it would be easy for a programmer other than myself to make repairs or upgrades. Without annotations, other programmers would have to spend more time trying to understand sections of code, which would be an issue for stakeholders of the solution.

Splitting the code into smaller procedures also has the benefit of reducing errors to smaller sections. When an error occurs, the section of code in which it is acting will be smaller, allowing for less debugging and therefore time spent.

That being said, the system is limited to the language it is written in. Object Pascal may eventually loose support, at which time repairs made to the solution will become more difficult or maybe even impossible. As well as this, errors may occur to the system that cannot be identified in a debugger, such as corrupt program files. These types of repairs are also difficult. Finally, the system is limited in that other programmers may simply not understand the code that is written. Despite the solution containing detailed annotation, parts of the solution may require additional documentation to be fully understood by programmers not so familiar in the language.

**(b) Describe how the program could be developed to deal with limitations of potential improvements/changes.**

Improvements can be made to the system by

**(c) Discuss potential further development of the solution.**

Further development may be carried out to the solution to provide more features to the user. One such feature could be to include a lexicon that is unique to the prior knowledge of each user. This could be achieved by running several tests on the types of words the user is already familiar with and eliminating them from the lexicon. This would provide a more tailored learning experience, instead of using the same levels for every user.

Another feature could be to include more configurable settings, such as accessibility controls. This would allow children who suffer from colour blindness, for example, to use the system more effectively, by changing the colours used in the program. This increases the target audience of the solution, generating more money for the stakeholders. Further changes could be made to the speech API to customize audio outputs and in-sync volume control.

Finally, further development could be made to the general set of words that are tested. Instead of using local flat-file word banks, the solution could connect to a much larger collection of words online. This would allow the program to have more game levels and keep the audience engaged for much longer. The solution could maybe include user submitted levels, like those that exist in Memrise. Any of these steps could make sure the user doesn’t run out of words to be tested on.