**Analysis and Design**

**Introduction**

Service request number one proposed game enhancements for version 2.0 of the popular Hangman game distributed by Simple Games Collective. The game itself was very rudimental and simple text based. A user would want much more engagement and visual queues so a complete overhaul of the program will have to completed.

A discussion was had with the company stakeholders after reviewing the service request form and they had no feedback, clearing it to proceed to the next phase.

Further study of the form myself, an improvement was identified. An adjustment will be made to the order in which the upgrades will be completed. The updates will now occur in the following order:

Version 1.1 – Welcome Screen Enhancement

Version 1.2 – Word List Selection Enhancement

Version 1.3 – Difficulty Selection Enhancement

Version 1.4 – Gameplay Enhancement

Version 1.5 – Game Over Screen Enhancement

Version 1.6 – Scoring Enhancement

The change, being the switching of the version 1.5 and 1.6 updates, will allow for a better flow of the game. In total, six areas were designated for improvements and the following analysis will examine problems and solutions for each, and how they can be implemented through design changes to update the game.

**Version 1.1**

**Welcome Screen Enhancement**

The initial pre-game program lacked both information and engagement, only asking the user for their name before starting the game, and this field could be left blank. One of the projects specified requirements was to turn the ‘welcome user’ statement into a function for the name of the user and provide playing instructions.

This will be done through creating a designated game area where information can be displayed then input options asked. A scoreboard will be shown as a way of introducing the game. It will create a connection to the game itself as returning plays may see their name and previous scores written, and new plays see what they could achieve.

The name of the user must be designated to a variable and validated. As the entire game will be formatted within a game area, the name will have to be a certain number of characters to avoid enlarging this area. It will also be unable to have spaces, but within these restrictions it is believed all users should be able to think of something to represent their name.

Basic rules will be shown to allow new players to grasp the basics of the game. The game itself will come with a user’s manual for detailed instructions if needed. As this is a basic game, game flow and basic rules could be self-determined with a couple of playthroughs of the game itself. A dummy input will be used to ensure the user has had sufficient time to read the rules, or to just skip the page if they have been read before.

The following pages will look at how these changes will be implemented into the game to complete the version 1.1 upgrade.

**High Level Function**

**Name: welcome**

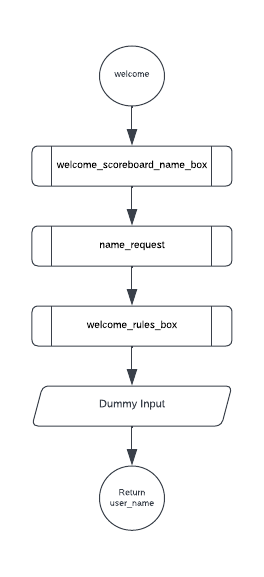
Parameters:  height, width

Return: name\_user

The entire welcome enhancement will be contained within a function that can be called from the main file. It will require arguments that will set the size for display boxes and return the name of the user, which will be used to add a more personalised feel throughout the game.

The welcome function itself will consist of three functions. Two will be displays, showing information to the user with the first containing a scoreboard and name request and the second containing the rules. The other will validate the name entered to ensure this field is not left blank or is too long and will end up causing the display box to expand.

Refer to figure 1 for the flowchart of the function.



Figure

**Middle Level Functions**

**Name: welcome\_scoreboard\_name\_box**

Parameters: height, width

Return:

A function will be created that will clear the screen before displaying a header, a general welcome message, and a scoreboard. It will then ask the user for their name, but there will be no input functionality within this function, instead that will come from the name validation function itself.

Refer to figure 2 for a diagram of what the display box will look like and the function that will be used to form it.

**Name: name\_request**

Parameters: height, width

Return: user\_name

A function will be created to allow the user to input their name. If the name field is left blank, the welcome\_name\_scoreboard\_box function will be called to clear the screen and display the welcome screen again. This time though, the second function will display an error message before asking the user again to input their name. If the name field has too many characters the same process will occur, but the error message will contain additional information about the maximum number of allowed characters. Once a valid name has been entered, the loop will break and return the name of the user.

Refer to figure 3 for a flowchart of the function.

**Name: welcome\_rules\_box**

Parameters: height, width, user\_name

Return: A function will be created to display the rules after the user has entered a valid name. It will clear the screen then display the created header, welcome the user using the name entered and show the rules of the game. As the rules will only need to be displayed once, a separate function will not be created. A text file containing the rules will be open within this function and each line printed.

Refer to figure 4 for a diagram of what the display box will look like and the function that will be used to form it.

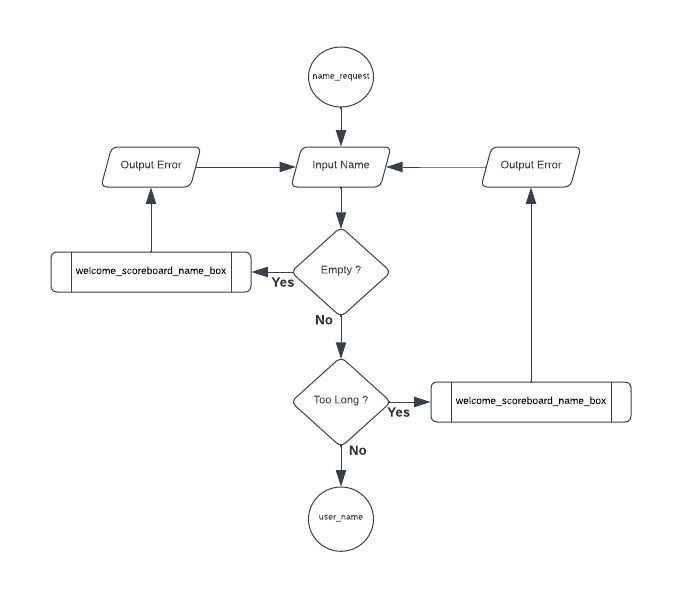
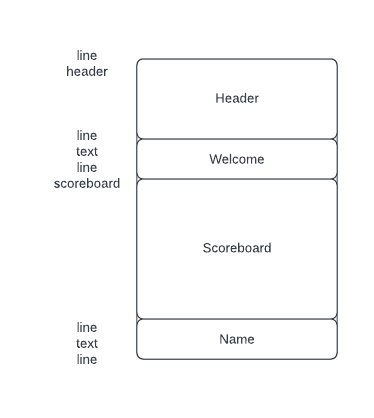
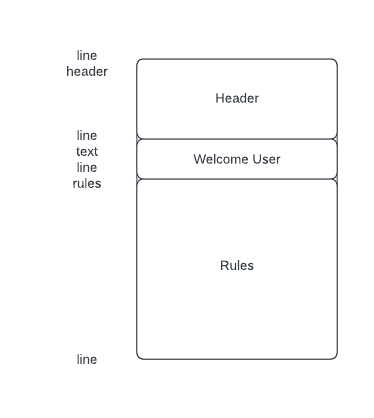


Figure 2

Figure 4

Figure 3

**Low Level Functions**

**Name: solid\_line**

Parameters: width, character, end\_character

Return: 1

A function will be created to display a single solid horizontal line across the screen with different end or cap characters. It will be used to create the top and bottom of each display box, as well as any line separators needed within. This will be stored in the utility\_functions file and imported into each other file, as it will be needed in every display. As it will only ever print a single line, it will return one.

Refer to figure 5 for an approximate image of the function will print.

**Name: header**

Parameters: inner\_width, game\_name, company, version

Return: line\_count

A function will be created to show the name of the game, the company and current version of the game within inbuilt borders. This will be shown at the top of every display. It will be stored in the utility\_functions file and imported into each other file, as it will be needed in every display. The name of the game itself, in this case Hangman, will be written in text art and stored in a list in the text\_art file. This function will print each line centrally within the display box. As this will be a generic header function with the ability to be used for other games, it will return the number of lines the name of the game occupies, plus one line for the company and program version respectively.

Refer to figure 6 for an approximate image of the function will print.

**Name: text\_line**

Parameters: inner\_width, text, position

Return: line\_count

A function will be created to display any text that needs to be shown within the display box. It will have the borders inbuilt and the ability to print either a list of texts, or a single line of text on the left, right or central. It will be stored in the utility\_functions file and imported into other files, as it will be needed by most displays. It will return the number of lines printed.

Refer to figure 7 for an approximate image of the function will print.

**Name: empty line**

Parameters: inner\_width

Return: 1

A function will be created to add an empty line within a display box. This will be used to create space within the box and simple be an empty line with inbuild borders. It will be stored in the utility\_functions file and imported into other files, as it will be needed by most displays. As it will only ever print a single line, it will return one.

Refer to figure 8 for an approximate image of the function will print.

**Name: scoreboard**

Parameters: inner\_width

Return: line\_count

As the scoreboard will be required twice, a function will be created to show it. The scores themselves will be stored as a text file so this function will open the file, use the information within to create lists, close the file and print the information. It will be stored in the utility file along with other common functions and imported when needed. It will return the number of lines printed.

Refer to figure 9 for an approximate image of the function will print.

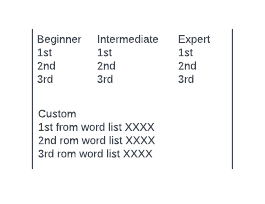


Figure 9

Figure 8

Figure 7

Figure 6

Figure 5

**Version 1.2**

**Word List Selection Enhancement**

The game currently contains a preset word for you to guess, which makes playing it again very unappealing once you know the word. One of the projects specified requirements was to replace the set word with code that allows the user to select an input file by name if it contains at least one word. This file of words must then be turned into a list before a single word is randomly selected for this instance of the game. This could be taken further by creating a menu structure, allowing users to select their choice by simply entering a number. This would greatly simplify the experience of the user, as they are no longer required to type in long file names.

A solution to this problem is to create lists of words. Three have been created that that have varying levels of difficulty to allow users different experiences.  They are:

For beginner level, a list of the most common 500 words.

For intermediate level, a list of the most common 5000 words.

For expert level, a list containing all English word.

The user will also be able to select from any list of words they upload into the word\_lists file. These custom lists will not be checked against the English words list as this will allow a greater diversity base for users. A user with different language backgrounds will be able to upload a file with word from their own language to play with. At this stage though, the language will have to use the same letters as English as other keyboards and letter are yet to be implemented in the game design. Customs lists will also have the restriction of containing at least one word.

These lists will be filtered so that only words with 4 or more letters can be randomly chosen to create a better game experience. The lists will still contain the smaller words, in case they are needed if a future update, and filtered in game.

The following pages will look at how these changes will be implemented into the game to complete the version 1.2 upgrade.

**High Level Function**

**Name: word\_selection**

Parameters: height, width, user\_name

Return: word, from\_file

The entire word selection enhancement will be contained within a function that can be called from the main file. It will require arguments that will set the size for display boxes and the name of the user to add personalisation. It will return the word for the game and the file which it was taken from.

The word\_selection function itself will consist of four functions. Two will be displays, showing a menu of premade word lists or custom word lists to select from. The other two will help validate the selection input and select a word at random from the selected list.

Refer to figure 10 for a flowchart of the function.

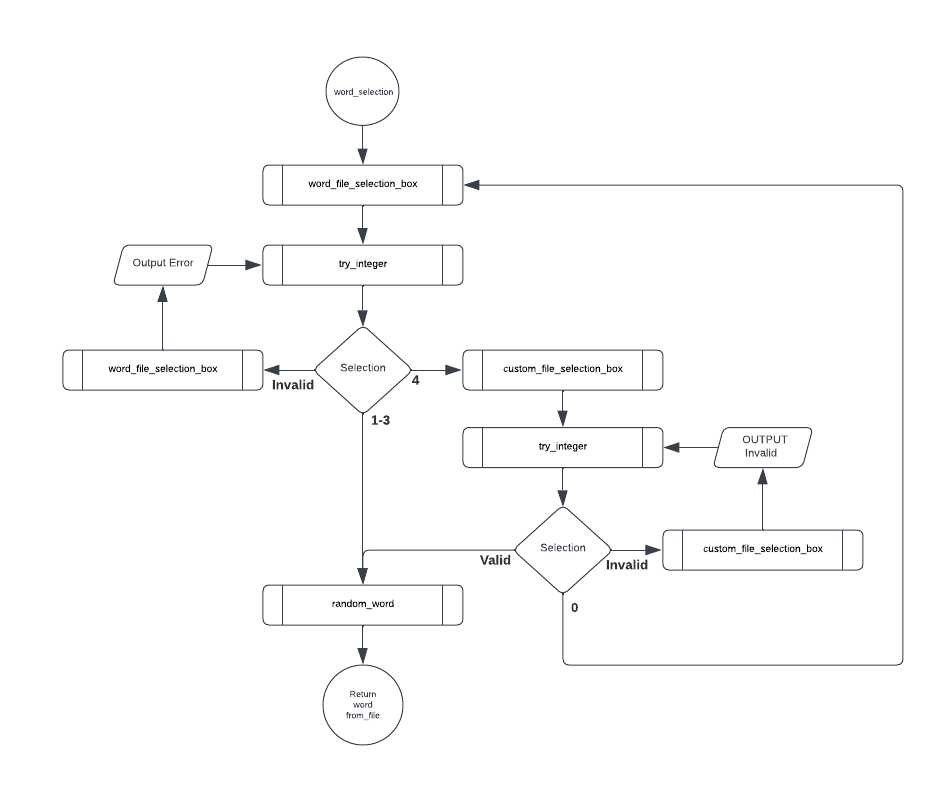


Figure 10

**Middle Level Functions**

**Name: word\_file\_selection\_box**

Parameters: height, width, user\_name

Return:

A function will be created to show the three premade lists, as well as give the user the option to select from any custom list that have been uploaded. A display box will be created after clearing the screen with the header at the top, followed by instructions and the names with descriptions of the choices. These choices will be stored in a list at the top of the file. There will be no input functionality within this function.

Refer to figure 11 for a diagram of what the display box will look like and the function that will be used to form it.

**Name: custom\_file\_selection\_box**

Parameters: height, width, user\_name

Return: custom\_list\_dir

A function will be created to show the custom lists that have been uploaded to the word\_lists folder. A display box will be created after clearing the screen with the header at the top, followed by instructions and numbers followed by the names of all the custom uploaded files. It will use the os.listdir function to get a list of all the files within the folder, then print them next to their respective numbered place. There will be an option to return to word\_file\_selection\_box if a custom list is no longer desired. There will be no input functionality within this function. It will return the directory of custom lists.

Refer to figure 12 for a diagram of what the display box will look like and the function that will be used to form it.

**Name: try\_integer**

Parameters:  question

Return: user\_input

A function will be created to attempt to change requested integer inputs from strings into integers. It will use the try and except method to return either the integer or a sting. The main function deals with the returned value as invalid results will call other functions. It was decided to keep this function to the basics of attempting to change the data type only.

Refer to figure 13 for a flowchart of the function.

**Name: random\_word**

Parameters: file

Return: word.upper()

A function will be created to select a random word from the selected word file. It will open the selected file and create a list of all the words that are 4 characters or longer before closing the file. A random word will then be chosen are returned.

Refer to figure 14 for a flowchart of the function.

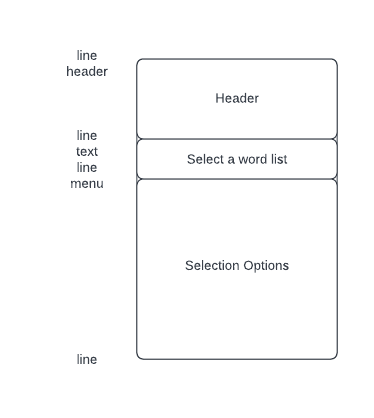
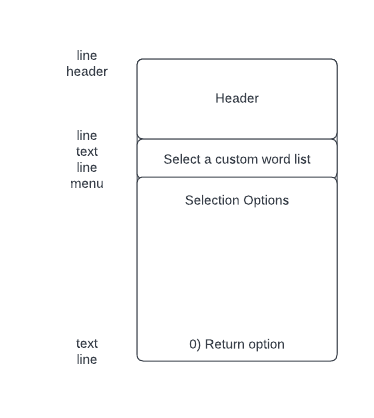
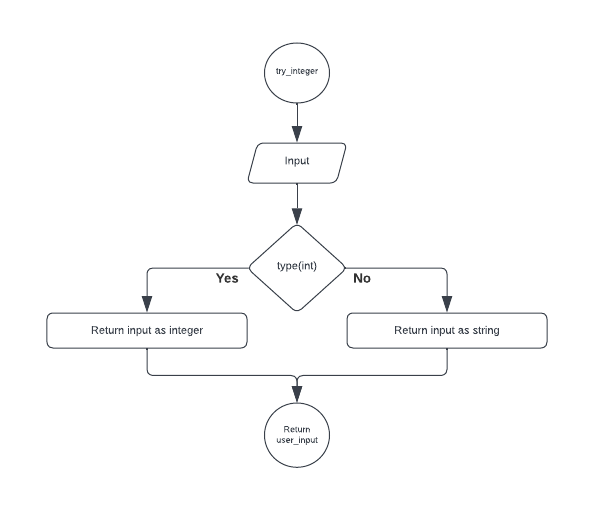
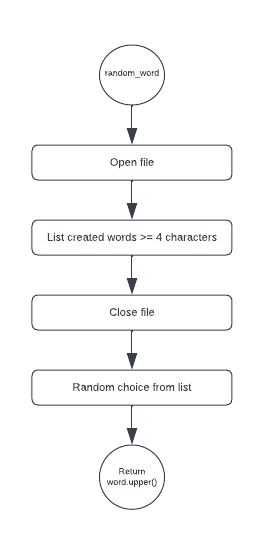
­­­­

Figure 14

Figure 13

Figure 12

Figure 11

**Low Level Function**

**Name: menu**

Parameters: inner\_width, options, start\_number, lines\_per\_option

Return: line\_count

A function will be created that can show a menu within the display box. It will be used to show numbered options for the user to select from. It will be displayed between inbuild borders and take a list of options. It will print the numbered choices, with the ability to print multiple lines per choice if required. It will be stored in the utility\_functions file and imported into each other file as needed and will return the number of lines it printed.

Refer to figure 15 for a flowchart of the function.

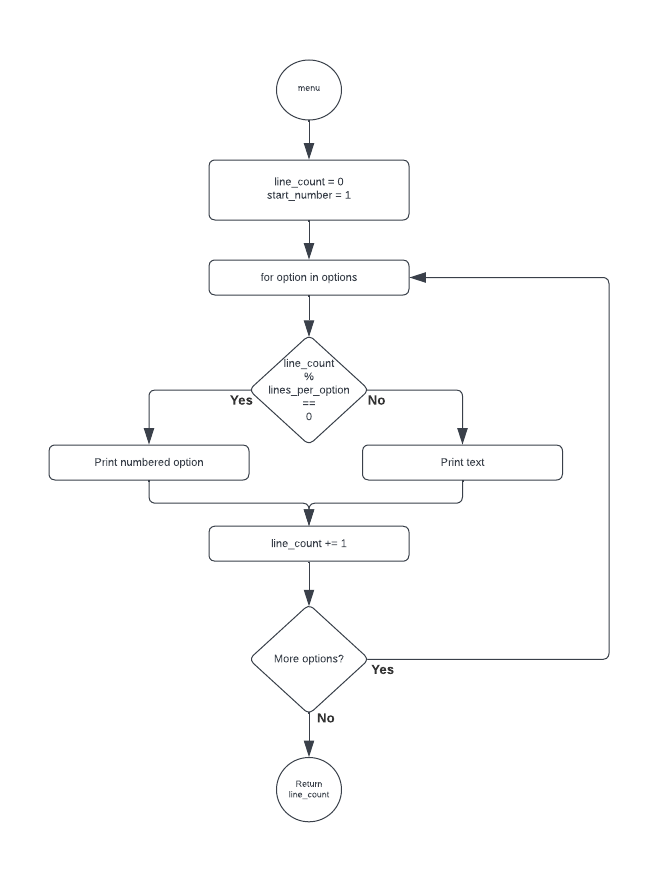


Figure 15

**Version 1.3**

**Difficulty Selection Enhancement**

One of the projects specified requirements is the ability to vary the difficulty of the game to cater for different user ability. A brief search was done on the history of Hangman and concluded that 10 lives are too many, as the common game of Hangman usually consists of only six lives. Having 10 lives makes the game unappealing as it provides no challenge to even below average user.

The following levels of difficulty shall now be implemented.

For easy difficulty, the user will have seven lives.

For normal difficulty, the user will have six lives.

For hard difficulty, the user will have five lives.

The addition of difficulty levels will add another level of fun and challenge to the game. With six lives the user will have to make a calculated decision on their next letter guess instead of a random input. This, in addition to the different word lists, will create a variety of difficulty levels that will cater to everyone from a child to a master wordsmith.

The following pages will show how this enhancement will be implemented into the game to complete the version 1.3 upgrade.

**High Level Function**

**Name: difficulty\_selection**

Parameters: height, width, name\_user

Return: lives, difficulty

The entire difficulty selection enhancement will be contained within a function that can be called from the main file. It will require arguments that will set the size for display boxes and the name of the user to add personalisation. It will return the number of lives the user will have in the game and the difficulty selected.

The difficulty\_selection function itself will consist of two functions. One will display the menu showing the difficulty levels the user can choose from. The other will be the try\_integer function, which will be used to help validate the selection input.

Refer to figure 16 for a flowchart of the function.

Diagram

Description automatically generated

Figure 16

**Middle Level Function**

Name: **difficulty\_selection\_box**

Parameters: height, width, name\_user

Return:

A function will be created to show the three different difficulty level through both text art and words. As each difficulty level will have their own avatar, images of them will be displayed now as part of the function. A menu will then be shown for the user to choose from. These choices will be stored as a list of strings. It will require arguments that will set the size for display box and the name of the user to add personalisation There will be no input functionality within this function.

Refer to figure 17 for a diagram of what the display box will look like and the function that will be used to form it.

Diagram

Description automatically generated

Figure 17

**Version 1.4**

**Gameplay Enhancement**

Although the gameplay itself worked, it lacked any engagement and was not visually pleasing. We want this to look and feel like a game, so all displays will continue to appear in a game box.

Custom comments will be displayed under the header, representing how the user is progressing. At first it will tell them that the time has start. From there, depending on if the guess was correct, incorrect, or invalid, multiple unique phrases will be used.

The game will then be enhanced by adding a better visual representation of the users remaining lives, as a user may remember from playing of a piece of paper. This will provide a better user experience, create a nostalgia feeling and possibly provide extra pressure to get their next guess correct.

The progress of the word will be enhanced, changing the appearance from ‘-’ to ‘\_’ and creating its own separate area within the game box to be displayed.

The options to choose from will also be spread out to be more visually pleasing and displayed in the same way as a keyboard.

The following pages will show how this enhancement will be implemented into the game to complete the version 1.4 upgrade.

**High Level Function**

**Name: gameplay**

Parameters: height, width, user\_name, word, difficulty, lives

Return: result, time, progress

The entire gameplay enhancement will be contained within a function that can be called from the main file. It will require arguments that will set the size for display boxes and the users name, the word, difficulty, and lives selected. The function will be timed, which will be used for scoring. The result will either be True if the game was won, False if the game was lost, or None if the user chose to quit before the game was finished. It will return the time in seconds, the result and progress on the word.

Refer to figure 18 for a flowchart of the function.

Diagram

Description automatically generated

Figure 18

**Middle Level Function**

**Name: gameplay\_box**

Parameters: height, width, user\_name, comment, picture, progress, lives, keyboard

Return: available

A function will be created to show the playing of the game. It will consist of the header, a message, a visual of the lives remaining, text of the lives remaining, the string with ‘ \_ ’ for remaining letters and a Qwerty keyboard visualisation of the remaining letters. An option will also be shown for if the user wishes to quit the game.

Refer to figure 19 for a diagram of what the display box will look like and the function that will be used to form it.

Diagram

Description automatically generated

Figure 19

**Name: validate\_character**

Parameters: height, width, user\_name, picture, progress, lives, keyboard, guesses

Return: guess

The askUserForSingleharacter function will be renamed to validate\_character to conform with PEP 8 conventions and will then be modified in the following ways.

As the gameplay\_box function will show the available letters, this section of code will be removed.

A more sophisticated error message will be shown, detailing why the input was incorrect. This will include too many characters, invalid character and letter already used.

Refer to figure 20 for a flowchart of the function.

Diagram

Description automatically generated

Figure 20

**Version 1.5**

**Game Over Screen Enhancement**

The original final message for the game was telling the player ‘You Won’ or ‘You Lose’. It completes a basic ending by giving the user a result before making them quit the game. One of the projects specified requirements was to create a more sophisticated game over message and give scoring details. As a more complicated scoring system will be implemented in version 1.6, the second half of the requirement will be fulfilled there. This enhancement will concentrate on the actual game over screen.

The greatest improvement here is telling the user what word they were trying to guess if they were unable to guess it. Not knowing can create frustration and annoyance in the user base and will reduce overall play volume.

To give users a more personalized experience, their name will be used in the game over screen for winning losing or quitting the game. If the player decided to quit instead of completing the game, the screen will be very similar to and be treated the same as a loss. In addition, different messages will be shown at the completion of game, depending on the result.

The following pages will show how this enhancement will be implemented into the game to complete the version 1.5 upgrade.

**High Level Function**

**Name: game\_over**

Parameters: height, width, user\_name, word, difficulty, result

Return:

The entire game over enhancement will be contained within a function that can be called from the main file. It will require arguments that will set the size for display boxes as well as the name of the user, the word the player was trying to solve, the difficulty so the appropriate hangman image can be used and the result of the game.  A dummy input used to continue.

Refer to figure 21 for a flowchart of the function.

Diagram

Description automatically generated

Figure 21

**Middle Level Function**

**Name: game\_over\_box**

Parameters: height, width, user\_name, picture, word, finish

Return:

A function will be created to show a display box for the conclusion of the game.  It will consist of a header, an overall message, an image, the word, and some final text about how the user went.

Refer to figure 22 for a diagram of what the display box will look like and the function that will be used to form it.

Diagram

Description automatically generated

Figure 22

**Version 1.6**

**Scoring Enhancement**

The current scoring system simply gave the user 10 points for every letter they guessed correctly. This is very basic and unappealing as the number of individual characters the word has is directly correlated to the maximum score the user could achieve. A complete overhaul of the scoring system will be undertaken with the time taken to solve the puzzle representing the best was to achieve a high score.

**Letter values**

Each individual letter will be given a value the same as they are worth in the board game scrabble, making uncommon letter score more if they are found within the word.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| 1 | 3 | 3 | 2 | 1 | 4 | 2 | 4 | 1 | 8 | 5 | 1 | 3 | 1 | 1 | 3 | 10 | 1 | 1 | 1 | 1 | 4 | 4 | 8 | 4 | 10 |

**Word list**

As word list contain more difficult words, your score from the letters will be multiplied by one of the following values.

For the beginner level list, a 0.75 multiplier will apply.

For the intermediate level list, a 1 multiplier will apply.

For any custom lists, 1 multiplier will apply.

For the expert level list, a 1.25 multiplier will apply.

**Difficulty**

As the number of lives vary, your score from the letters will be multiplied by one of the following values.

For the easy difficulty, a 0.75 multiplier will apply.

For the normal difficulty, a 1 multiplier will apply.

For the hard difficulty, a 1.25 multiplier will apply.

Once both multipliers are considered, a high variety of difficulties and scores can be achieved.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Beginner (.75) | Intermediate / Custom (1) | Expert (1.25) |
| Easy (.75) | .56 | .75 | .94 |
| Normal (1) | .75 | 1 | 1.2 |
| Hard (1.25) | .94 | 1.2 | 1.56 |

**Time**

Time will be scored as taking 120 seconds as the time a user should take to complete the game, although this may be on the generous side. The time multiplier will increase exponentially in both directions using the following formula.

Time Multiplier = 1 / (Time / 120)

This will give the following score multipliers between 20 and 180 seconds.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Time | 20 | 40 | 60 | 80 | 100 | 120 | 140 | 160 | 180 |
| Multi | 6 | 3 | 2 | 1.5 | 1.2 | 1 | 0.86 | 0.75 | 0.67 |

**High Level Function**

**Name: scoring**

Parameters: height, width, user\_name, word, from\_file, difficulty, result, time\_taken

Return:

The entire scoring enhancement will be contained within a function that can be called from the main file. It will require arguments that will set the size for display boxes and calculate the score. The score will be calculated and the leader\_board file opened and read. If the score is higher, it will replace the beaten score and the leader\_board file written to reflect this. A function will then be used to display a screen like that at the very beginning with the score board and a message. The message will show how the current games score was calculated and inform he user if they have beaten a score.

Refer to figure 23 for a flowchart of the function.

Diagram

Description automatically generated

Figure 23

**Middle Level Function**

**Name: final\_scoreboard\_box**

Parameters: height, width, word, from\_file, difficulty, time\_taken, placing, word\_score, list\_multi, diff\_multi, time\_multi, score

Return:

A function will be created to show a display box for the scoring of the game after clearing the screen.  It will consist of a header, the scoreboard, score calculations, and some final text about how the user scored.

Refer to figure 24 for a diagram of what the display box will look like and the function that will be used to form it.

**Name: calculate\_score**

Parameters: word, from\_file, difficulty, time\_taken

Return:  word\_score, list\_score, difficulty\_score, time\_score, score

A function will be created to calculate all the individual score / multipliers and a total score. It will take in all the original values and return the results.

Refer to figure 25 for a flowchart of the function.

**Name: write\_high\_score**

Parameters: user\_name, from\_file, score

Return: place

A function will be created to check if the score from the game, if won, is higher than any currently recorded. If so, it will put the score in its correct position and write the text file to reflect this.

Refer to figure 26 for a flowchart of the function.

Diagram

Description automatically generated

Figure 26

Diagram

Description automatically generatedDiagram

Description automatically generated

Figure 25

Figure 24

**Conclusion**

The above analysis has found solutions to the specified project requirements and design changes will create an engaging and playable game once completed.

Although this will bring the Hangman version 2.0 to an acceptable level, many more possible changes were identified during the analysis stage. They include but are not limited to:

* Allowing two users to play, one setting the word and the other guessing.
* More customized responses
* Databasing each game for future data analysis
* Introduction of a user profile
* Introduction of achievements
* Multiple language support

These could be looked at for future upgrades, if desired, to version 3.0 and beyond

A full test plan for the upgrade to version 2.0 is outlined in document 2, Test Plan.

It is recommended that this project move to the development stage.