Passphrase Generator - SWDP

Problem Definition

Known Facts	User Requirements	Necessary Processing	Alternative Solutions
• Passphrases consist of a series of spaced words chosen from the Diceware Word Lists • These words are chosen by random by performing 5 dice rolls for each word • 5 dice rolls means 5 digits, which constitutes a key associated with each word in the word lists. The program generates these keys, looks them up, and combines the resulting words to form a secure passphrase • Like an ordinary die, all digits are between 1 and 6	• Type 1 for Get Started, 0 for Exit • Answer the resulting questions by typing the appropriate numbers shown on-screen. Most notably, provide how many words you would like in your passphrase	Seed pseudorandom number generator. Introduce and familiarize user with program's operation. Ask user if they are ready to begin. While answer is not 0 or 1, throw error and ask again. If answer is 0, then exit program. Ask user for number of words in passphrase. While answer is not between 6 and 10, throw error and ask again. Load Diceware word list into map. For number of words in passphrase: • Generate random 5-digit code • Look up word associated with code in map and	Seed pseudorandom number generator. Introduce and familiarize user with program's operation. Ask user if they are ready to begin. While answer is not 0 or 1, throw error and ask again. If answer is 0, then exit program. Ask user for number of words in passphrase. While answer is not between 6 and 10, throw error and ask again. For number of words in passphrase: • Generate random 5-digit code • Read from text file line by line until row with code is found • Extract corresponding word and

• The largest words are 6 characters long • Word lists come in a multitude of languages, ranging from English to Latin • Word lists exist as text files and must be read by the program • Users can specify how many words they would like in their passphrase • The program's interface must be clear and explain its operation		append to passphrase • If current iteration is not equal to final iteration, add spacing to passphrase • Clear random code Display passphrase to user. Display success message and bid user farewell. End of program.	append to passphrase • Close text file • If current iteration is not equal to final iteration, add spacing to passphrase • Clear random code Display passphrase to user. Display success message and bid user farewell. End of program.

Analysis (IPO Chart)

Input Data	Processing Steps	Output Data
• A 1 or 0 depending on whether the user is ready to begin • A number between 6 and 10 representing how many words the	Seed pseudorandom number generator. Introduce and familiarize user with program's operation. Ask user if they are ready to	• A message welcoming the user to Passphoria • A series of messages explaining how Passphoria works

passphrase should be	begin. While answer is not 0 or 1, throw error and ask again. If answer is 0, then exit program. Ask user for number of words in passphrase. While answer is not between 6 and 10, throw error and ask again. Load Diceware word list into map. For number of words in passphrase: ● Generate random 5-digit code ● Look up word associated with code in map and append to passphrase ● If current iteration is not equal to final iteration, add spacing to passphrase ● Clear random code Display passphrase to user. Display success message and bid user farewell. End of program.	● A message asking the user if they are ready to begin, listing possible responses 1 for Get Started and 0 for Exit ● An error message for invalid inputs ● A message asking the user how many words their passphrase should be ● A message revealing the user's passphrase ● A success message thanking the user

Design Using Pseudocode

Pseudocode was written in a separate text file to preserve formatting.

Testing/Verification

Test Case 1: Ensure data validation works

To test this, I entered invalid data for each of the questions asked by the program. As expected, I was met with a red error message each time saying, “ERROR: Invalid entry. Please try again.” No matter how many times I entered invalid data, I received the same error message. Thus, I can confirm that Passphoria’s data validation works properly.

Test Case 2: Ensure background music plays

This is a relatively straightforward test case. I made sure my Windows audio was on, ran the program, and sure enough, the predetermined soundtrack began playing. I also made sure the soundtrack looped by keeping the program open for an extended period of time.

Test Case 3: Ensure exit functionality works

It is important that users can opt out of the program if they so choose. To test this, I entered 0 for the “Ready to Begin?” question, after which the program immediately stopped. Thus, I can confirm that the exit functionality works correctly.

Test Case 4: Ensure user can customize their text colour

Since there are 3 text colours to choose from, I ran the program 3 times, choosing a different colour on each run. As expected, when I chose cyan, all subsequent inputs were

coloured cyan; when I chose magenta, all subsequent inputs were coloured magenta; when I chose default, all subsequent inputs were coloured light grey.

Test Case 5: Ensure music setting works

Some people find music distracting; others find it enjoyable. Thus, users should have the freedom to enable/disable Passphoria's background music. To test this, I entered 1 for the "Do you want to continue listening to music?" question. As expected, the music continued playing. Likewise, when I entered 0, the music stopped. This confirms that the music setting functions as intended.

Test Case 6: Ensure user is referred to by name

Being called by your name enhances personalization and, thus, user experience. To verify that users are referred to by name, I entered my name in response to the appropriate question and saw that, in subsequent outputs when the program intended to mention my name, the name was always accurate.

Test Case 7: Ensure passphrase is generated properly (i.e., according to user inputs)

To test this, I performed several trials of the program, customizing the passphrase differently each time. In each trial, the passphrase generated adhered to my user

specifications, including number of words, number of spaces separating each word, capitalization, languages, and special characters.

Test Case 8: Ensure passphrase strength indicator is accurate

I ran the program multiple times until I landed on a passphrase 40+ characters long, a passphrase 20–39 characters long, and a passphrase under 20 characters long. The results were as expected, with the first length rendering an excellent score, the second length rendering a strong score, and the last length rendering a weak score.