**PROJECT 2**

**HANGMAN**

The year is 2047. Paper has been outlawed, as have whiteboards, and the concept of fun entirely. Pencils and markers are now contraband. The concept of a ‘game’ has become a distant memory, as people spend their evenings and days mindlessly scrolling the infinite feed of the one mega-conglomerate that has acquired all the social media apps. But you, you’re different. By approaching this problem using your own tech skills, we might be able to find a way to have some real fun, combining a classic game with the technology of the future.

We’re going to create a digital version of the game ‘hangman’. It’s a classic, so here are the rules if you somehow aren’t familiar with them:

1. A word is selected, and you draw blanks that represent each letter of that word.
2. You attempt to guess the word letter by letter, until you have an idea of what the word is, and then you may guess the word.
3. For every letter you guess that isn’t in the word, or every incorrect word you guess, an extra body part is added to the hung man. Pretty morbid for a kids’ game, huh?
4. When The head, body, arms, and legs have all been added (and each facial feature as well if you’re playing that way) the game is lost
5. The game is won if you guess the word before the full hangman is drawn.

**STRUCTURE**

The program will read in a list of valid words from a file that I’ve provided for you. Each word is on a new line, so you should be able to read it in directly to a list.

Your program will then randomly select one of those words to be the selected word for that round. By using a for loop, we can break that word down into characters and store them in a list. This list will serve as our ‘answer key’, the full list of correct answers.

You’ll want to put your entire program in a while loop that will loop while the player still has attempts, printing out the current state of the hangman and the word with however many letters the player has already guessed. It’s optional, but I’d recommend printing out their incorrect guesses as well.

Each time the player guesses, you’ll want to check in that list to see if their guess is contained within it, and if so, what indexes it’s contained at. Each index that’s discovered should be added to a second list so we can display what they’ve currently guessed. You’ll also want to check if the player has entered the entire word on each guess, just as a failsafe.

If the player makes an incorrect guess, well first off – skill issue, but secondly, we’ll need to keep track of that letter in a list of incorrect guesses (make sure the guess doesn’t go through if it’s already in that list).

If the number of incorrect guesses exceeds your personal loss condition for hangman, then display a game over message.

**RUBRIC**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Criteria** | **Advanced (4)** | **Proficient (3)** | **Developing (2)** | **Beginner (1)** |
| Functionality | The program correctly selects a random word from the provided list, accurately tracks the player's guesses, and correctly determines the win/loss conditions. All edge cases are handled seamlessly. | The program selects a random word, tracks guesses, and determines win/loss, but may have minor issues with edge cases. | The program has issues with selecting a word, tracking guesses, or determining win/loss conditions. Significant bugs are present. | The program does not function as intended. Major issues with core functionality. |
| Code Structure and Organization | Code is exceptionally well-organized, modular, and easy to understand. Appropriate use of functions, variables, and control structures. Excellent documentation and comments. | Code is well-structured, with good use of functions, variables, and control structures. Documentation and comments are present. | Code structure is somewhat disorganized, with room for improvement in the use of functions, variables, and control structures. Limited documentation. | Code is poorly structured, with inadequate use of functions, variables, and control structures. Little to no documentation. |
| Error Handling | The program gracefully handles all possible user inputs and edge cases, providing clear and helpful feedback to the player. | The program handles most common user inputs and edge cases, with only minor issues. Feedback to the player is generally clear. | The program has issues handling certain user inputs or edge cases, and the feedback to the player is sometimes unclear or unhelpful. | The program does not properly handle user inputs or edge cases, and the feedback to the player is inadequate or missing. |
| Creativity and design | The program includes creative and engaging visual elements, such as ASCII art or animations, that enhance the overall user experience. The game design is polished and intuitive. | The program includes some visual elements that contribute to the user experience. The game design is functional and reasonably intuitive. | The program has limited visual elements, and the game design could be improved to enhance the user experience. | The program lacks visual elements, and the game design is poorly conceived or executed. |

| **Criteria** | **Advanced (4)** | **Proficient (3)** | **Developing (2)** | **Beginner (1)** |
| --- | --- | --- | --- | --- |
| **Functionality** | The program correctly selects a random word from the provided list, accurately tracks the player's guesses, and correctly determines the win/loss conditions. All edge cases are handled seamlessly. | The program selects a random word, tracks guesses, and determines win/loss, but may have minor issues with edge cases. | The program has issues with selecting a word, tracking guesses, or determining win/loss conditions. Significant bugs are present. | The program does not function as intended. Major issues with core functionality. |
| **Code Structure & Organization** | Code is exceptionally well-organized, modular, and easy to understand. Appropriate use of functions, variables, and control structures. Excellent documentation and comments. | Code is well-structured, with good use of functions, variables, and control structures. Documentation and comments are present. | Code structure is somewhat disorganized, with room for improvement in the use of functions, variables, and control structures. Limited documentation. | Code is poorly structured, with inadequate use of functions, variables, and control structures. Little to no documentation. |
| **Error Handling** | The program gracefully handles all possible user inputs and edge cases, providing clear and helpful feedback to the player. | The program handles most common user inputs and edge cases, with only minor issues. Feedback to the player is generally clear. | The program has issues handling certain user inputs or edge cases, and the feedback to the player is sometimes unclear or unhelpful. | The program does not properly handle user inputs or edge cases, and the feedback to the player is inadequate or missing. |
| **Creativity & Design** | The program includes creative and engaging visual elements, such as ASCII art or animations, that enhance the overall user experience. The game design is polished and intuitive. | The program includes some visual elements that contribute to the user experience. The game design is functional and reasonably intuitive. | The program has limited visual elements, and the game design could be improved to enhance the user experience. | The program lacks visual elements, and the game design is poorly conceived or executed. |