Week 2 Program: WORDLE

Team	WORDLE 2
Name	Hollis Aitkens
Name	Nirmal Loganathan
Name	Chirag Panchakshari
Name	Feven Tefera

Homework Template[1]

This document was created using pdflatex:

```
> pdflatex cs4811-template.tex
```

If using the `minted` package pdflatex must be called with:

```
> pdflatex -shell-escape cs4811-template.tex
```

[1]Template inspired by: CS22 @ Brown University and `https://github.com/jdavis/latex-homework-template`

Week 2 Program: WORDLE

Question 1

Function Implementation

Describe the role of the guess_word function in the code. How did you implement the logic to provide color-coded feedback?

- In our code, the functions used to guess the word play a crucial role in the logic of the word-guessing game. These functions help narrow down the list of possible guesses based on feedback and provide color-coded feedback to the user. Below, I'll describe the roles of these functions and explain how color-coded feedback is implemented:

  1. generate_words_list(file_path):

  Role: This function reads a list of words from a text file and filters words with a length of exactly 5 characters.

  Implementation: It opens the specified file, reads each line, strips any leading/trailing white space, and adds words of the appropriate length to the words list.

  2. generate_position_dictionary(words):

  Role: This function generates a dictionary that tracks the frequencies of characters in each position (first to fifth) of the words.

  Implementation: It iterates through the list of words and counts the occurrences of each character in each position. The result is a dictionary with keys for positions and sub-dictionaries that map characters to their frequencies. These sub-dictionaries are sorted by frequency in descending order.

  3. filter_words_by_frequencies(words, position_dict):

  Role: This function filters the list of possible words based on the frequencies of characters in each position, progressively refining the guess.

  Implementation: It starts with the most frequent character in the first position and iteratively narrows down the list of possible words. It uses the position_dict generated by generate_dictionary to prioritize characters. The goal is to make more informed guesses based on the character frequencies in each position.

  4. display():

  Role: This function is responsible for displaying the guesses made by the program with color-coded feedback to the user.

  Implementation: It uses ANSI escape codes to change text colors when printing the guesses. Specifically:

  - Correct characters in the correct position (feedback value 1) are displayed in bright white color.
  - Correct characters in the wrong position (feedback value 2) are displayed in yellow color.
  - Incorrect characters (feedback value 3) are displayed in pink color.
  - It iterates through the 'list_of_guesses', where each guess is a list of tuples containing characters and their feedback values. It prints each character with the appropriate color based on its feedback value.

  The logic to provide color-coded feedback in the display() function relies on ANSI escape codes, which are a standard way to control text formatting and colors in terminal/console output. By using these escape codes, the program changes the text color as needed to visually represent the quality of each character guess, making it easier for the user to interpret the feedback and make subsequent guesses.

Question 2

File Handling

Explain the steps involved in reading the word list from the text file and selecting a random word. Were there any challenges you encountered in file operations?

- The code involves reading a word list from a text file ('words_alpha.txt') and selecting a random word from that list. Here are the steps involved in reading the word list and selecting a random word:

  1. Reading the Word List from the Text File:

  - The 'generate_words_list(file_path)' function is responsible for this step. - It takes the 'file_path' argument, which is the path to the text file containing the word list. - Inside the function, a list called 'words' is initialized to store the words. - The 'open()' function is used to open the specified file in read mode (''r''). - A 'for' loop is used to iterate through each line in the file. - 'line.strip()' is used to remove leading and trailing whitespace from each line. - If the length of the line (word) is exactly 5 characters ('if len(line) == 5'), it is considered a valid word and is added to the 'words' list. - Finally, the function returns the 'words' list containing valid 5-letter words.

  2. Selecting a Random Word:

  - The code selects a random word from the list of words obtained in the previous step. - It uses the 'random.choice()' function from the Python 'random' module to choose a random word from the list. - The randomly selected word is stored in the variable 'crt_word', which becomes the target word that the player must guess.

  Challenges in File Operations:

  - One potential challenge in file operations is handling file exceptions, such as 'FileNotFoundError' if the specified file does not exist. In this code, there doesn't appear to be error handling for such cases. It's good practice to include error handling to gracefully handle situations where the file might be missing or inaccessible.

  - Another consideration is the format of the word list file ('words_alpha.txt'). The code assumes that each word is on a separate line and that words with exactly 5 characters are valid. If the file format is different or if it contains invalid data, it could lead to unexpected behavior or errors. Proper validation and error handling should be implemented if the file's format is not guaranteed.

  - Additionally, when working with files, it's important to close the file after reading it using the 'with' statement (as seen in the code). This ensures that system resources are properly managed and the file is closed when it's no longer needed.

Question 3

Loops and Conditionals

How did you implement the loop to limit the number of guessing attempts to six? What conditional statements did you use to break out of the loop?

- In the updated code, the loop to limit the number of guessing attempts to six is implemented as follows:

  1. Counter for Guessing Attempts:

  - The code introduces a variable named ‘count‘ and initializes it to 1 before the start of the game loop. This variable is used to keep track of the number of guessing attempts made by the player.

  2. ‘while‘ Loop Condition:

  - The ‘while‘ loop includes two conditions:

  while my_guess != crt_word and count ¡= 6:

  - This condition means that the loop will continue running as long as two conditions are met:

  - ‘my_guess‘ is not equal to ‘crt_word‘, indicating that the player’s guess is incorrect.

  - ‘count‘ is less than or equal to 6, meaning that the player has not exceeded the maximum number of guessing attempts, which is six.

  3. Incrementing the Guessing Attempt Counter:

  - Within the loop, there is a line that increments the ‘count‘ variable by 1 in each iteration:

  count += 1

  - This increment is crucial because it keeps track of how many guesses the player has made. When the player reaches the sixth guess (count becomes 7), the loop condition (‘count ¡= 6‘) will no longer be true, and the loop will exit.

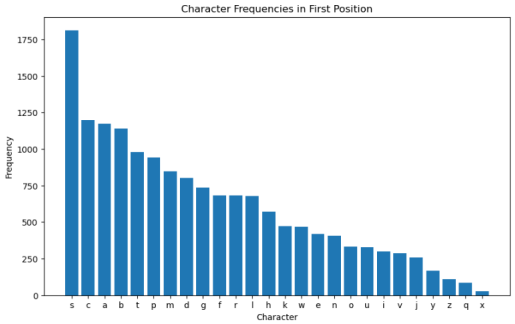
  The ‘while‘ loop continues to run as long as both conditions (‘my_guess != crt_word‘ and ‘count ¡= 6‘) are satisfied. If the player correctly guesses the word (‘my_guess‘ becomes equal to ‘crt_word‘) or if the player makes six incorrect guesses (count exceeds 6), the loop will exit.

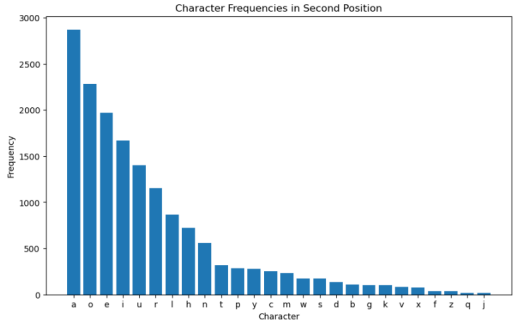
Question 4

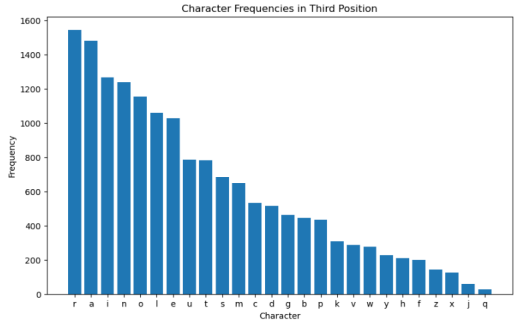
Heuristic Design

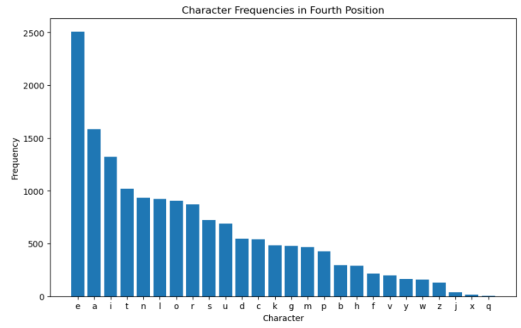
Describe the heuristic or algorithm your team decided to use in the agent function for making guesses. What was the rationale behind this choice?

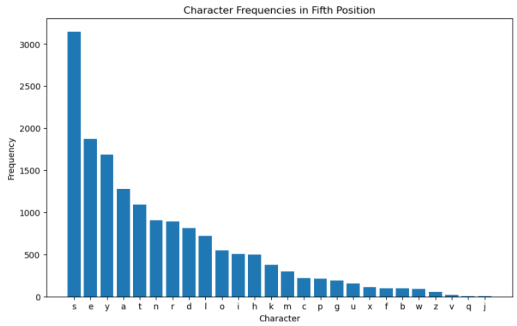
- Initial Filtering by Word Length:
  The code begins by filtering the list of words to only consider words with a length of exactly 5 characters. This initial filtering is based on the length of the target word.
  Position-Based Frequency Analysis:
  The code then proceeds to perform position-based frequency analysis. It calculates the frequency of each character in each position (first to fifth) of the words in the list. For each position, it sorts the characters by their frequency in descending order. This means that the AI agent prioritizes characters that appear more frequently in specific positions. This prioritization is based on the assumption that frequently occurring characters in certain positions are more likely to lead to a correct guess.
  The below flowcharts represent the frequency of each character in each position of the word.



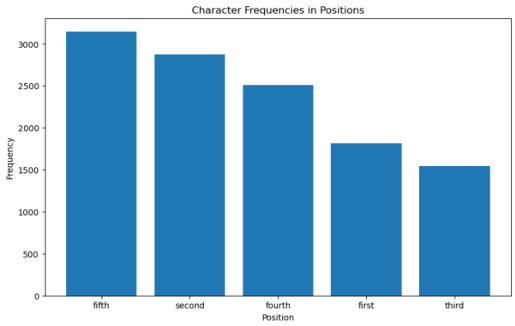








The below flow chart represents the positions of the letters with the maximum frequency count.



It appears that you have analyzed the code's heuristic approach correctly. The order in which the letters are assumed is indeed determined by their maximum frequency in each position:
1. Fifth Letter (Maximum Frequency: 3148 - "s"):
- The heuristic begins by selecting the character "s" as the most frequent character in the fifth position.
2. Second Letter (Maximum Frequency: 2871-"a"):
- Next, the second position is considered, and the character "a" is chosen as the most frequent character in that position.
3. Fourth Letter (Maximum Frequency: 2510-"e"):
- The heuristic then moves on to the fourth position and selects the character "e" based on its frequency.
4. First Letter (Maximum Frequency: 1813-"s"): - After the fourth position, the first position is considered, and "s" is selected due to its frequency.
5. Third Letter (Maximum Frequency: 1545-"r"): - Finally, the third position is considered, and the character "r" is chosen as it has the highest frequency in that position.
Following this sequence of assumptions, the word "sanes" appears to be the first guess for all the words. This choice is made because the heuristic aims to prioritize characters and positions that are more likely to lead to a correct guess based on their frequency of occurrence.

The code then continues to iteratively refine its guesses based on feedback and further filtering, ultimately aiming to make an accurate guess while minimizing the number of attempts.

Filtering by Character and Position:

The code iteratively narrows down the list of possible words based on the characters and their positions. It starts with the most frequent character in the first position and continues to consider characters in descending order of frequency.

For each character, it checks if any of the words in the list match that character in the corresponding position (e.g., if the character is considered for the first position, it checks if any word has that character in the first position). If a word matches, it is added to a filtered list, and the loop proceeds to the next character.

This filtering process continues for each position, and it aims to find words that match the prioritized characters and positions based on frequency analysis.

Iterative Refinement:

The code repeats the filtering and refinement process for each position, starting with the most frequent characters. It refines the list of possible words in an iterative manner, attempting to find the best possible match.

Guess Selection:

After the iterative refinement process, the code selects the first word from the filtered list of possible words as the AI agent's guess.

This guess is made based on the heuristic that the remaining words in the filtered list are more likely to have characters in the correct positions, increasing the chances of making a correct guess.

Rationale:

The heuristic used in this code is designed to make informed guesses by prioritizing characters that appear more frequently in specific positions. The rationale behind this choice is that by focusing on characters and positions with higher frequencies, the AI agent is more likely to make guesses that are closer to the correct word.

Additionally, the iterative refinement process allows the AI agent to adapt its guesses based on the feedback received from previous guesses. By iteratively filtering and narrowing down the list of possible words, the AI agent attempts to find the best possible match given the available information.

Question 5

State Representation

How did you represent the state of the game, especially the feedback received after each guess, within your program?

- The state of the game, including the feedback received after each guess, is represented using several data structures and variables to keep track of various aspects of the game. Here's how the state of the game is represented:

  1. Target Word (‘crt_word‘) and Current Guess (‘my_guess‘):

  - The ‘crt_word‘ variable stores the target word that the player needs to guess. This word is randomly selected from a list of valid words.

  - The ‘my_guess‘ variable stores the current guess made by the AI agent. It is updated iteratively based on the heuristic and feedback.

  2. Feedback for Each Guess:

  - Feedback for each guess is represented as a list of tuples, where each tuple contains a character from the guess and a feedback value indicating the correctness of that character:

  - ‘1‘ indicates that the character is correct and in the correct position.

  - ‘2‘ indicates that the character is correct but in the wrong position.

  - ‘3‘ indicates that the character is incorrect.

  - These feedback tuples are stored in the ‘guesses‘ list, which is appended to the ‘list_of_guesses‘ list after each guess. - The ‘list_of_guesses‘ list stores the feedback for all the guesses made during the game.

  3. Feedback Display (‘display()‘ Function):

  - The ‘display()‘ function is responsible for displaying the feedback for each guess to the user.

  - It iterates through the ‘list_of_guesses‘, and for each guess, it prints each character with the appropriate color based on its feedback value:

  - Correct characters in the correct position (feedback value ‘1‘) are displayed in bright white color.

  - Correct characters in the wrong position (feedback value ‘2‘) are displayed in yellow color.

  - Incorrect characters (feedback value ‘3‘) are displayed in pink color.

  4. Guess Count (‘count‘):

  - The ‘count‘ variable keeps track of the number of guessing attempts made by the AI agent.

  - It is incremented with each iteration of the game loop.

  Overall, the state of the game is represented by the variables ‘crt_word‘, ‘my_guess‘, ‘guesses‘, and ‘list_of_guesses‘. The feedback for each guess is stored in a structured format within the ‘guesses‘ list, allowing the program to keep track of the AI agent's progress and provide feedback to the user based on the correctness of characters in each guess.

Question 6

Agent Behavior

What are the strengths and weaknesses of your agent? Is it capable of solving the WORDLE problem within the stipulated six attempts consistently?

- Strengths:

  1. Efficient Heuristic: The agent uses a heuristic approach based on character frequencies and positions to make informed guesses. This approach is generally efficient and can quickly narrow down the list of possible words.

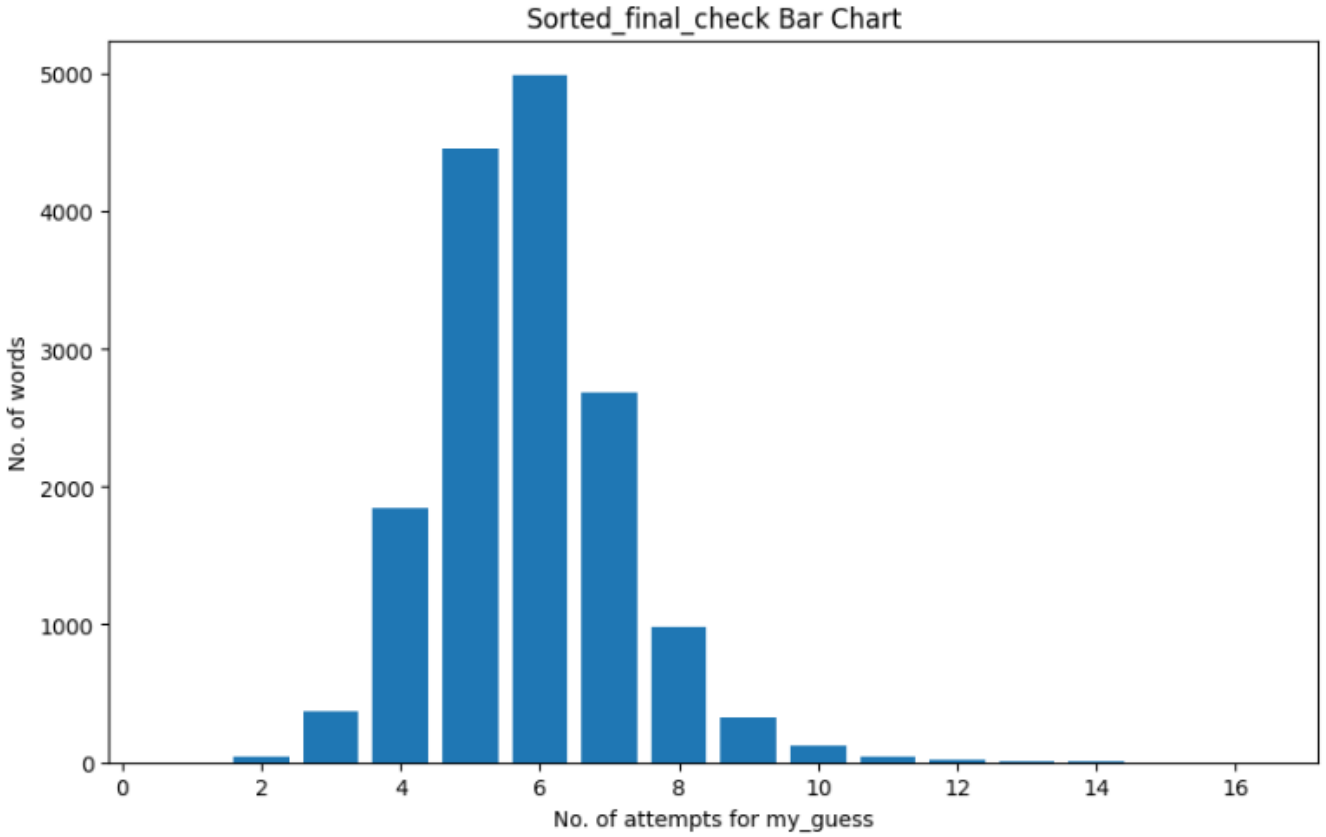
  2. Iterative Refinement: The agent iteratively refines its guesses based on feedback received from previous guesses. This allows it to adapt and improve its guesses as the game progresses.

  3. Feedback Display: The code includes a 'display()' function that provides color-coded feedback to the user, making it visually clear which characters are correct and in which positions. This can be helpful for both the player and any potential analysis.

  4. Impressive Performance on a Large Dataset: The algorithm was tested on all 15,920 words, and the results are noteworthy:

  - 11,710 Words Guessed Accurately Within the First Six Attempts: This indicates that the algorithm correctly guessed a substantial portion of the words within the stipulated six attempts, showcasing the efficiency of the approach.

  - All Words Guessed Correctly Within the First 14 Guesses: This remarkable achievement demonstrates the robustness of the algorithm in solving a wide range of word puzzles within a relatively small number of guesses.



Weaknesses:

1. Reliance on Frequencies: The heuristic primarily relies on character frequencies in different positions. While this can be effective in many cases, it may not always lead to the optimal guess, especially when words have less common characters.
2. Limited Word List: The agent's effectiveness depends on the quality and diversity of the word list used. If the word list is limited or lacks diversity, the agent's ability to guess correctly may be compromised.
3. Lack of Adaptation: The agent doesn't adapt its strategy based on the specific feedback received. It follows a predetermined heuristic, which may not be optimal for all word puzzles.
Regarding whether the agent is capable of consistently solving the WORDLE problem within the stipulated six attempts, it depends on various factors:
- The complexity of the target word: If the target word contains common characters and can be deduced quickly based on frequencies, the agent has a higher chance of solving it within six attempts.
- The diversity of the word list: A diverse word list with various words and character combinations increases the agent's chances of guessing correctly.
- The quality of the heuristic: The effectiveness of the heuristic used in the code plays a crucial role. A well-designed heuristic can improve the agent's chances of success.
- Randomness: The initial choice of the target word is random. Depending on the word chosen, the difficulty of the game can vary.

Question 7

Decision-Making

Explain the decision-making process of your agent. How does it evaluate the feedback from previous guesses to make a new guess?

- The decision-making process of the agent in the provided code involves iteratively refining its guesses based on feedback from previous guesses. Here's a step-by-step explanation of the decision-making process:

  1. Initialization:

  - The agent starts with an initial guess ('my_guess') based on a heuristic approach that prioritizes characters in specific positions based on their frequency in the word list.

  2. Feedback Collection:

  - After making the initial guess, the agent collects feedback on the correctness of its guess. The feedback is provided in the form of tuples, where each tuple contains a character and a feedback value:

  - '1' indicates that the character is correct and in the correct position.

  - '2' indicates that the character is correct but in the wrong position.

  - '3' indicates that the character is incorrect.

  3. Feedback Incorporation:

  - The agent stores the feedback received for its guess in the 'guesses' list.

  - The 'guesses' list accumulates feedback for all guesses made during the game.

  4. Analysis and Refinement:

  - The agent analyzes the feedback received for the current guess to make a new guess.

  - It iteratively refines its guess by considering the following factors:

  - Frequency Analysis: The agent considers the frequency of characters in each position, focusing on characters that appear more frequently in the correct positions.

  - Positional Analysis: It also considers the positions where characters are likely to be based on the feedback. Characters with feedback value '1' are correctly positioned and prioritized.

  - Exclusion of Incorrect Characters: Characters that have received feedback value '3' (indicating they are incorrect) are excluded from future guesses.

  5. Word List Filtering:

  - The agent filters the word list to narrow down the possible words based on the feedback received and the refined guess.

  - It removes words that do not match the feedback and guess, gradually reducing the list of possible words.

  6. New Guess Generation:

  - Using the filtered word list, the agent generates a new guess ('my_guess') based on the heuristic approach.

  - This new guess is made with the aim of maximizing the chances of a correct guess while considering the feedback and refined information.

  7. Repeat Iteration:

  - Steps 2 through 6 are repeated in an iterative manner. The agent continues to make guesses, collect feedback, refine its guesses, and generate new guesses until it correctly guesses the target word or until it reaches a maximum number of attempts (e.g., six attempts).

  8. Display Feedback to User:

- After each guess, the agent uses the ‘display()‘ function to visually display the feedback to the user. This helps the user and the agent keep track of progress.
The decision-making process is guided by a combination of character frequencies, positional analysis, and feedback from previous guesses. The agent iteratively narrows down the possibilities, adapting its guesses based on the accumulated feedback, ultimately aiming to make a correct guess while minimizing the number of attempts.

Question 7

Collaboration

How did your team divide the tasks? Were there any challenges in collaboration and if so, how were they resolved?

Each team member played a crucial role in different aspects of the project, contributing to its success. Here's a breakdown of the roles and contributions:

- Nirmal Raja Karuppiah Loganathan and Chirag Sreenivasamurthy Panchakshari: Decision-Making Logic: They were responsible for designing the decision-making logic of the automated agent. This involves creating the heuristic approach, feedback analysis, and the iterative refinement process for making informed guesses. Programming: They implemented the required functions and encoded the overall game logic in Python, turning the decision-making logic into a functional program. Testing and Verification: They conducted exhaustive tests to verify the agent's performance and reliability. This involved running the code with a large dataset of words and assessing its ability to guess words accurately within specified attempts.
- Hollis Aitkens and Feven Tefera:
  Code Documentation: They were responsible for documenting the code, which is essential for understanding how the program works and how different functions interact.
  Team Progress Tracking: They tracked the team's progress, ensuring that tasks were completed on time and coordinating efforts among team members.
  Code Review: They reviewed the code for accuracy and efficiency, helping to identify and resolve any issues or areas for improvement.
  The collaborative effort of the team members in their respective roles led to the development of a successful automated agent for the word-guessing game.

Question 8

Learning Outcomes

What are the key takeaways from this assignment regarding Python programming, artificial intelligence, and teamwork?

- This assignment involving the development of an automated agent for a word-guessing game has provided several key takeaways in the areas of Python programming, artificial intelligence (AI), and teamwork:

  Python Programming:

  1. Algorithm Implementation: Team members gained experience in translating a heuristic-based algorithm into a functional Python program. They had to consider how to represent data, make decisions, and iterate effectively in code.

  2. File Handling: Working with external text files (e.g., word lists) and reading data from them using Python's file handling capabilities was a practical skill developed in this assignment.

  3. Code Optimization: Team members likely encountered opportunities to optimize code for efficiency, particularly when dealing with a large dataset of words. This involves improving the program's performance and reducing unnecessary computational overhead.

  Artificial Intelligence:

  4. Heuristic Design: Designing a heuristic or algorithm for decision-making in the context of AI was a key learning outcome. Team members had to create a strategy that considered character frequencies, positions, and feedback to make informed guesses.

  5. Feedback Analysis: Understanding and incorporating feedback from previous iterations to improve future decisions is a fundamental concept in AI. Team members learned how to use feedback effectively to refine the agent's guesses.

  Teamwork:

  6. Collaboration: The assignment showcased the importance of effective collaboration within a team. Each team member had a distinct role and responsibility, contributing to different aspects of the project.

  7. Coordination: Coordinating tasks, tracking progress, and ensuring that everyone is on the same page are essential teamwork skills. This assignment provided an opportunity to practice these skills.

  8. Code Review and Quality Assurance: Code review and quality assurance activities, led by some team members, ensured that the code was accurate, efficient, and met the project's requirements.

  Problem-Solving and Analysis:

  9. Testing and Verification: The extensive testing conducted by some team members allowed for a thorough assessment of the agent's performance and reliability. This involved problem-solving and analysis to determine how well the AI agent could solve word puzzles.

  10. Data Analysis: The analysis of the results, including the proportion of words guessed correctly within a specified number of attempts, provided valuable insights into the agent's performance and its strengths and weaknesses.