Wordle Battle

Team Members:

Serafina Yu, Caroline Ek, Zeid Zawaideh

Problem Description

Wordle is a web game by the New York Times where you try to guess a 5 letter word with 6 tries. The game only lets you submit valid 5 letter words and lets the player know whether or not the letter in their guess is in the word, in the word but in the wrong position, or not in the word. At the end of the game, players are able to share their results with friends as a friendly competition to see who guessed it in as fewest tries as possible. To simulate a player vs player situation, we created a version of Wordle that allows the player to compete against an AI using the same amount of turns (6). Whoever guesses the word first wins.

Approach:

Since this is a game where the AI has a large number of possible words/states, we decided to try to implement the First-choice Hill Climbing algorithm for choosing words. In our version of the game, the user will always be the first player to begin the game and afterwards alternate turns with the AI. To keep track of what letters are in the correct position, we store the letters in a variable called display_word. Initially display_word starts as just 5 stars, each star acting as a placeholder until the correct letter is unveiled.

Since the player chooses the first word, the AI will use hints about what letters are and are not in the word to determine its neighbors. Normally when choosing neighbors in a hill climbing algorithm, we would make small changes in our current state which results in a neighbor. However, in scenarios where there are no hints as to what letters might be in the word, we might just choose randomly from the list of possible words. Thus our generate_neighbors function returns a list of possible solutions that have not already been guessed. To narrow down the AI's guesses, we create a fitness function which gives a score to each neighbor–adding 2 points for each letter in the correct spot, 1 point for letters in the incorrect spot, and -1 point for every invalid letter used. There are many scenarios in which multiple neighbors will have the same point value. As a first-choice hill climbing algorithm, it randomly chooses a neighbor amongst the best neighbors.

After each turn, a check_guess function updates attributes to represent the number of moves left in the game, what letters are still valid, what letters have been exposed in the target word, and what letters are in the incorrect position. These attributes are used in the fitness_function, allowing the AI to score each neighboring word appropriately.

Software Details

Programming Language: Python3

Datasets: We will be using a dataset containing all possible Wordle solutions and words that are allowed as guesses as well from Kaggle by Bill Cruise. There are approximately 2000 possible solutions and over 10,000 possible words that are allowed as guesses but will never be solutions.

Existing Code: No existing code. This program was developed from scratch.

Libraries: The built-in Python libraries random and string were the only libraries used in this program. The random library was used to choose the Wordle word randomly and to allow the AI to choose randomly from the list of best neighbors. The string library was used to store the alphabet into a variable.

Our program is an executable Python file which starts the game in the terminal for the user to play against the AI.

Evaluation

Playing our game, we realize that it is not very user-friendly in that it does not let the player know at all times what letters are invalid. As a result, it is much harder to win against the AI since it is harder to remember what letters were used. However, if a proper UI or web app was made for this program, we believe users would have a much better chance against the AI.

To evaluate the time complexity of the AI's hill climbing algorithm, we took into account all functions used during the AI's turn: fitness_function(guess), hill_climbing(max_iterations), and generate_neighbors(). The generate_neighbors function iterates through the potential_words list thus the time complexity for this is O(w) where w is the number of words in the list. The hill_climbing function iterates the amount of times max_iterations is set to and has a nested for loop that iterates through each neighboring word at each iteration. So the time complexity for this function is O(m*n) where m is the value of max_iterations and n is the neighboring words. Finally, fitness_function iterates through each letter of a neighboring word so its time complexity is O(l), where l is the number of letters in the word. Since the fitness_function and generate_neighbors functions are both nested in the for loop of the hill_climbing function, the time complexity would be O(w*m*n*l).

```
def fitness_function(self, guess):
       correct_position_score = 2
       correct_letter_score = 1
       invalid_letter_score = -1

       score = 0
       for i, letter in enumerate(guess):
           if letter == self.display_word[i] and letter != "*": # If
letters are in the correct position (ignore * symbol)
               score += correct_position_score
           elif letter in self.correct_letters_wrong_pos: # If letters are
in the incorrect position
               score += correct_letter_score
           elif letter not in self.valid_letters and letter != "*": # -1
point if using invalid letters (ignore * symbol)
               score += invalid_letter_score
       return score

   def generate_neighbors(self, word):
       # neighbors = []
       neighbors = [
           word for word in self.potential_words # Allow neighbors to be
pulled from the entire dataset of possible solutions
           if word not in self.guesses # Choose words from this lest that
haven't been guessed already
       ]
       return neighbors

   def hill_climbing(self, max_iterations):
       best_guess = None
       best_score = 0
       current_guess = self.display_word
       current_score = self.fitness_function(current_guess)
       print(current_guess, current_score)
       top_neighbors = []
       all_neighbors = {}

       for _ in range(max_iterations):
           neighbors = self.generate_neighbors(current_guess)
           for neighbor in neighbors:
               neighbor_score = self.fitness_function(neighbor)
               all_neighbors[neighbor] = neighbor_score
               if neighbor_score > current_score:
                   current_guess = neighbor
                   current_score = neighbor_score
                   if neighbor_score >= best_score:
                       best_guess = current_guess
```

```python
                        best_score = neighbor_score
                        print("Best score:", best_score)
            else:
                break  # No better neighbor found, terminate hill climbing

         # Find the max score
        max_score = max(all_neighbors.values())

        # Create a set to store words with max values
        best_guesses = {key for key, value in all_neighbors.items() if value == max_score}

        # Print the keys corresponding to the words with max values
        for key in best_guesses:
            print(key, all_neighbors[key])
        if best_guess:
            print("Best guess:", best_guess)

        return random.choice(list(best_guesses)) if best_guess else random.choice(neighbors)

    def ai_guess(self):
        return self.hill_climbing(10000)
```

Conclusion:

Through Battle Wordle, we learned how to apply the first-choice hill-climbing algorithm with attempts to select the best possible word among many other neighbors. By creating a fitness function, neighbors were narrowed down and it is very likely for the AI to win against the user as of current. While our game is only available through the terminal, we would like to create a user interface in the form of a web application for anyone to play. The user interface would also be built with the purpose of making it easier for players to identify what letters they have and have not used already, increasing their chances of winning against the AI.

References:

- Bill Cruise, 2022, Wordle Valid Words
    - https://www.kaggle.com/datasets/bcruise/wordle-valid-words?select=valid_guesses.csv