843. Guess the Word

Problem Explanation

We are given a guessing game, where we are given a list of unique six-letter words, and one of those words is the "secret". We can make a guess on what the secret word could be by calling `master.guess(word)`. If the word we guess is not in the word list, it returns -1. Otherwise, it returns an integer indicating the number of matches of both letter and position our guess has with the secret word.

We are tasked to find the secret word within 10 guesses. If we manage to correctly guess the secret word within 10 guesses, we pass the test case.

For example, given secret = "acckzz", and wordlist = ["acckzz","ccbazz","eiowzz","abcczz"], if we guess "aaaaaa", -1 would be returned as it's not in the word list. If we guess "acckzz", 6 would be returned as it's the secret word. If we guess "ccbazz", 3 would be returned as it has three matching characters at correct positions with the secret word.

Solution Approach

The solution approach involves generating random secret guesses. If a guess does not match the secret word, we eliminate all words from the word list that wouldn't have resulted in the same guess result.

This is done within a for loop that runs 10 times. At each iteration, we make a guess from the current word list. If the guess fully matches the secret (i.e., all characters match), we break the loop. If not, we update the word list to only include words that have the same number of matches with our guessed word, and continue to the next guess.

Python Solution

```
python
import random
class Solution:
    def findSecretWord(self, wordlist: List[str], master: 'Master') -> None:
        for i in range(10): # You have 10 guesses
            guess = random.choice(wordlist) # randomly pick a word from the wordlist
            x = master.guess(guess) # check how many characters match
            wordlist = [w for w in wordlist if sum(i==j for i,j in zip(guess, w)) == x]
        # filter wordlist for the next iteration to only include words that would have the same match result
```

Java Solution

```
java
import java.util.*;
class Solution {
    public void findSecretWord(String[] wordlist, Master master) {
        for (int i = 0; i < 10; ++i) {
            String guess = wordlist[new Random().nextInt(wordlist.length)];
            int x = master.guess(guess);
            List<String> wordlist2 = new ArrayList<>();
            for (String w : wordlist)
                if (match(guess, w) == x)
                    wordlist2.add(w);
            wordlist = wordlist2.toArray(new String[0]);
        }
    }

    public int match(String a, String b) {
        int matches = 0;
        for (int i = 0; i < a.length(); ++i)
            if (a.charAt(i) == b.charAt(i))
                matches ++;
        return matches;
    }
}
```

C++ Solution

```
cpp
class Solution {
public:
    void findSecretWord(vector<string>& wordlist, Master& master) {
        srand(time(nullptr));  // Required

        for (int i = 0; i < 10; ++i) {
            const string& guessedWord = wordlist[rand() % wordlist.size()];
            const int matches = master.guess(guessedWord);
            if (matches == 6)
                break;
            vector<string> updated;
            for (const string& word : wordlist)
                if (getMatches(guessedWord, word) == matches)
                    updated.push_back(word);
            wordlist = std::move(updated);
        }
    }

private:
    int getMatches(const string& s1, const string& s2) {
        int matches = 0;
        for (int i = 0; i < s1.length(); ++i)
            if (s1[i] == s2[i])
                ++matches;
        return matches;
    }
};
```

JavaScript Solution

```
javascript
/**
 * // This is the Master's API interface.
 * // You should not implement it, or speculate about its implementation
 * class Master {
 *   guess(word: string): number {}
 * }
 */
var Solution = function() {
    this.findSecretWord = function(wordlist, master) {
        for (let i = 0; i < 10; i++) {
            let guess = wordlist[Math.floor(Math.random() * wordlist.length)];
            let x = master.guess(guess);
            wordlist = wordlist.filter(w => getMatches(guess, w) === x);
        }
    };

    var getMatches = function(s1, s2) {
        let matches = 0;
        for (let i = 0; i < s1.length; i++)
            if (s1[i] === s2[i])
                matches++;
        return matches;
    };
};
```

C# Solution

```
csharp
using System;
using System.Collections.Generic;

public class Solution {
    public void FindSecretWord(string[] wordlist, Master master) {
        var rand = new Random();
        for (var i = 0; i < 10; ++i) {
            var guess = wordlist[rand.Next(wordlist.Length)];
            var x = master.Guess(guess);
            var newList = new List<string>();
            foreach (var word in wordlist) {
                if (Match(guess, word) == x) {
                    newList.Add(word);
                }
            }
            wordlist = newList.ToArray();
        }
    }

    private static int Match(string guessed, string word) {
        var matches = 0;
        for (var j = 0; j < guessed.Length; j++) {
            if (guessed[j] == word[j]) {
                matches++;
            }
        }
        return matches;
    }
}
```

Please note that we are using a random picking strategy here, which means the solutions have a non-deterministic behavior which depends on the specific "randomness" at each runtime.# Conclusion

The above-specified approach provides the solution to the guessing game in Python, Java, JavaScript, C#, and C++. Since we're dealing with randomness, the number of guesses it takes may change each time we run our program.

However, the approach maintains its effectiveness through its repeated testing on a reduced sub-list. The list gets smaller after every guess, making it likely to find the secret word within the allowable 10 guesses.

The random choice method ensures we're examining different words every time; the filtering process discards words that are less likely to be the secret one. We're left with words that shared the same number of matching letters (in the right locations) with our previous guess.

The critical thought behind this solution is to use the information received from every guess to narrow down the list of possible secret words.

Despite the simplicity of this approach, we must also keep in mind that it may not guarantee the identification of the secret word within the given limit of 10 guesses in every possible scenario. For this reason, it possesses a probabilistic nature. However, it would work well in most cases, making it an efficient solution for this task.