

# Server Engineering Challenge - Barcelona Studio

Together with this document, you will find a zip file **CodeChallenge.zip** with 2 folders, one named **Program** and another named **Resources**. The **Program** folder has a Visual Studio solution with a project for the game you are to make and a project for unit testing your game. The **Resources** folder will have text files with resources to use such as the English dictionary to validate words against (american-english-large.txt), the letter reels (reels.txt), and the scores that should be awarded for each letter (scores.txt).

## Gameplay

The way that you play ReelWords is a bit like a slot machine. There are 6 reels in total and each reel is made up of various characters. These characters are **not** random but are in a set sequence that doesn't change.



If you look at the above image, you will see a mockup of the ReelWords game. In the green rectangle are the letters that the player can use to make up their word. They can use these letters in any order, however they can only use each letter tile **once**. Notice that there are 2 **K** characters in the above screenshot, that means that the word that the user submits can have up to 2 **K** letters in the word that they submit.

The red rectangle in the above screenshot is the **Reel**. If the player were to use the letter **D** in the word that they submit, then **D** will be moved back to the top of the reel and the next letter that would move down is the letter **Q**. This will happen infinitely and always in the same order.

In the above screenshot, if the player were to play the word **RAKE**. Then the last 4 reels will move down to the next letter and all each letter that was used will move back to the top of the reel to be re-used. Also, in this case, the player will earn 8 points as denoted by the number in the bottom right of each tile in the screenshot above; this is the letters score.

# Trie Data Structure

*Feel free to ask for more information or look up more information online as needed.*

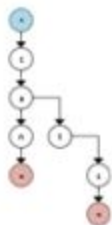
The Trie is a simple data structure that allows for fast indexing of words that can be quickly searched. You start with a root node that is identified by a character that is not allowed to be normally searched, in this case we are indexing the English language, so the letters we **cannot** use are the alphabet (a-z). This means any symbol characters will be sufficient for use. Let's use a ^ (carrot) character since it looks a bit like the top of a tree.

Each node has any number of child nodes, in this case the maximum children are 26 since we are using the English language and there are 26 unique characters. If we were to insert the word "can" into the trie, we would start at the root node, then add each character in order as a child of each other.



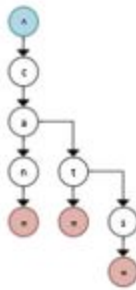
On the left you will see a diagram of the trie when we insert the word "can". We start at the root node (in blue) and then add the first letter as a child of the root, next we add each subsequent letter as a child of the previous. When we finally insert all the letters, we tell the trie that we've inserted the last letter of the word by inserting a child character symbol (not a-z). We will use the = (equals sign) character to say that it is the end of a word (the red node).

In the next sample we are going to add the word "cats" to the trie.

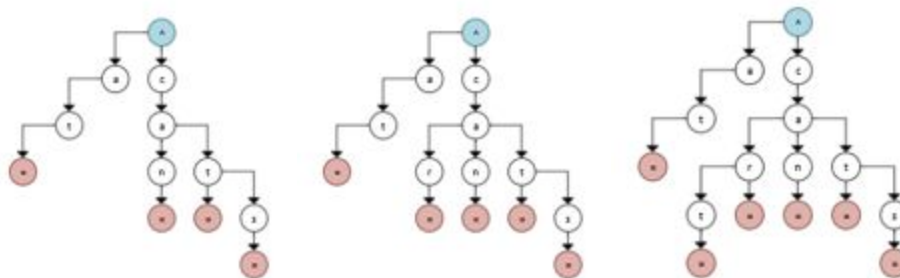


In this case we begin at the root node and look for the first letter we want to insert. We find the letter "c" so we move to that node and check for the next letter in "cats" which will be "a". We see that there is already an "a" child for the "c" node and so we move to that node and check the next letter. We check for the letter "t" and find that there is not a child node for this, so we create one and now we are in the same state as the first insert and can insert "s" as a child and then the end node (=).

Inserting continues in this form. Notice if we were to insert the word "cat" that we already have all the nodes already for the word "cat" so we just need to add an end node (=) to the last child node for "cat" which is "t".



In the next images we will add the words “at”, “car”, and “cart” respectively.



At this point we have discussed how we can insert any word into the trie in memory. From this information you are to infer how to **Search** for words as well as **Delete** words from the trie while keeping its integrity.

## Resource File Information

### american-english-large.txt

This is a list of valid words that the user can play in the game. Each entry is separated by a UNIX line ending. The user should not be able to play any words that are not found within this word list. This word list is to be parsed into the in-memory lookup data structure which will be described in these instructions.

### reels.txt

This is a list of all the letters that make up each reel of the game. Each row is a unique reel, so where “u d x c l a e” are the letters that make up one reel, “e y v p q y n” are the letters that make up the next. Each reel is separated by a UNIX line ending and each character on the wheel is separated by a space character.

## scores.txt

This is a list of all the letters in the English alphabet and the score that should be awarded for using the associated letter. Each letter is separated by a UNIX line ending; the letter and score are separated by a space character.

## Code Challenge / Expectations

1. Use the included .sln file to write your code
2. Create the Trie data structure with the following functions
  - a. Insert
  - b. Search
  - c. Delete
3. Read the american-english-large.txt file and insert all words into the Trie
4. Create a game of ReelWords
  - a. Generate reel objects using the provided reels.txt file (these reels should start at random positions as a slot machine would end at random positions)
  - b. Show the player the list of letters that they can input
  - c. Compare player input word against letters in a row (see gameplay section for more information) to make sure they input valid letters
  - d. Inform the user of invalid input
  - e. Inform the user of invalid words
  - f. Inform the user of the points they've earned from valid word input and present the new set of letters they can pick
  - g. When a successful word is submitted, correctly move the used letters to the top of the reels as explained in the gameplay section
5. Create unit tests for your code (all are required to pass) in the provided Tests project (This uses the standard Visual Studio Test framework). There are already tests written for Trie which should pass

We should be able to run the program and play a console-based version of Reel Words by typing the word we wish to submit into the command prompt.

## General guidance


### Language and frameworks

Although the base code is provided in C# (the main language we use), you are free to use whatever language you want.

## Outside Resources

You are welcome to consult any external, online resources that you find helpful as you work on this challenge, but be prepared to explain your work and ensure you understand your solution thoroughly.

## Submission

When you have completed your implementation, archive it and email it to the  interviewer who requested you complete the challenge.

## Evaluation

Apart from the core of the challenge, we will value other aspects like how well the application is tested, how clean is the code or how organized the project is, so we recommend spending some time on these aspects too.