# Megordle: A Wdolre Usclnbaermr

Meg Arney, Department of Computer Science
Faculty Advisor: Dr. Matthew Beauregard

## Abstract

Our goal was to make a dynamic word unscrambler app based on the random generation of words from a database.

The original database contains 4,320 of the most common words in the English language.

This project was originally designed in Visual Studio Code in java and takes input from the terminal. The project was later translated into C#, transferred into Unity, and provides a visual and dynamic interface for users.

## Benefits

According to *The Oakland Press,* word games build vocabulary, improve focus, stimulate the brain, and release endorphins, in addition to providing entertainment. Word games have also caused an improvement in memory, cognitive skills, concentration, and problem solving.

## Inspirations and Influences

This game was originally based off a puzzle in an IQ puzzle book that displayed letters in a random order across a graph. The goal of this puzzle was to follow the edges and nodes of the graph to find the longest word without reusing edges or nodes. Leaning into the random order of the nodes in this graph, the game was developed as a word unscrambler where the user is given a scrambled word and are tasked to figure out what the word unscrambled to.

After play-testing the unscrambler, there was a desire to have some form of a hint or confirmation that the player was on the right track. Thus, inspiration was taken from Wordle. This was implemented so that if a player typed their guess into the terminal, the terminal would give a wordle-like output. For example, if the word was “court”, the scrambled word displayed as “rtcuo”, and the player guessed “cortu”, then the terminal would output “c o _ _ _”.

## Selecting A Word Length and Increasing Difficulty

In order for the game to allow for different word lengths, we needed to develop a way for users to select how long the word would be. To do this, we developed a system for users to select a level. Users can select between 3 levels: Level 1 (5 letters), Level 2 (8 letters), or Level 3 (12 letters).

To increase the difficulty of the game, we provided a way for users to be given extra letters that were unused in the word. To do this, we developed a system for users to select a difficulty. Users may select between 3 difficulties: Level 1 (no extra letters), Level 2 (1 extra letter), or Level 3 (2 extra letters).

## Terminal-Based Solution

The first version of Megordle was designed and implemented in Visual Studio Code in java and takes input from the terminal. This was where the logic for the game was designed. Such logic includes but is not limited to:

- Sorting the database
- Selecting a word length
- Preventing repetition
- Adding unused letters
- Scrambling the word
- Attempts system
- Checking guess
- Hint system
- Point system
- Streak system
- Cash system
- Market
- Help menu

Strategies had to be developed and tested for each of these categories. For example, when finding a method to prevent repeating words, the first idea discussed was to create a List for the words that had already been used and iterate through that List when selecting a word to make sure it did not already exist in that List. This strategy was eventually scrapped in favor of using a HashMap called usedWords that stored the word sorted alphabetically in its key and the word in its value. This decreased the time complexity and allowed for an easier method of checking if the word was already stored by checking whether or not the method .get(word) returned null.

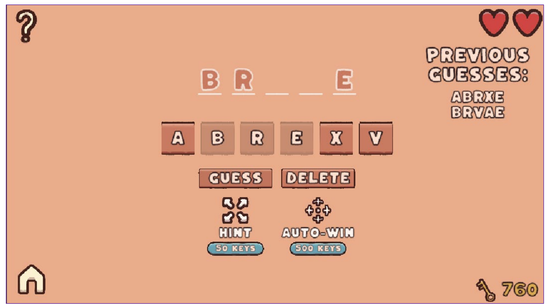


**Figure 3** A screenshot displaying a test game of the Unity-Based version playing level 1 difficulty 2

## Converting to Unity

The second and current version of Megordle was translated from java into C# so that it could be implemented in Unity. Although most of the logic converted with little issue, there were some methods that had to be reconfigured to better suit C#.

One example is the aforementioned method of preventing repetition. Originally, the usedWords HashMap was directly translated into C#’s version of a HashMap, the Dictionary. This was later changed when I discovered C#’s List held only one parameter instead of the two that the Dictionary required and had a .get() method. The List was implemented to just take the word that was being used and allowed for the used words to be stored more efficiently.

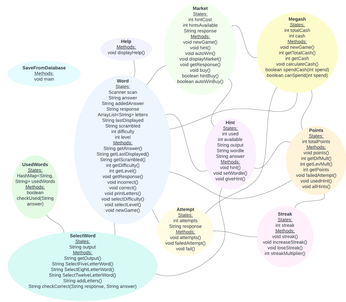


**Figure 1** A graph displaying the methods for the terminal-based solution

**Figure 2** A demo of the Terminal-Based version

## Unity-Based Solution

The Unity-based version of Megordle was built with Unity and used assets found on itch.io. Aside from the code conversions, the input and output for this version had to be greatly changed and improved upon.

One example is how data is saved for the game. In Unity, PlayerPrefs is a class that stores Player preferences between game sessions. They can store strings, floats, or integer values into the user’s platform registry.

PlayerPrefs were used in order to save a list of words the player has not encountered yet. The databases for each of the levels is a text file where all of the words are on one line and are separated by commas. These text files are connected to the game in Unity’s Inspector as TextAssets. If it is the first time the player has opened the game or they have reset their data, the text files are pulled and put into a string that is saved in a PlayerPref called “FiveLetterList”, “EightLetterList”, and “TwelveLetterList”, respectively. The words are then separated into three different Lists; one for level 1, level 2, and level 3.

When a game is selected, the word for that game is removed from their List. If the player leaves the game, the Lists are individually converted to a string where each word is separated by a comma and stored in the associated LetterList PlayerPref. When the player opens the game again, these strings are converted back into Lists from which the words are randomly selected.

**Figure 4** A demo of the Unity-Based version

## Future Directions

Megordle is working towards becoming accessible to anyone who wants to play it on desktop with hopes that it will also be fitted to be downloadable on mobile devices.

## Contact

Meg Arney, Dr. Matthew Beauregard
Department of Computer Science
P.O. Box 13063, SFA Station
Nacogdoches, Texas 75962
Arneymm@jacks.sfasu.edu
936.468.2508

## Unity Assets

Icons: https://penzilla.itch.io/vector-icon-pack?download
Font: https://ggbot.itch.io/kaph-font?download
Music: https://not-jam.itch.io/not-jam-music-pack
Sound effects: https://ellr.itch.io/universal-ui-soundpack

## References

1. Word Frequency data (https://www.wordfrequency.info/intro.asp)
2. The Oakland Press (https://www.theoaklandpress.com/2022/06/12/the-brain-boosting-benefits-of-word-games/)