Project Report for a Spelling Game

*How the AI blocks are included into the program:*

This project had to use the new AI blocks in Snap! It involves speech synthesis, speech recognition and word embedding. This idea for the program came from Chapter 1 about speech synthesis on the website which suggested making a spelling game where the words to be spelled are spoken. The program uses speech synthesis so that a voice calls out the spelling words for the game. It is also used to convert the instructions into speech. Speech recognition is used to recognise the theme for the game that the user has chosen. The user is asked for a theme and they can say their answer into the microphone. If they do not have access to a microphone, they also have the option to type in the theme that they want. Word embedding is used to select the words that the users are tested on. Once a theme has been chosen, the spellings are based on words that are close to the theme using Euclidean distance. The ten closest words are selected so the user is tested on ten spellings to do with their theme.

*The experience of using the AI blocks:*

The speech synthesis blocks were easy to use because you simply have type in the instructions or word that is read out. Speech recognition was much more difficult. At first, when this part of the program was not working, the user has to wait to type in the theme they want. I thought I found this difficult because I found the sample program on the website confusing. I tried copy the layout of the sample but I was not sure which variables are being used and where. However, after refreshing the browser, the problem was fixed. Using the block for finding the closest word to another word was simple because it is clear where to put the information. However, I struggled programming having a different word each time. At first I tried to add the next closest word to an array of words that the user had been tested on by creating a new array, but this caused the theme and its closest word to be repeated. Then I found the block which adds to array. This solved the problem. The only issue left is that the program does not differentiate between the singular and plural of a word, however, I have concluded that it is important to be able to spell both.

*Analysis:*

This is a basic structure diagram for the program. However, I did not split the program into subroutines because the program is short and manageable and I was not familiar with creating subroutines or functions when I wrote it.

Snap! uses a graphical interface and the messages appear in speech bubbles. However, the spellings are spoken by the computer using speech synthesis, therefore, for consistency, the messages are also read out.

A theme is chosen so the spellings are not random. The user can speak in to the microphone, and using speech recognition the computer will set the theme. If there are any errors, or the user has not spoken, they will be given the option to type in their answer. However, if there are errors in the typed answer, the word is not recognised, or nothing is entered, then the spellings are common words such as ‘is’, ‘for’, and ‘that’.

The score is tracked as the game is played and the user can view it. They will also be told if their spelling is correct or incorrect once they have entered their spelling. The program accepts either capital or lowercase letters and ignores spaces at both the beginning and end of the word. It does not ignore spaces in the middle of a word.

There is a different word each time because once a word had been used and spelt, it is added to an array of words that are excluded from finding the next closest word. This array includes the theme so that the user is not asked how to spell the theme if they typed in their choice of theme.

Once the game has been completed, the final score from the game will be announced. There are also comments depending on how well the user performed in the game.

*Testing:*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test: | Purpose: | Input: | Type: | Result: |
| 1 | To see if the speech recognition works | Say the theme | Valid | The theme is picked |
| 2 | To ensure the program does not crash if speech recognition is not used | Do not speak | Invalid | The user is asked to type in their theme. |
| 3 | So that the user can type in their chosen theme | Do not speak then type in a theme | Valid | The theme is picked |
| 4 | To see what makes the words random | Enter nothing into the theme – no presence check | Invalid | Random words |
| 5 |  | Enter ‘animals’ as the theme – correct spelling | Valid | The words are to do with the theme |
| 6 |  | Enter ‘aminals’ as the theme – incorrect spelling | Invalid | Random words |
| 7 |  | Enter ‘dhfsojdbf’ as the theme – nonsense | Invalid | Random words |
| 8 |  | Enter ‘aquatic animals’ as the theme – two or more words | Invalid | Random words |
| 9 | Checking that it marks the spellings correctly for the word ‘animals’ | Enter ‘animals’ as the spelling – correct spelling | Valid | Score increases by 1  Outputs ‘Correct! Well done.’ |
| 10 |  | Enter ‘aminals’ as the spelling – incorrect spelling | Invalid | Output ‘Oh no! That was incorrect.’ |
| 11 |  | Enter ‘ANIMALS’ as the spelling – with capital letters | Valid | Score increases by 1  Outputs ‘Correct! Well done.’ |
| 12 |  | Enter ‘ animals’ as the spelling - with spaces | Valid | Score increases by 1  Outputs ‘Correct! Well done.’ |
| 13 |  | Enter ‘animals ’ as the spelling – with spaces | Valid | Score increases by 1  Outputs ‘Correct! Well done.’ |
| 14 |  | Enter ‘ani mals’ as the spelling – with spaces in the middle | Invalid | Output ‘Oh no! That was incorrect.’ |

*Variables:*

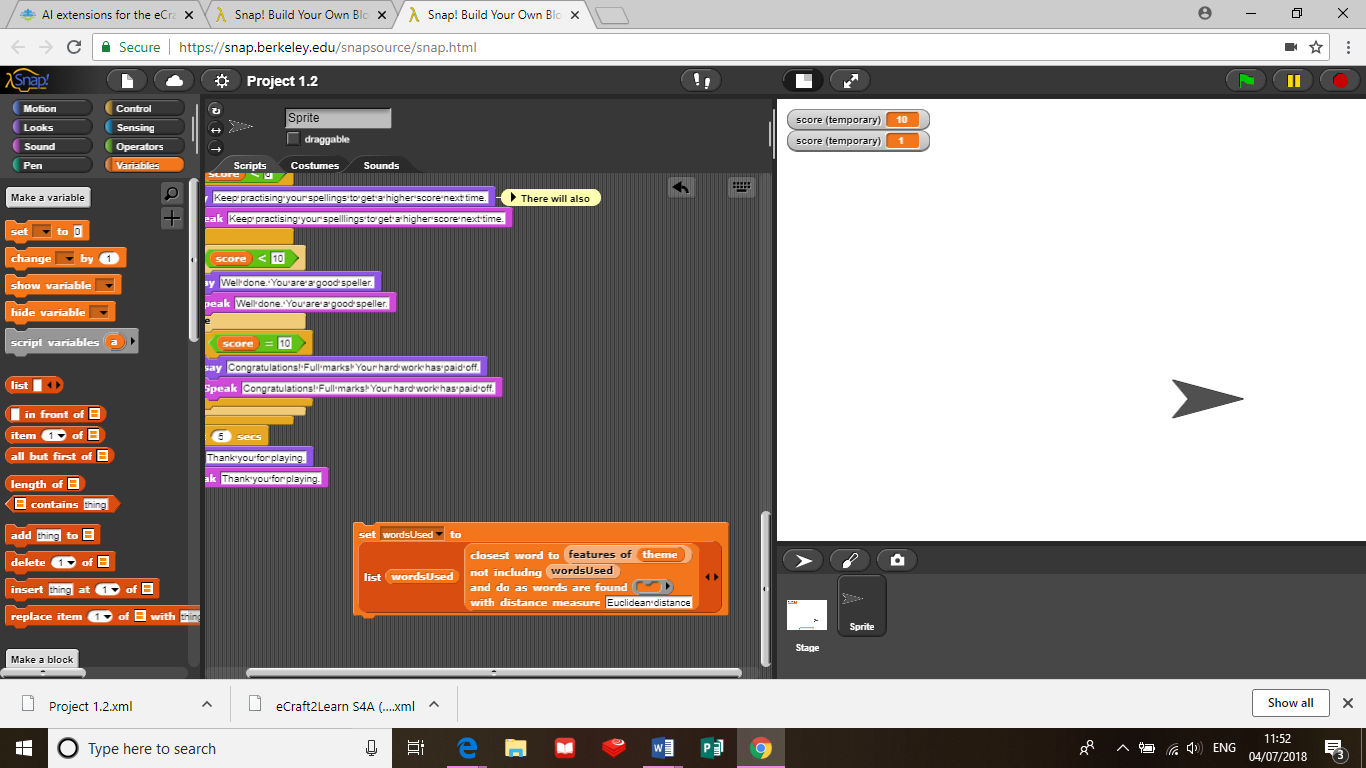
|  |  |  |
| --- | --- | --- |
| Name: | Type: | Use: |
| theme | string | This holds the theme chosen by the user that is either recognised by speech recognition or entered by the user. |
| score | integer | This holds the user’s score during the game. Every time a word is spelt correctly the score is increased by 1. |
| wordsUsed | string array | This holds the words that the user has been tested on so they can be excluded when trying to find the next word to be spelt. It holds the theme so they are not tested on the how to spell the theme they entered, and then after each round the spelling is added so it is not repeated. |
| i | integer | This is part of the ‘for’ loop. It controls how many spellings the user will be tested on. |

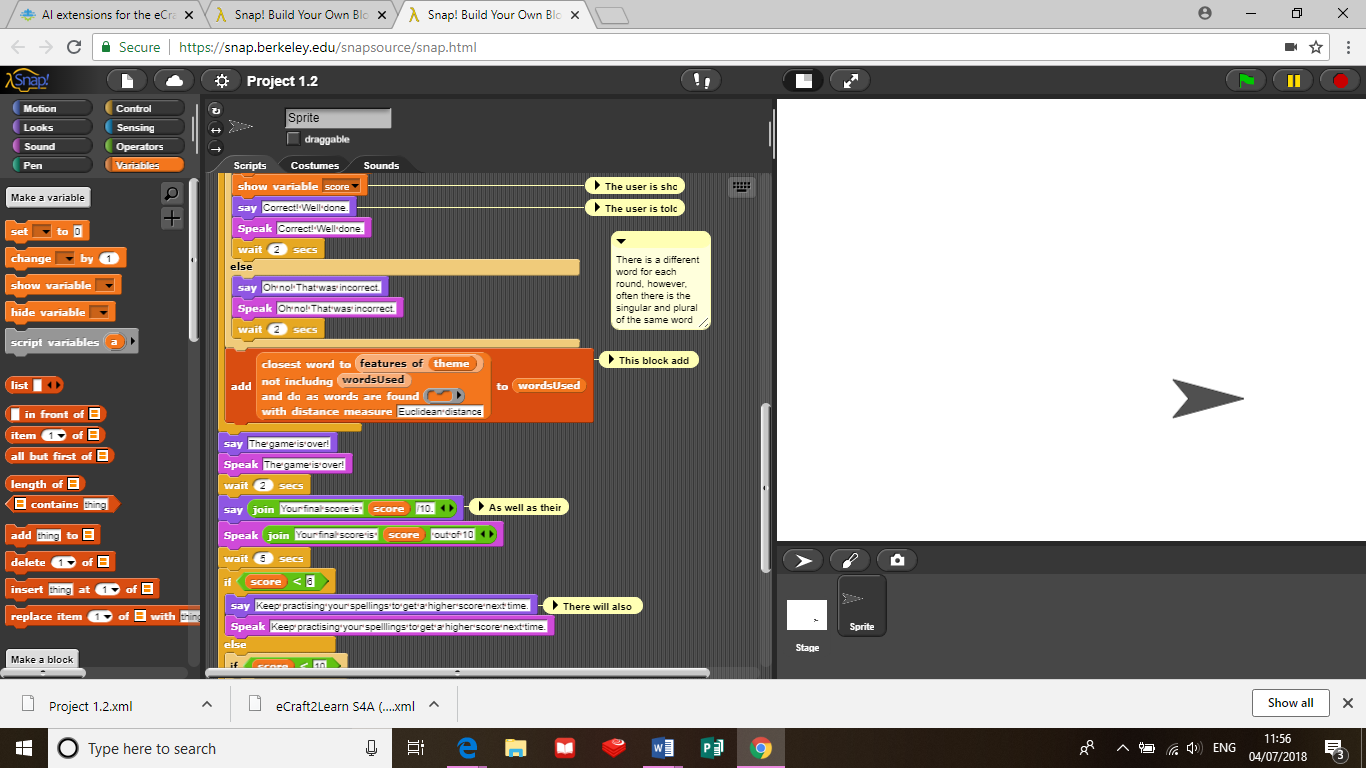
*Validation:*

There is very little validation needed for this program because the spellings ignore capitals and spaces. When the user is choosing a theme, if there is an error with the speech recognition, the user can type it in. There is no validation for the typed in option because the program does not crash if nothing is entered do there is no need for a presence check. The program runs if the theme is spelt incorrectly as well. However, there will be spellings such as ‘the’, ‘this’, and ‘as’ which has nothing to do with the theme. If there are spaces in the theme, this will also happen, because it can only compare single words.

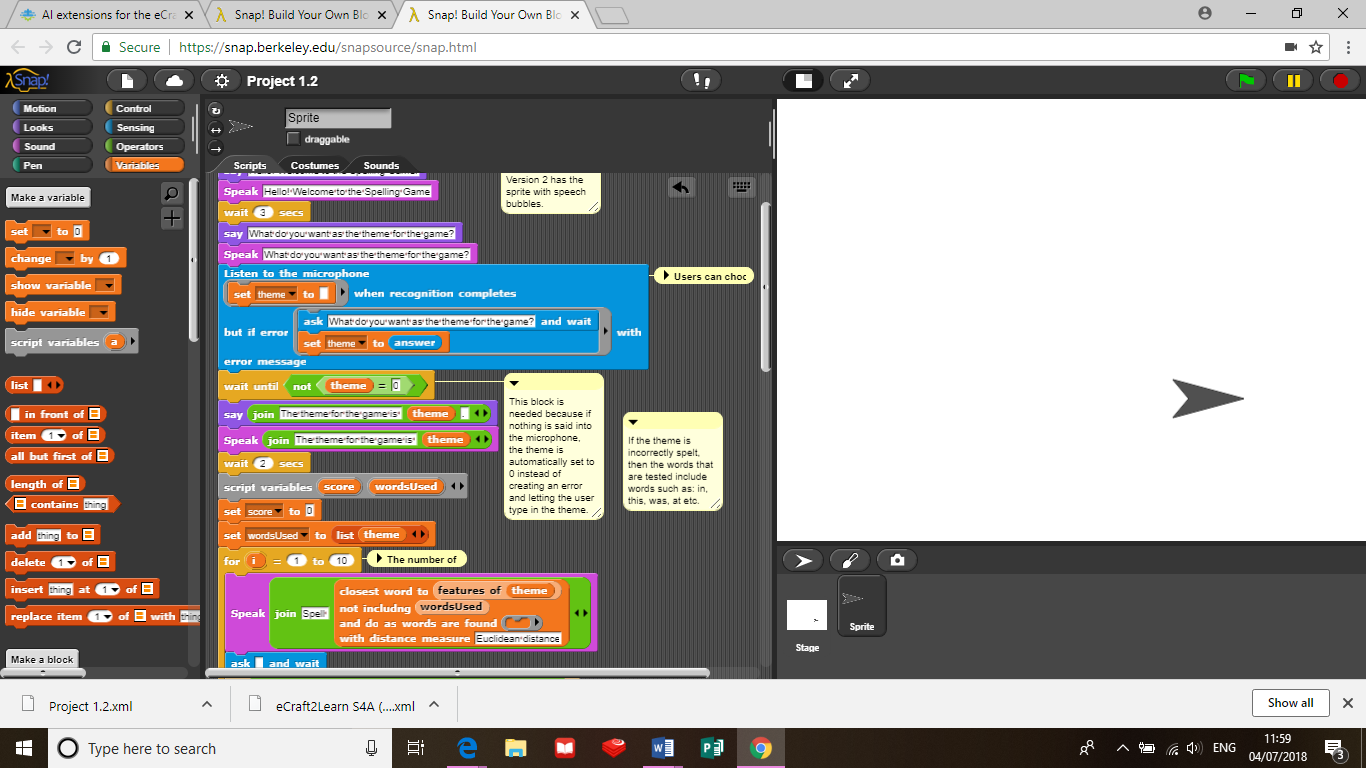
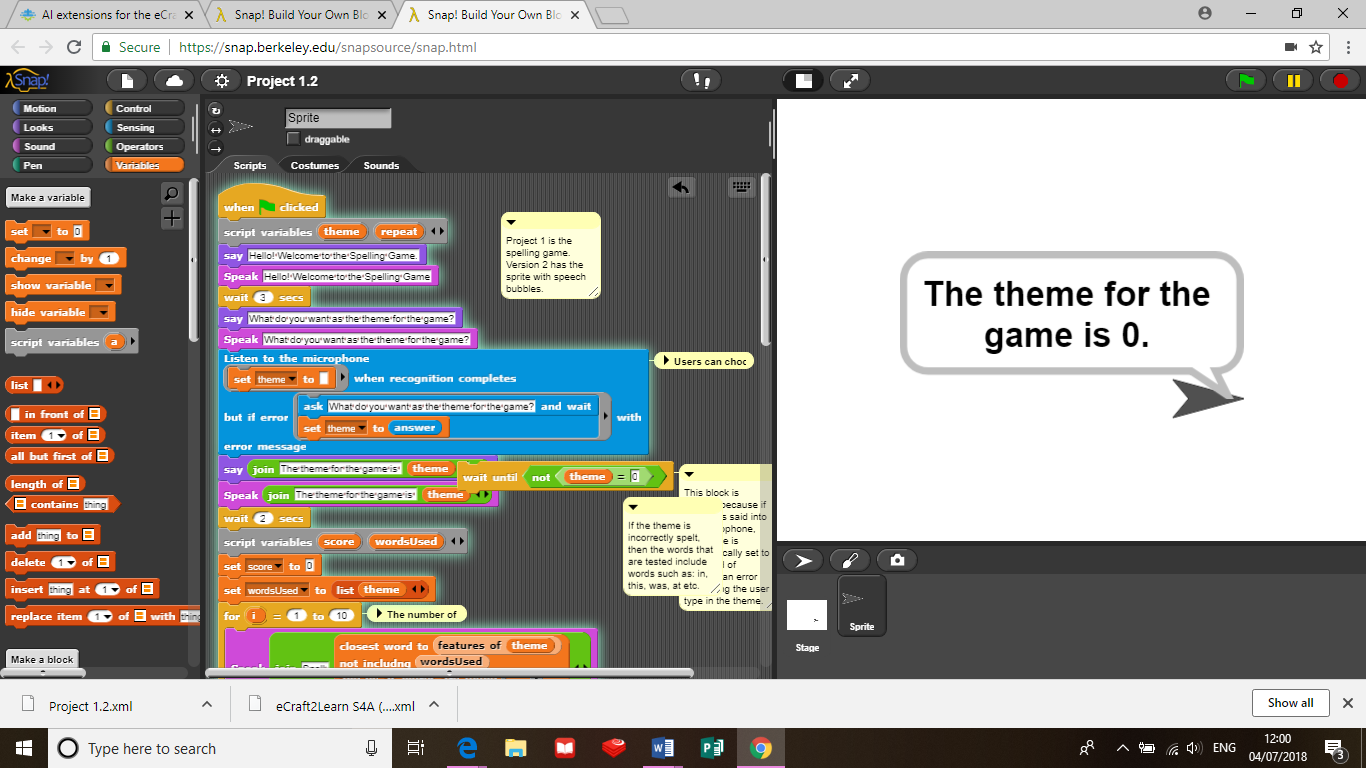
*Development:*

Originally, I tried to create a new array to exclude more words from finding the next closest word so that there were always different spellings.

However, this caused the theme and first closest word to the theme to repeat.



I then saw a block which could add to an array. This worked, so there is a different spelling each time.

I also discovered that this block is needed because if nothing is said into the microphone, the theme is automatically set to 0 instead of creating an error and letting the user type in the theme.

*Evaluation:*

I think that this program works as a spelling game because it asks the user aurally to spell different words, telling them whether their spellings are correct or incorrect and gives them a final score at the end. What does not really work is the theme. This is because some word connections are unusual and it is difficult to validate the words being typed in, although the program does not crash if the theme is not entered or is spelt incorrectly.

*Possible improvements:*

* Ask the user if they want to play again.
* Tighten the validation on the theme.
* Split the program into more manageable subroutines.
* Do not allow both singular and plural versions of words.
* Output a list of words at the end of the game that the user needs to practise, showing their incorrect spelling and comparing it to the correct spelling.
* Change the display and costume of the sprite to make the game more fun and interesting.
* Change the accent of the voice to British.