Criterion B: Solution Overview

Justification of Methodology

I chose the design methodology of an incremental prototyping model in the System Development Life Cycle (SDLC), since the entire solution can be easily broken down into smaller stages or processes that are relatively independent. The smaller stages are identified as:

- Backend:
    1. Creating data model with Java classes
    2. Constructing a database(collection) class that can transfer the data from the permanent storage (text files) into the program.
    3. Constructing a workshop class that can create, read, update, and delete (CRUD) data from the permanent storage.
    4. Constructing a class that include the learning exercises (remember and check; guess and type and multiple choice).

- Frontend GUI interacting with the backend:
    1. Home page
    2. Word collections
    3. Collection selection
    4. Creation page
        - New collection
        - New word
    5. View/edit page
        - View collection
        - Edit collection
        - View word
        - Edit word
    6. “Remember and check” exercise
    7. “Guess and type” exercise
    8. “Multiple choice” exercise
    9. Reverse “remember and check” exercise
    10. Reverse “guess and type” exercise
    11. Reverse “multiple choice” exercise

As the first increment of the incremental model, backend is designed, programmed, and tested as follows:

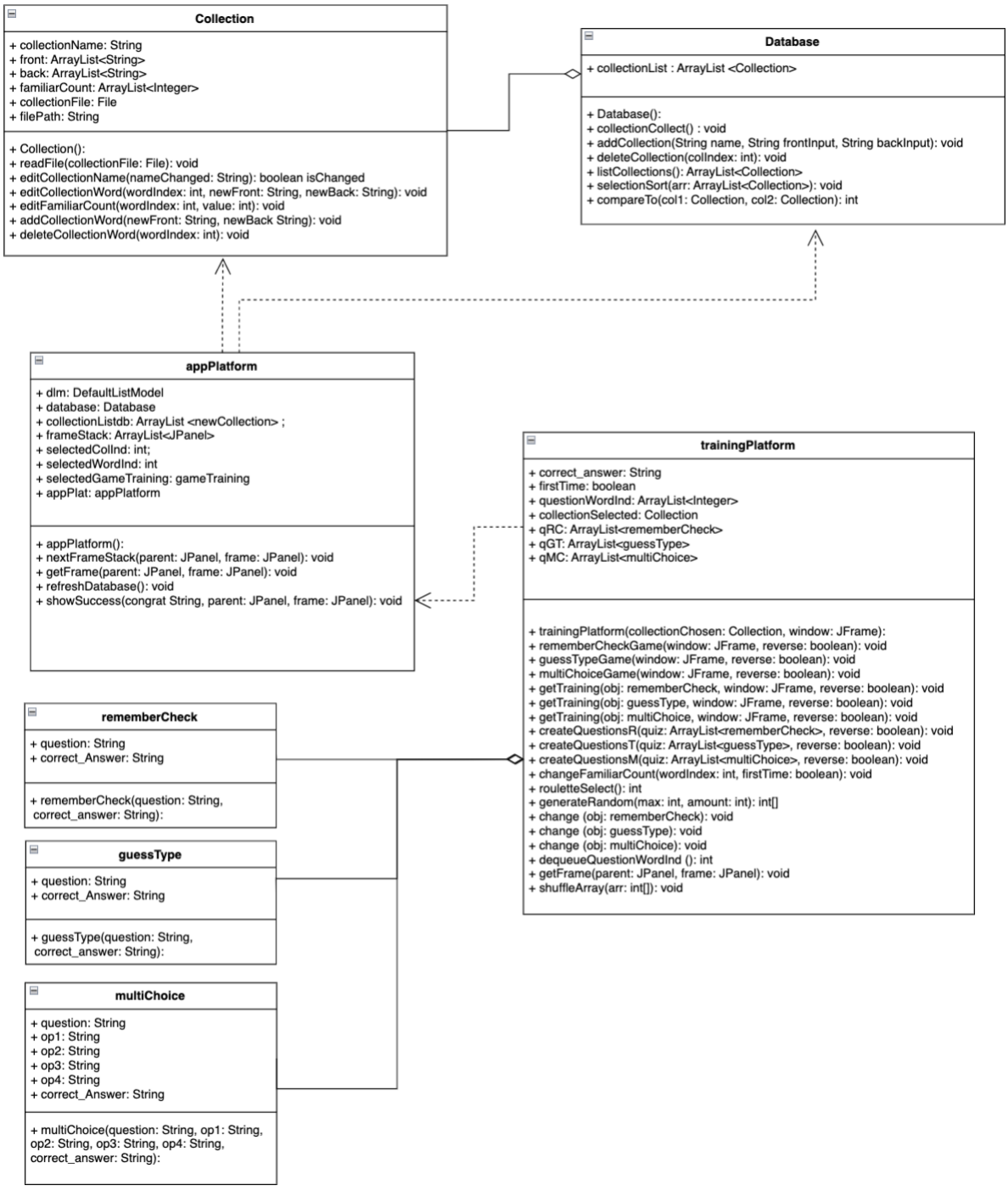
1. Analyze the client’s requirement of the program
2. Identify the functionalities that the backend of the program should achieve
3. Design the back end of the program with UML class diagram
4. Code the program for each stage of the backend
5. Test the code for each stage of the backend

Then, for each of the frontend stages, the following steps are carried out repeatedly:

1. Refer to the drafted prototypes
2. Identify the visuals and functionalities that the graphical interface should incorporate
3. Design the visual and graphical components on the user interface
4. Design the processes performed on the user interface with a flowchart
5. Ask for client's feedback and improve the design as needed
6. Program the user interface based on the design
7. Test functionalities of the stage according to the test plan

After the steps for each individual stage are completed, the stages are integrated, and the entire program is tested and debugged as the final increment.

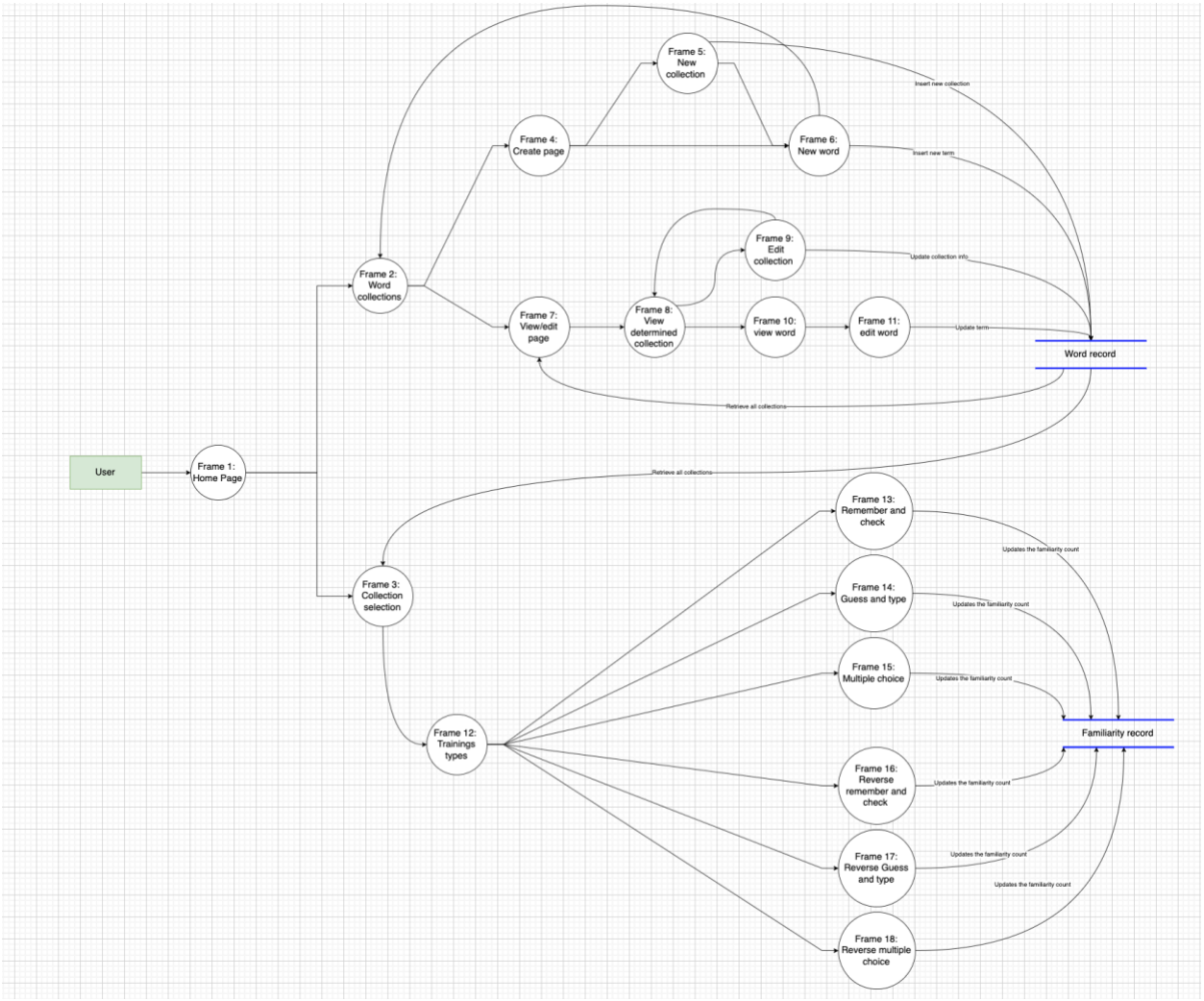
UML Class diagram:



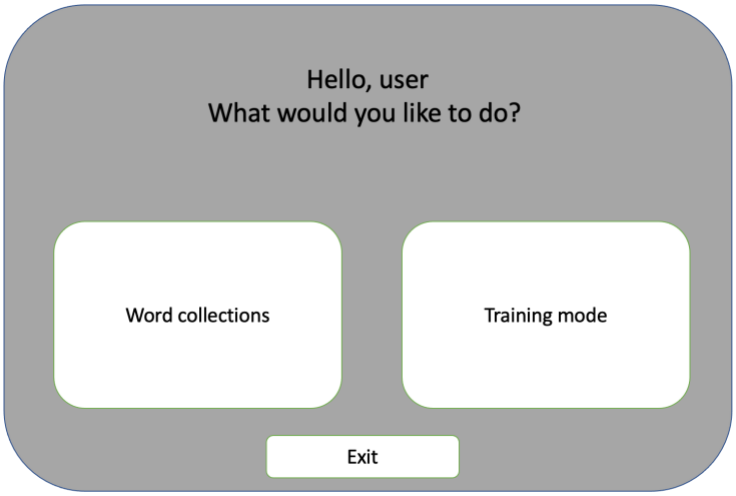
The methods and variables used are public as different classes need to access them. Besides, since each student has their own system and word lists, security issues are minimized and private elements are not often used.

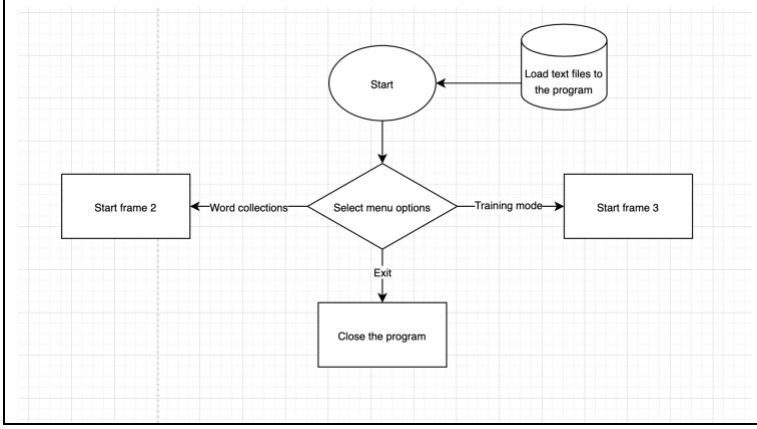
Data flow diagram:

The program's frontend was built to have multiple independent frames, each doing a different process. The data flow diagram shows how data is transferred between different processes and how the frames interact with the backend databases in general.

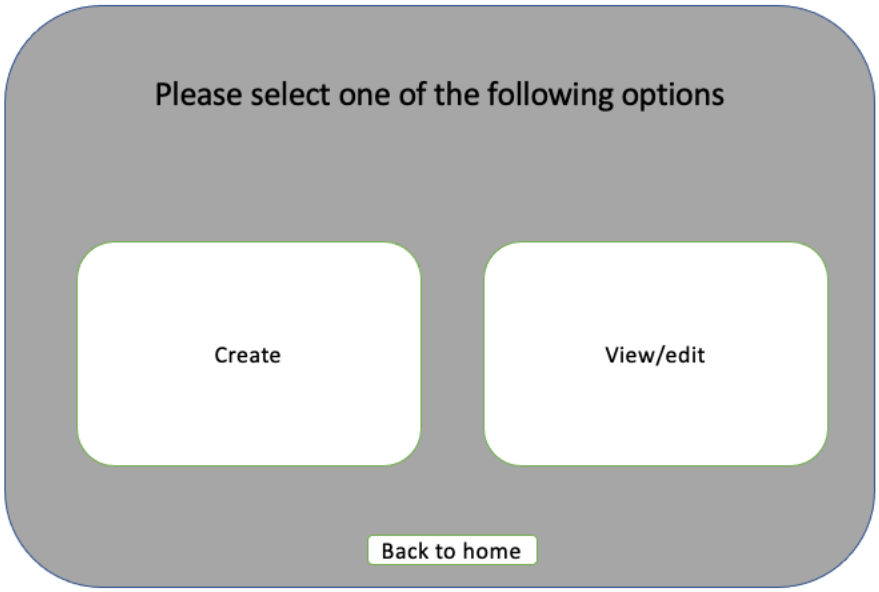


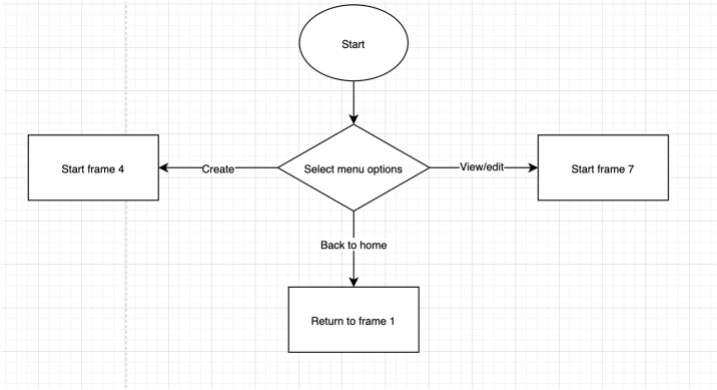
Frame 1: Homepage



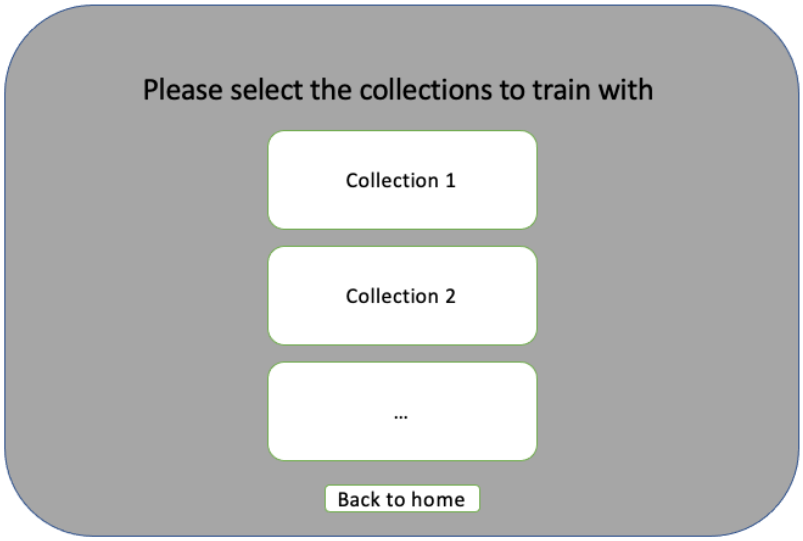


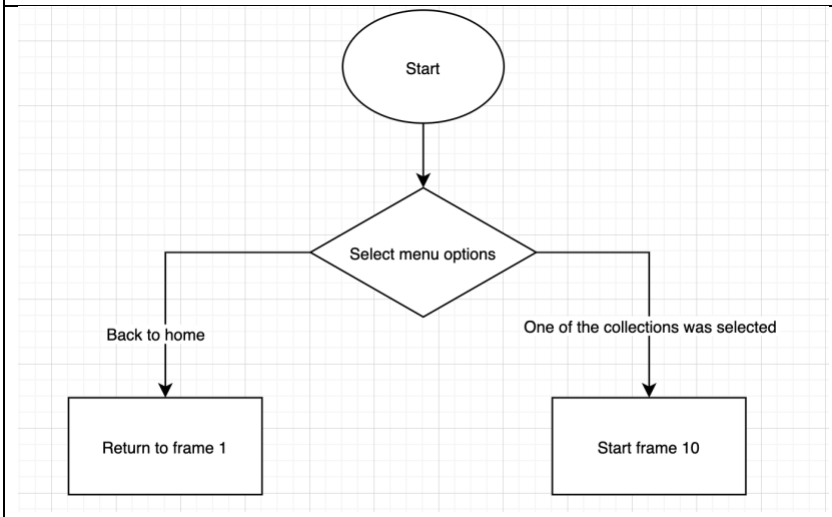
Frame 2: Word collection



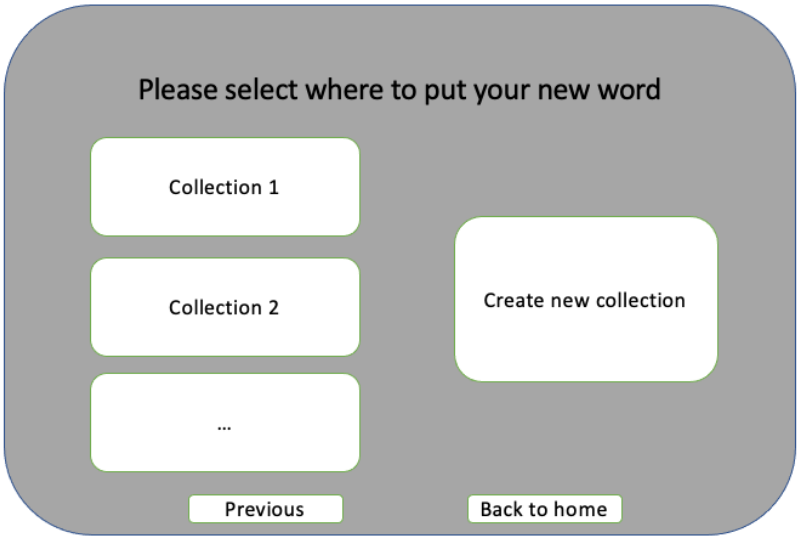


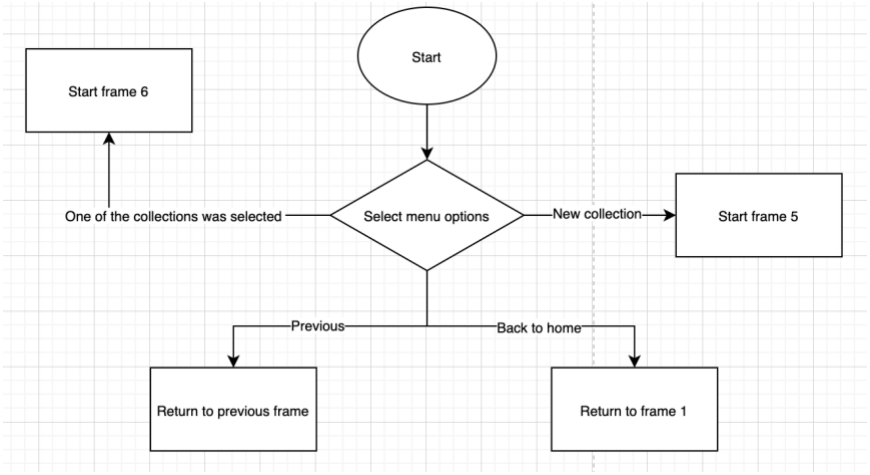
Frame 3: Collection selection



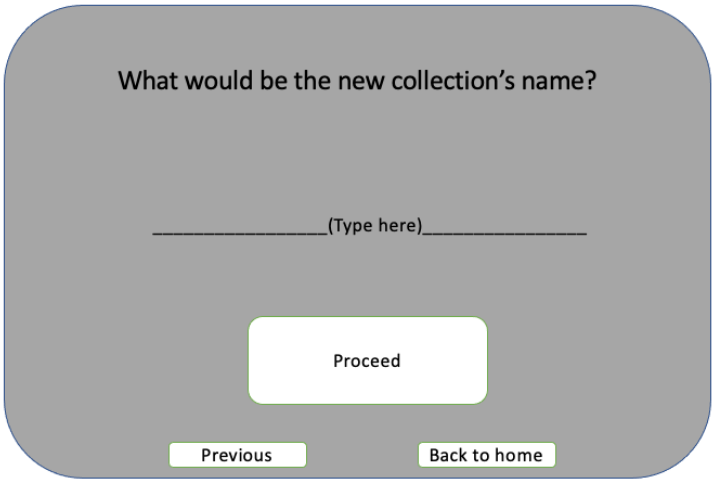


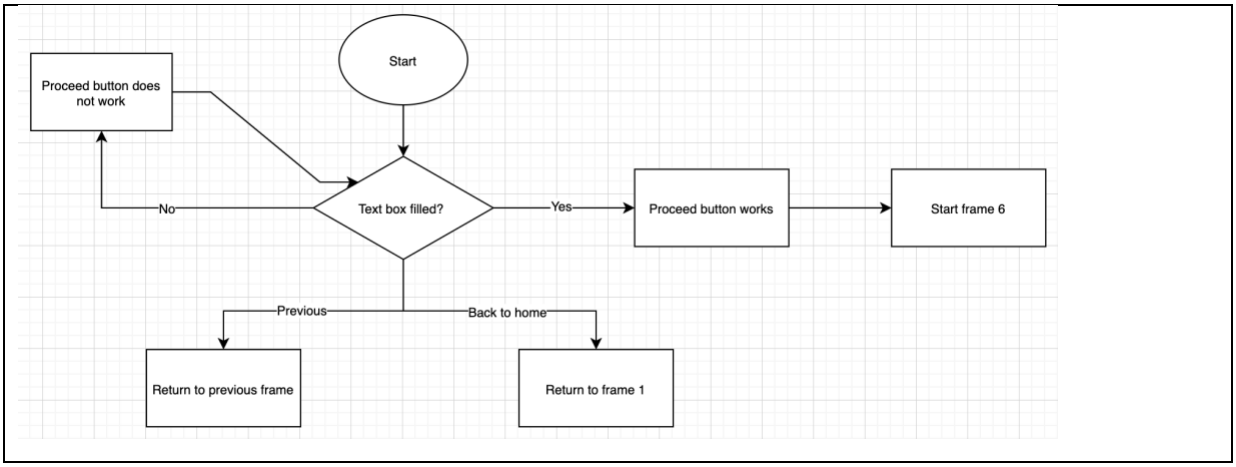
Frame 4: Create page



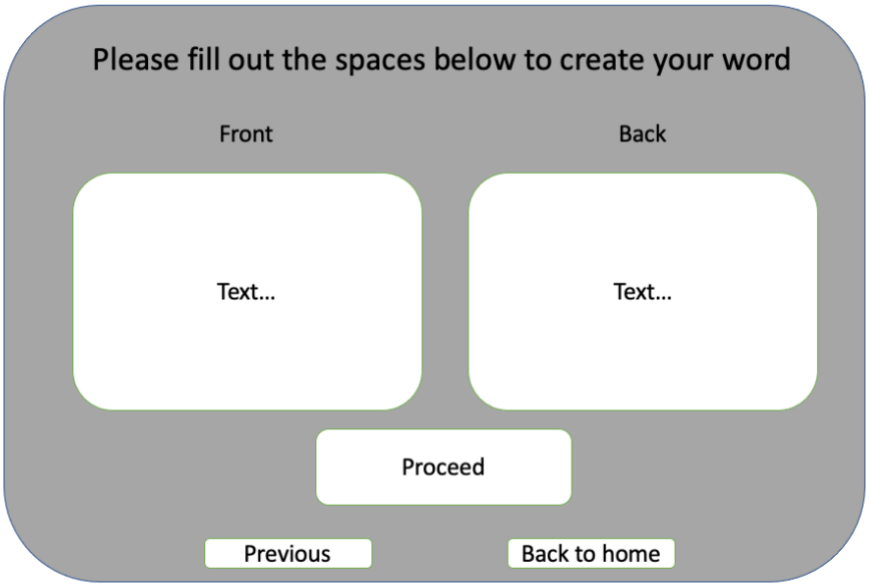


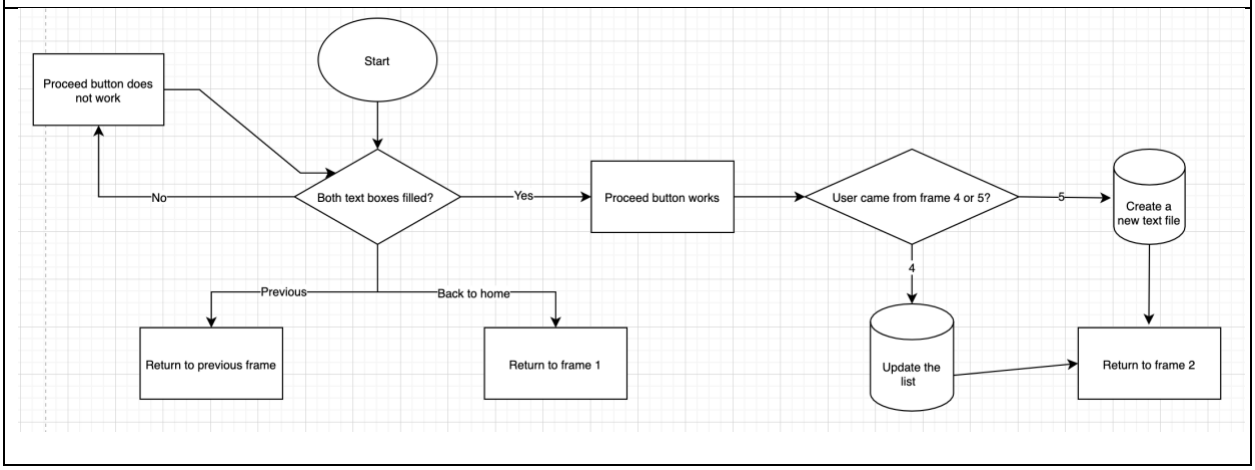
Frame 5: New collection



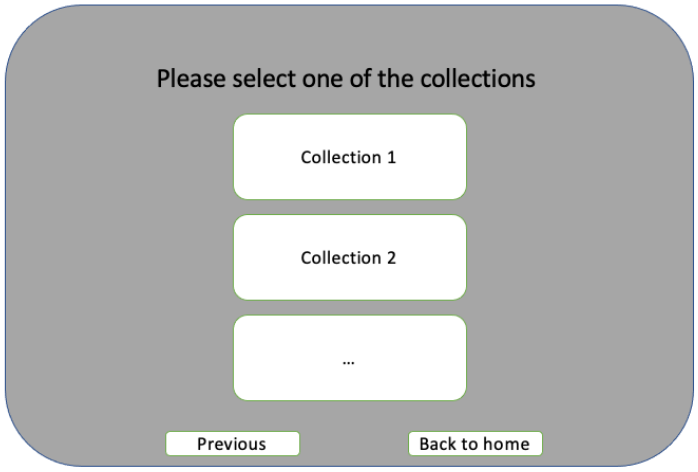


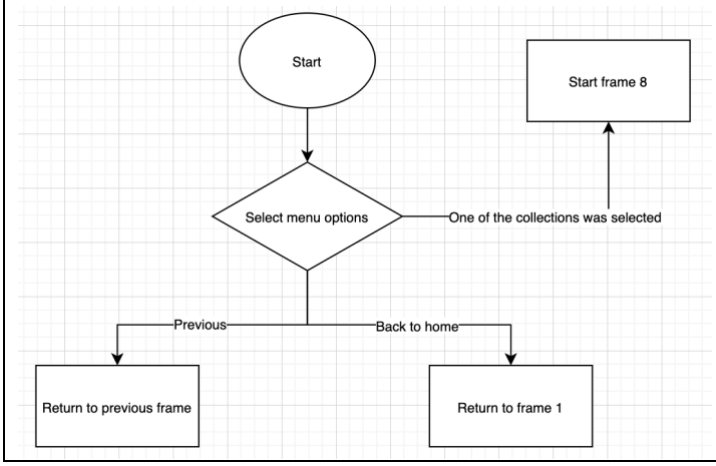
Frame 6: New word



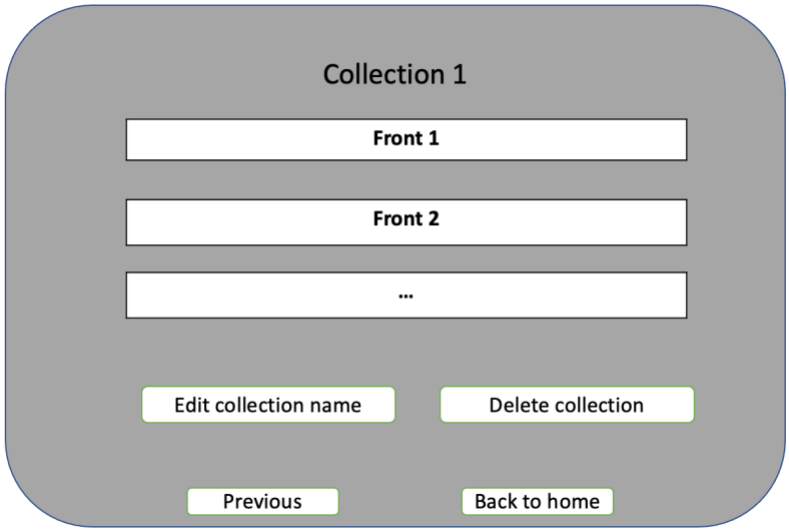


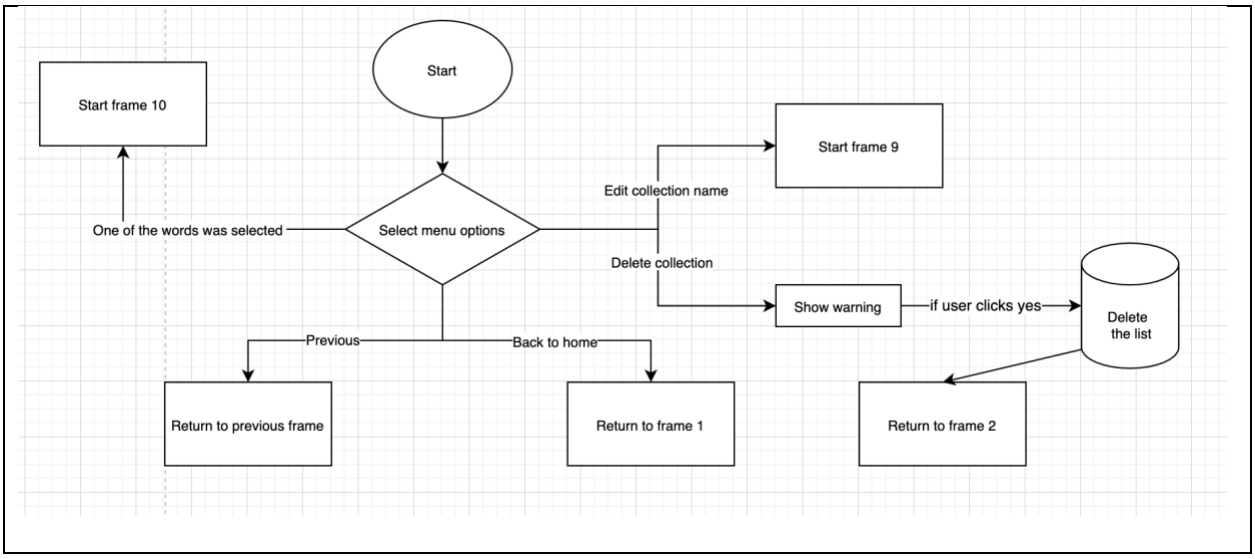
Frame 7: View/edit page



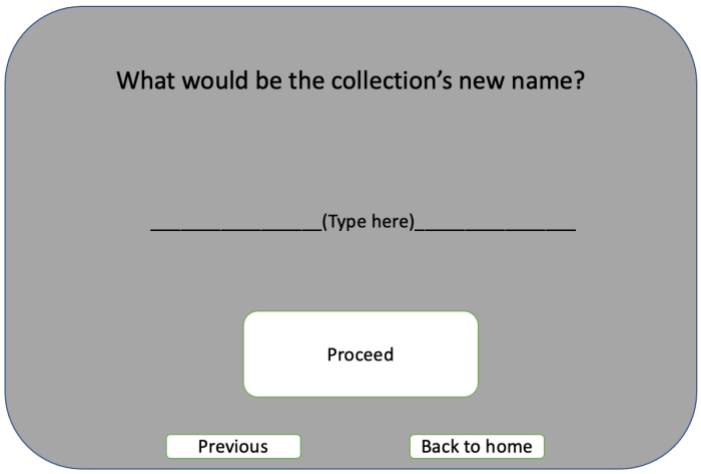


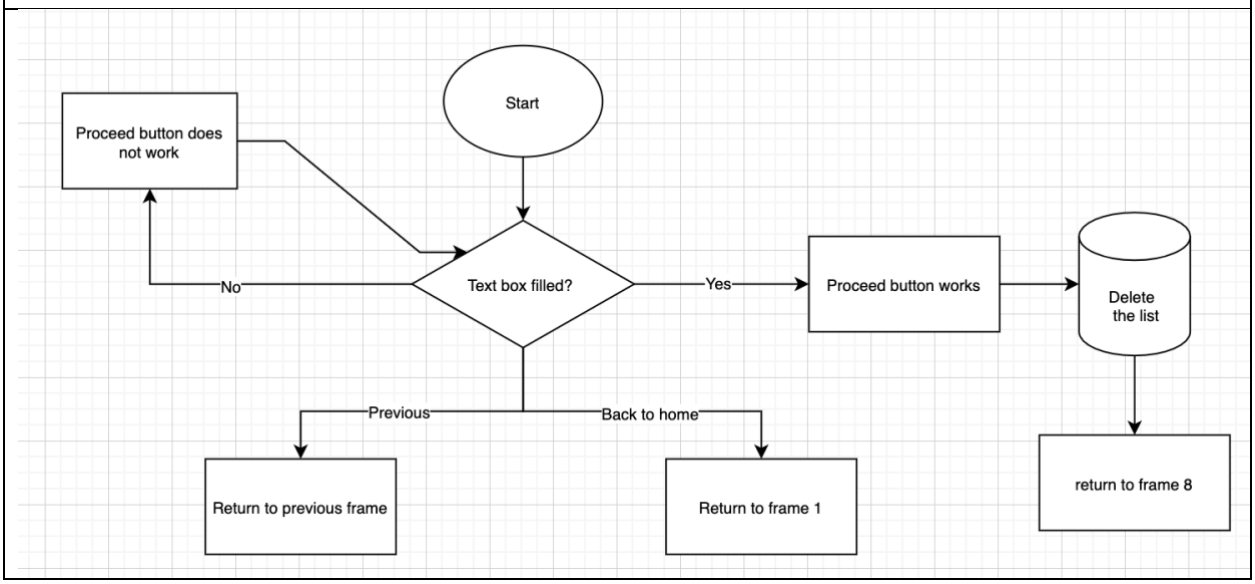
Frame 8: Collection view



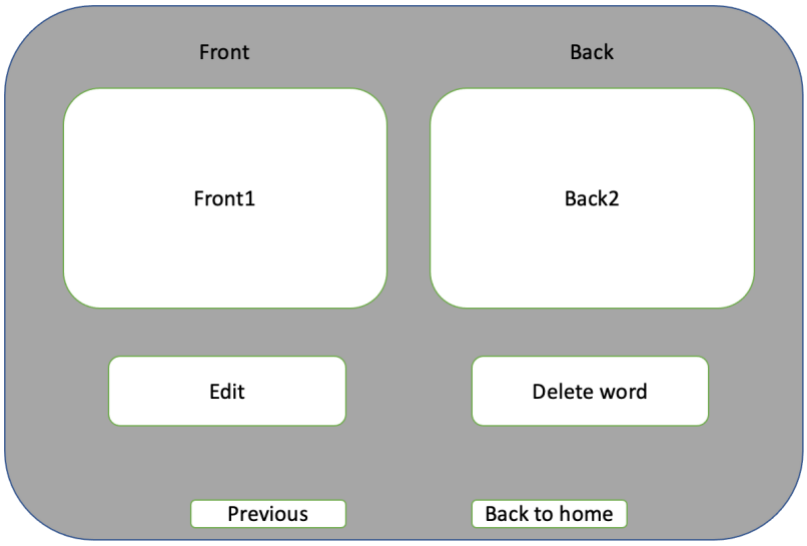


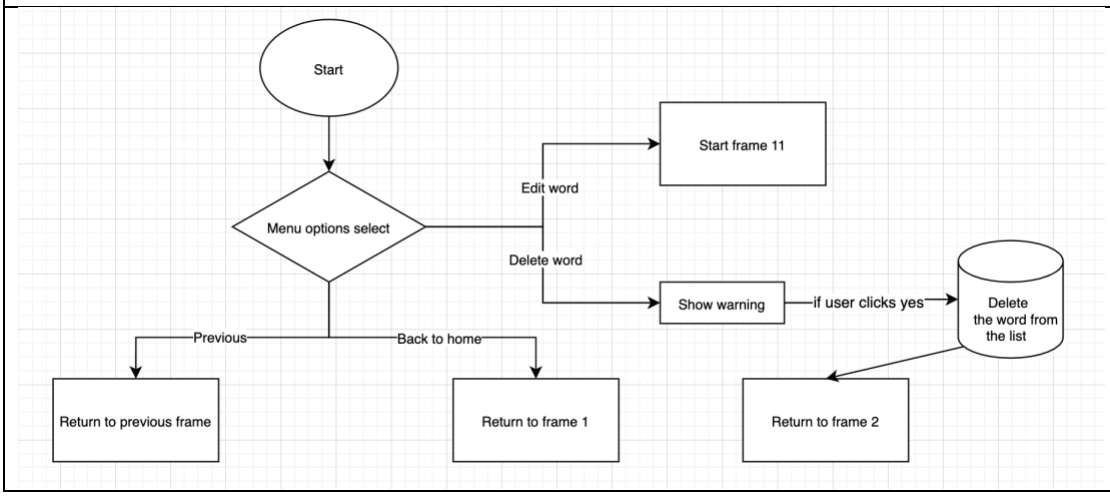
Frame 9: Edit collection name



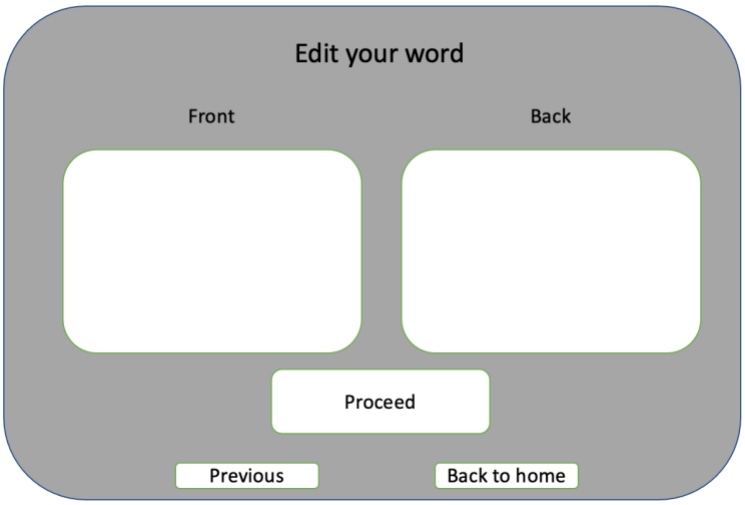


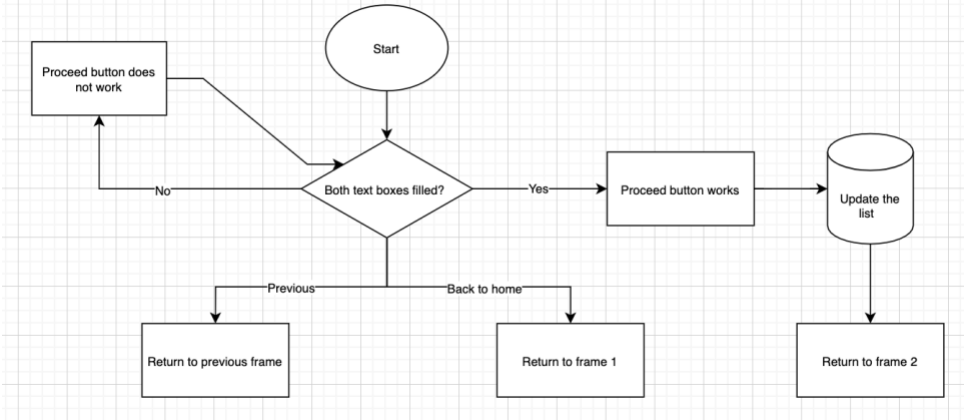
Frame 10: View word



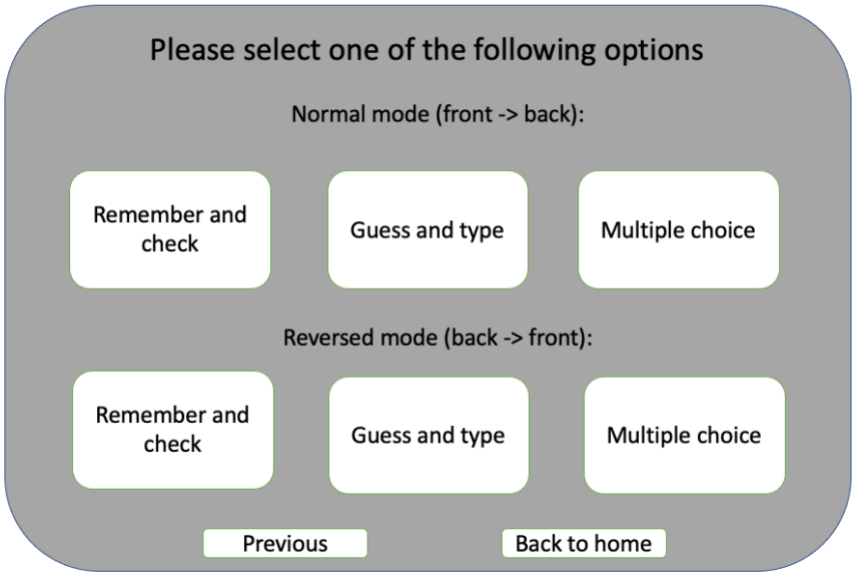


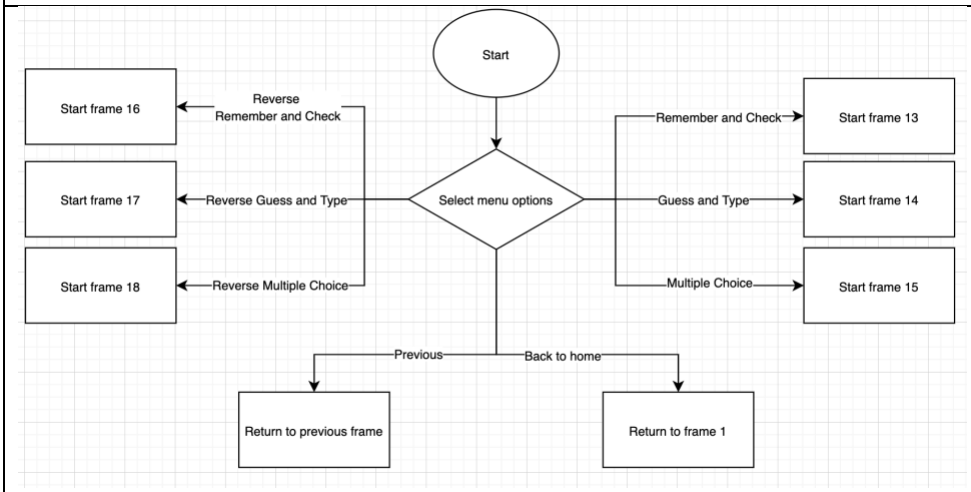
Frame 11: Edit word



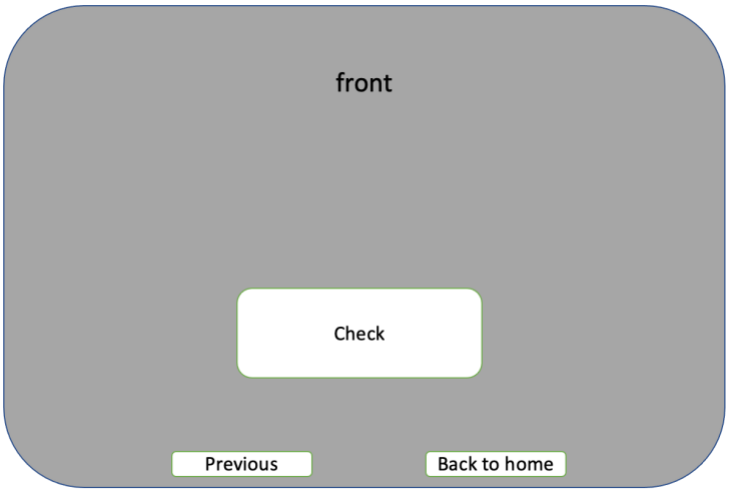


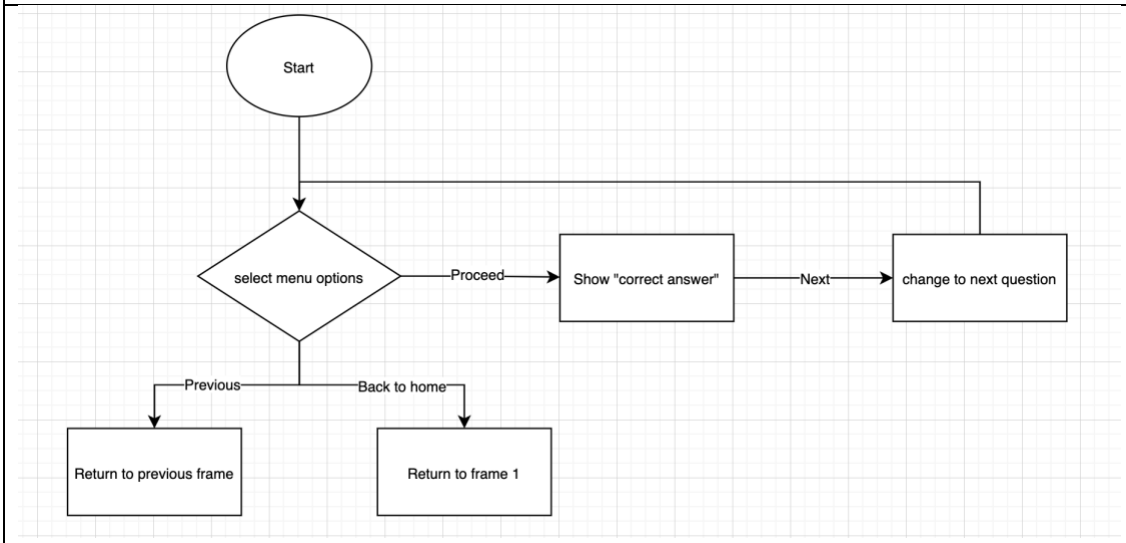
Frame 12: Training types



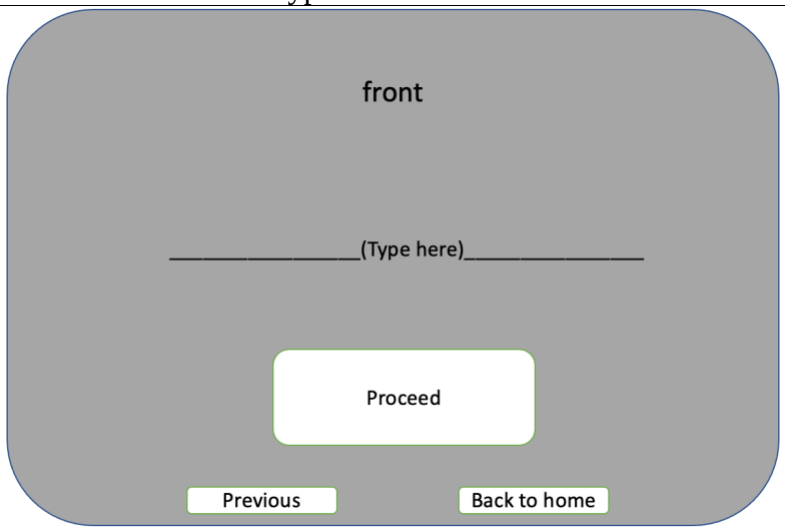


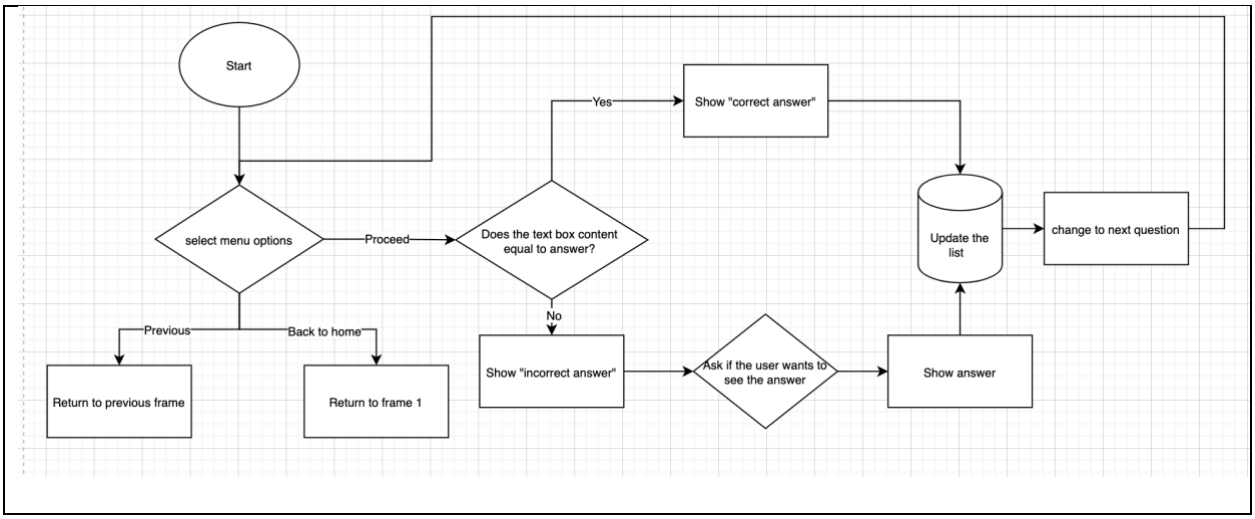
Frame 13: Remember and check



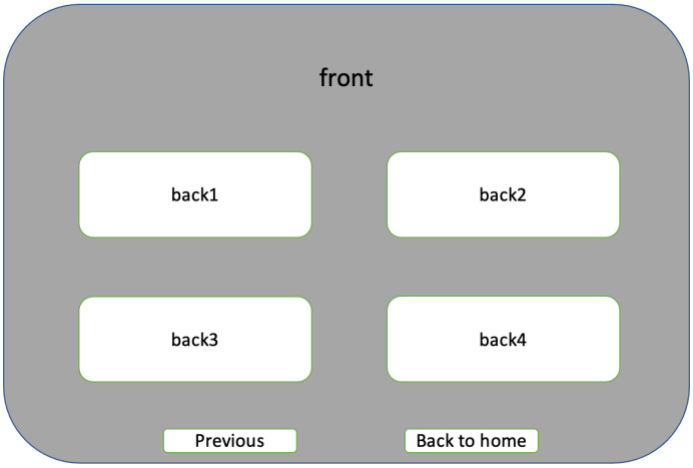


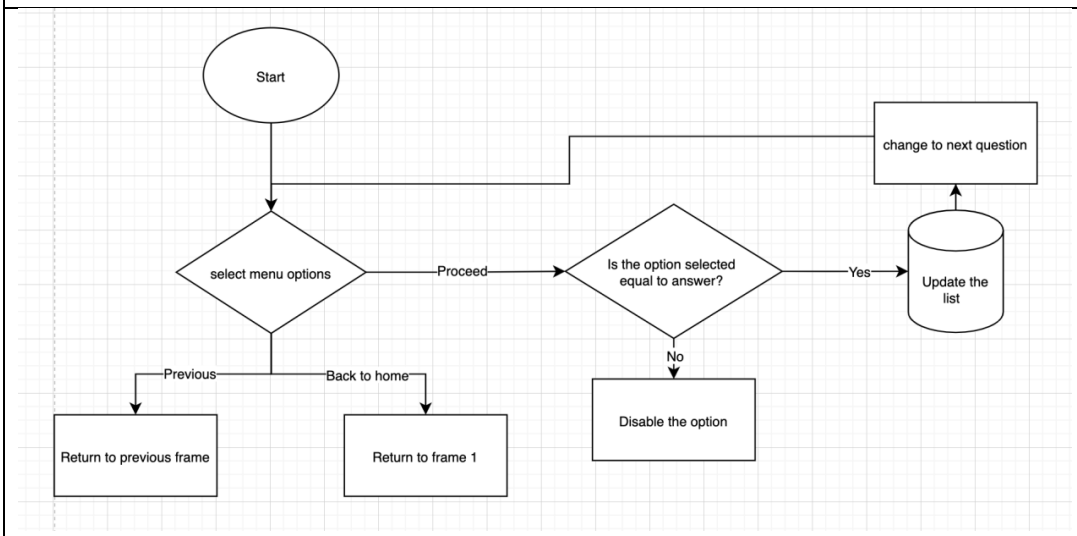
Frame 14: Guess and type



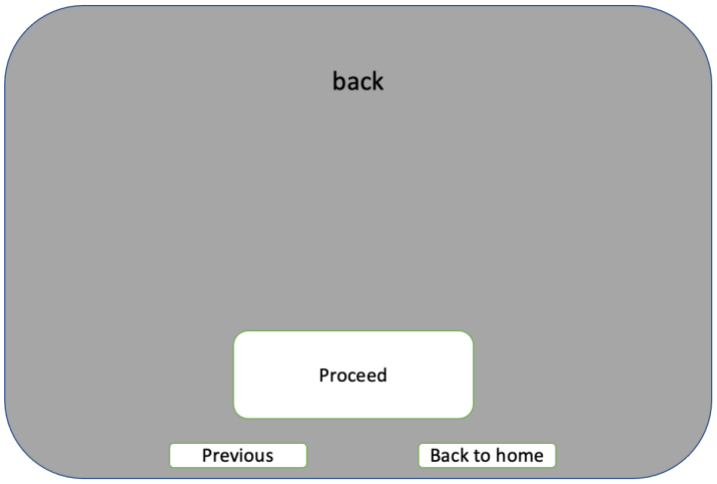


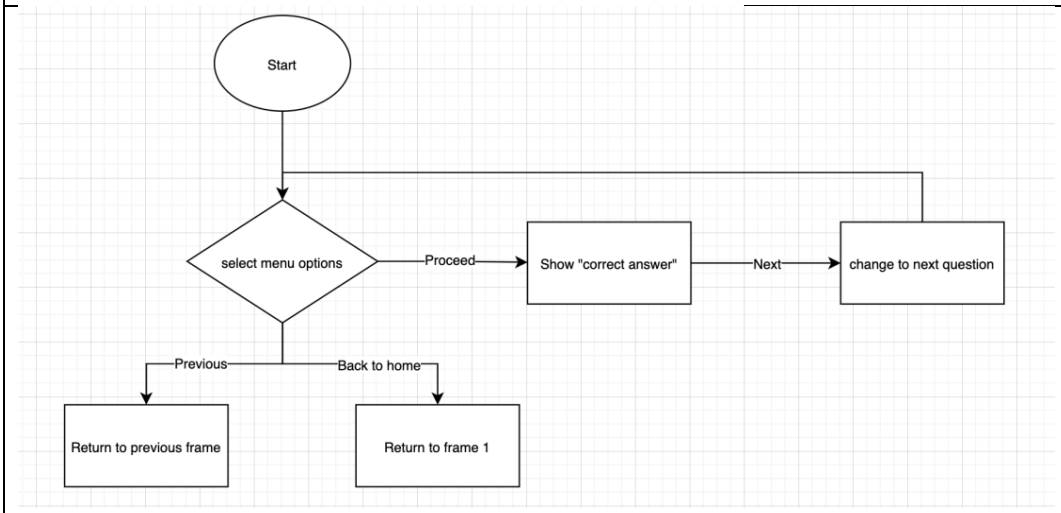
Frame 15: Multiple choice



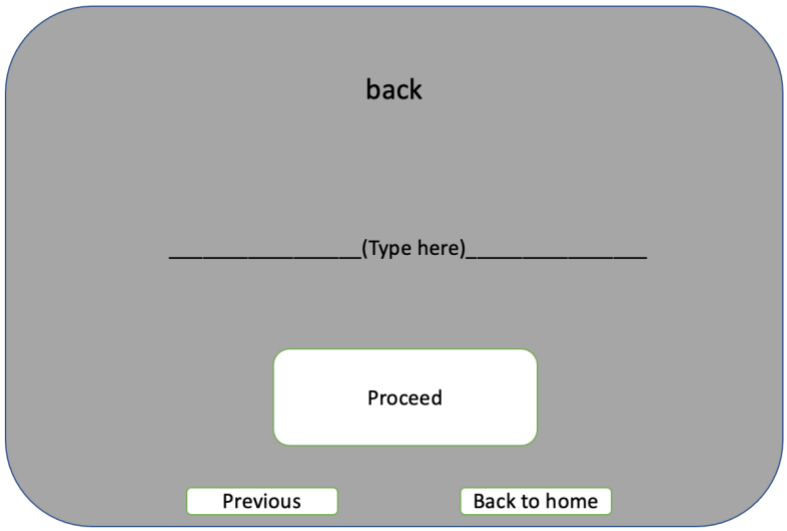


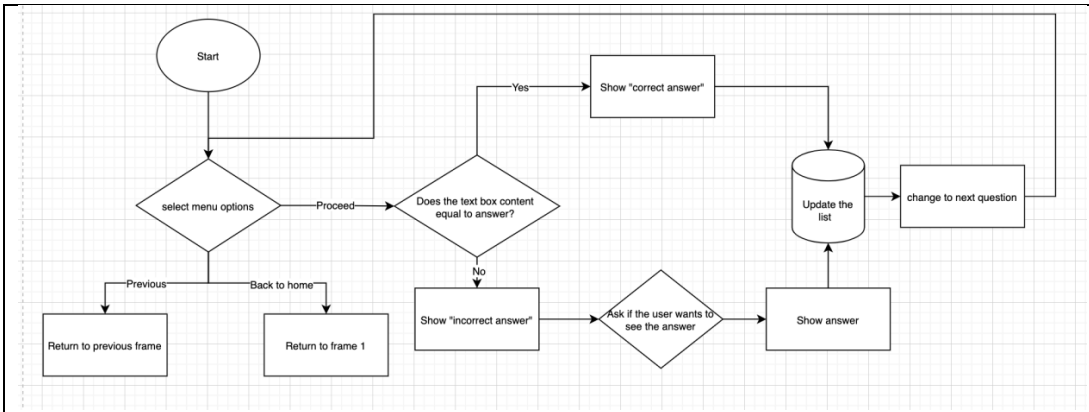
Frame 16: Reverse remember and check



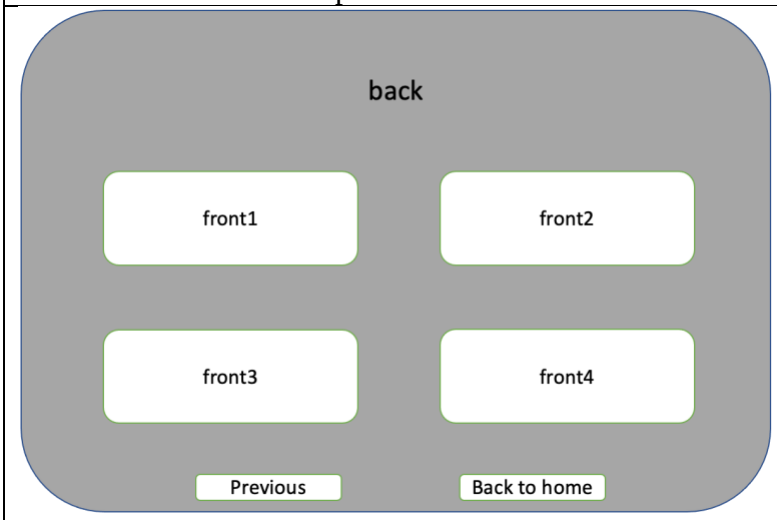


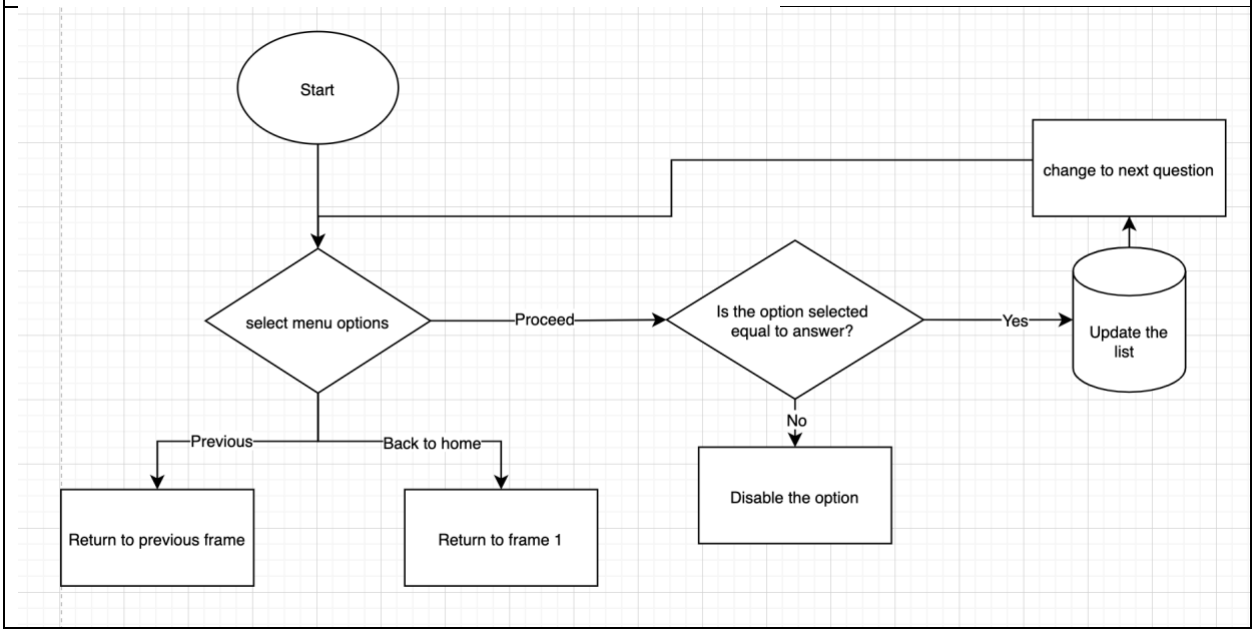
Frame 17: Reverse guess and type





Frame 18: Reverse multiple choice





Test plan:

Success criterion	Testing process	Aim
1	Input any character, including with accents, into the field and see if it gets shown in the text file and in the program	Test if the program can accept any word
1	Check if the user cannot proceed in the word creation without filling the “front” AND “back” of the word	Ensure the program is not creating words without definitions or vice versa
2	After creating a new word by clicking on “new collection”, see if a new text file is created.	See if the storage system works
3	Create another word but by adding a new word onto an already created collection and see if it gets shown in the text file	Test if the program can also add words without creating txt files.
4	Check if there are the number of words shown in the program is the same as the number of words shown in the text file	Ensure that every word in the text file is read and converted
5	Go to the view/edit page and view a word and check if it is the same as shown in the text file	Ensure that the user can see any word that they have created completely and identically
6	Click on “edit” button on the view panel and check if the word was changed in the text file	Test if the program can edit words from the text files.
7	Click on “delete” button on the view panel and check if the word was deleted in the text file	Test if the program can delete words from the text files.
8	Click on “edit collection name” button on the “word list” panel and check if the collection’s name was changed in the text file	Test if the program can edit the text files’ names.
9	Click on “delete collection” button on the “word list” panel and check if the text file is deleted	Test if the program can delete text files in case the user wants to.
10	Go to training mode and see if there are existing collections	See if there are already lists for students to practice
11	Check if the normal exercise shows the “front” and hides the “back”	See if the exercise works as intentioned
12	Type an incorrectly typed word and see if it is not accepted	See if the system can correctly recognize correct words
12	See if the program asks if the user wants to see the answer if he answered incorrectly	Ensure the user doesn’t get stuck in one of the exercises

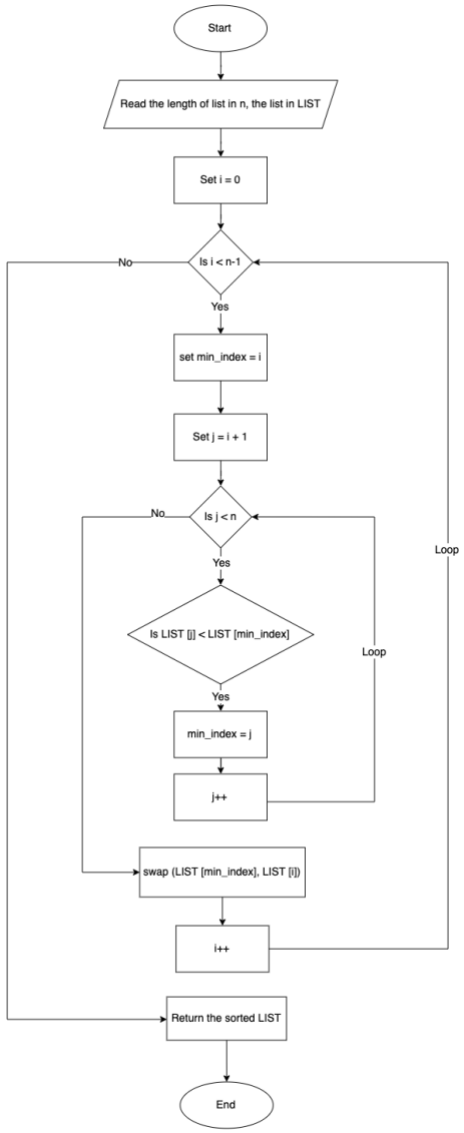
13	Check if the “Guess and type” and “Multiple choice” exercises changes the familiarity count in the text files	This is essential so that the less familiar words gets picked more often
14	Check if the user can enter if the collection has less than 4 words	Ensure that system can only work under the predetermined condition
14	Check that no options are identical	Ensure the learning process is challenging
15	See if the word with a higher familiarity count gets shown more often	Ensure that the student is “learning from their errors”
16	Go to any normal exercise and check if the exercise asks the user for the “back”	See if the normal exercises work as intentioned
16	Go to the same exercise but in reversed mode and check if the exercise asks the user for the “front” instead	Ensure the reversed training system is “back to front”

Explanation of key algorithms

Selection sort

<table>
<tr><td colspan="2">Function: sort a list of objects with average time complexity of $O(n^2)$ by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning (“Selection sort”).</td></tr>
<tr><td colspan="2">Usage in the program: sorting collections alphabetically.</td></tr>
<tr><td colspan="2">Justification: Selection sort is one of the two sorting algorithms that I have learned in class, and it is faster than bubble sort as it does less writings. I have chosen the algorithm as I am familiar with its mechanics and the number of objects to be sorted is relatively small.</td></tr>
<tr><td>Pseudocode (“Selection sort”).</td><td>Flowchart of the algorithm</td></tr>
<tr><td>

```
// LIST is a Collection

// Int min_index

// n is the length/size of LIST

 Loop i from 0 to n - 1
    min_index = i
    Loop j from i+1 to n
        if list[j] < list[min_index] then
           min_index = j
        end if
    end loop

     if min_index != i  then
        swap (LIST[min_index], LIST[i])
     end if
 end loop

return LIST
```

</td><td> 
</td></tr>
</table>

Roulette wheel selection

Function: a type of Fitness Proportionate Selection, where every individual can be selected with a probability that is proportional to its fitness. Therefore, “fitter” individuals have a higher chance of being chosen. (“Genetic Algorithms - Parent Selection.”).

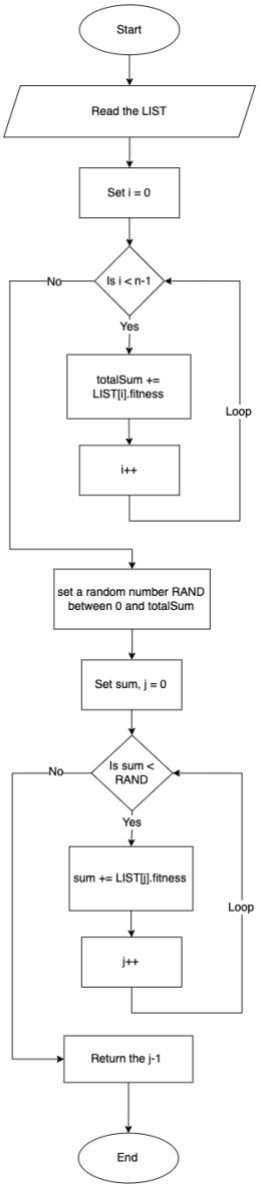
Usage in the program: selecting less familiar (higher fitness) words to train with in the training section

Justification: With Mrs. Morey’s statement on “learning from their errors”, the algorithm allows the program to select the words which the student has more errors on.

Pseudocode (“Genetic Algorithms - Parent Selection.”)

```
// LIST is a Collection

// Int totalSum, sum = 0

// n is the length/size of LIST

Loop i from 0 to n-1
   totalSum += fitness(familiarity) of the word at
LIST[i]
end loop

// RAND is a random number between 0 and totalSum

Loop while sum < RAND
      Sum += fitness(familiarity) of the word at LIST[j]
      j++
end loop

return j-1
```

Flowchart of the algorithm



Work Cited:

"Genetic Algorithms - Parent Selection." *Tutorialspoint*, https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_parent_selection.htm.

"Selection Sort." *GeeksforGeeks*, 31 Feb. 2014, https://www.geeksforgeeks.org/selection-sort/.

"Stacks - Isaac Computer Science." *Isaac Computer Science*, https://isaaccomputerscience.org/concepts/dsa_datastruct_stack?examBoard=all&stage=all.

Word count: 176