Game of Scramble

This is a two-player game in which Player1 enters a word. The program then scrambles it up, and Player2 needs to rearrange the letters to unscramble the word.

Explore the game:

If you want to play with my final program to get a feel for this game, click the link given at the end of the article. Try not to peek at the scripts yet, since we want to design them ourselves below.

1. Click the "Green flag".
2. Read the instructions.
3. Click "Play" to start the game. Player1 should enter the secret word.
4. Player2 then tries to unscramble the word:
    a. Click the letters to recreate the word left to right.
    b. Click "Hint" when stuck.
5. Click "Play" to play the game again.
6. Check your score.

Click here to download all the program files.

Snap and CS Concepts Used

When we design this program, we will make use of the following Snap and CS concepts. Learn these concepts if you don't know them before proceeding further.

- Algorithms
    - Abstraction
    - Using algorithms
    - Designing new algorithms
- Animation using costumes
- Arithmetic
    - Expressions
    - Basic operators (+, -, *, /)
    - Advanced operators (round)
- Concurrency

    - Synchronization using broadcasting
- Conditional statements:
    - Conditions: YES/NO questions
    - Relational operators (=, <, >)
    - Conditionals (IF)
    - Conditionals (If-Else)
    - Conditionals (Wait until)
    - Boolean operators (and, or, not)
- Data structures – list
    - List operations
    - List traversal
    - Advanced operations (map, combine)
- Data types – basic
    - Integers
- Data types – strings
    - String operations (join, letter, length of, split)
- Divide and conquer (program design technique)
- Events
- GUI (graphical user interface)
    - Basic widgets: labels, buttons
    - Synchronizing backend logic with GUI
- Looping (iteration)
    - Looping - simple (repeat, forever)
    - Looping - conditional (repeat until)
- OOP
    - Clones
    - Clones differentiation: using private id
- Procedures
    - Built-in
    - User defined (custom)
    - Simple
    - With inputs
- Program output
    - Text
- Random numbers
- Sequence
- Stopping scripts

- User input
    - Text
    - Click buttons
    - Input validation
- Variables
    - Simple
    - Local/global scope
- XY Geometry

Version 1 High Level Design

As usual, we can consider the backend (logic) and the frontend separately. The backend will hold the data (the words) and will contain all the logic for the game. The frontend will handle display of information and user interaction.

The initial "basic" version of our program uses a simple user interface but implements the essential logic of the game. It allows 2 human players play the game between them. The "frontend" just displays the variables.

The backend logic needs to do the following:

(1) Keep track of the words: both the original and scrambled.
(2) Whenever a letter is entered, verify that it is the expected one.
(3) Detect when the game is over when the word has been fully unscrambled.

Objects:

The game in this version is simple enough to have all the code with a single object (sprite).

Data:

There are 3 words we need to handle:
(1) The actual word entered by Player1, let's call it "input".
(2) The scrambled representation of the same word, let's call it "jumble".
(3) The "output" word which Player2 will construct letter by letter.

Snap's capability of handling strings (i.e. words) is quite limited. Since we have to do a lot of string processing, i.e. juggling of letters, in this game, we will use lists instead of strings.

(1) List "inputL" will hold the input word.
(2) List "jumbleL" will hold the scrambled version.
(3) List "outputL" will hold the output string.

Feature Idea # 1: Set things up

Get a word from Player1 and convert to list format.

Design:

We will use the ASK command to get the word, and then use the split operator to convert to a list.

Given: inputWord (string of letters)

We also need to create the list "outputL" of the same length as "inputL" but containing all "_" characters (to represent blanks). The following map operator does that job.

Remember, the game should hide "inputL" since it is the secret word, and only show "outputL" and "jumbleL".

Feature Idea # 2: Scramble the word

Scramble the word supplied by Player1 and convert to list format.

Design:

We already know how to convert to list format. How do we scramble the word? It is the process of changing the order of the letters. Our "random" operator comes to rescue. See the algorithm below.

```
Algorithm Jumble
Given: list inputL
Steps:
jumbleL = empty list
Repeat until inputL is empty
      Pick a random letter C from inputL
      Append C to jumbleL
      Delete C from inputL
End repeat
```

It is true we are destroying the list inputL in this process, but we still have the original word in "inputWord".

Feature Idea # 3: Play the game

Get letters from Player2 and reconstruct the word.

Design:

We now have the list "jumbleL" that contains the word but in a different order, and the list "outputL" which shows all blanks (the letter "_" for better visibility). We need to run the following sequence to play the game:

```
Repeat until the word is fully reconstructed
      Get a letter from Player2
      Check if it is a correct entry
      If not, flag error
      Else, copy it to "outputL" and remove it from "jumbleL"
End repeat
```

How do we check if Player2 has entered a correct letter? Since the word must be reconstructed from left to right, we could look at "outputL" and "inputWord" and find out which letter is expected next. For example, if inputWord is "program" and outputL contains "p" followed by 6 "_" letters, we know the next expected letter is "r".

Let us take care of these steps in the following algorithm.

```
Algorithm Letter Clicked
Given: letter C (entered by user)
I = Get index of item "_" (using the "index of" operator)
Ch = I'th letter in inputWord
If C = Ch
      Copy C at location I in outputL
      Remove C from jumbleL (see below)
Else
      Tell user the entry was incorrect
End if
```

Removing C from jumbleL is straightforward. We can once again use the "index of" operator to locate C and then delete it. The list may contain multiple instances of C. We only delete the first.

Save as Program Version 1

Congratulations! You have completed all the features of Version 1. Compare your program with my program at the link below.

File: scramble-1.sb2

How to play the game:

1. Click the "Green flag".
2. Player1 should enter the secret word.
3. Player2 then tries to unscramble the word by entering letters to recreate the word left to right. The lists "outputL" and "jumbleL" show the progress.

Version 2 High Level Design

In this "user-friendly" version, we will have a proper, nice-looking graphical interface. Instead of entering letters by typing, user would be able to click on a sequence of letters. Also, the words will be displayed as graphical letters instead of variables.

The following feature ideas describe in detail how we can include this new interface and what changes are required to be made to the existing code.

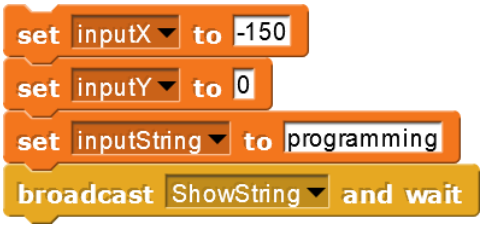
Feature Idea # 4: Graphical display

Display both the jumbled list and the reconstructed list in graphical format.

Design:

We will use one of my earlier programs "ShowString" as a starting point. This program takes a string (sequence of alphabet) and displays it in a graphical format. Look up this guide to figure out how to download this program.

Here is an example of how this code is invoked:



The string is displayed at (-150, 0).

Display:

The sprite that comes with this code has 26 costumes each showing one of the letters. The costumes are named as "A-glow", "B-glow", and so on. Each letter displayed is a clone of this sprite.

We will need to modify this component slightly to suit our purpose as described below.

Step 1: Make modifications suitable for the reconstructed word.

(1) In our game, the reconstructed word is initially all blank. We should show it as a sequence of blank squares. For example, if the input word is "computer" we will show 8 squares as below.

And then, as Player2 starts entering correct letters, they will start appearing in place of the squares. In the above example, after entering c, o, and m, the display would look like this:

To achieve this, we will rename the "blank-glow" costume as "_-glow" since we represent blank characters with "_".

(2) We need to associate each clone with a letter in "inputWord". We can achieve that by providing 2 private variables to each clone: "myLetter" will contain the current letter displayed by the clone, and "myPosition" will indicate the location of the letter in "inputWord". For example, for the clone showing "M" in the above example will have myLetter=M and myPosition=3. When the game starts, all myLetters will be "_" since the output is a empty string.

(3) Each clone will calculate its position on the screen using the starting (X, Y), "myPosition", and width of each costume.

(4) Since the word changes every so often, each clone will need a separate script to update itself. A string variable will help the clone know its new letter, if any. The clone will then change its costume if necessary.

With these changes, we will have a suitable component (sprite and code) to take care of displaying the reconstructed word throughout the game.

Step 2: Make modifications suitable for the jumbled word.

We will need all the changes made above (in Step 1) for the jumbled word also. So, we will simply duplicate the sprite for the jumbled word. But we will need a few more changes as described below.

(1) Unlike the reconstructed word, the jumbled word is interactive. That is, user can click on its letters. So, we will need a script that takes a user click and transmits the letter ("myLetter") to the backend script "letter clicked" (we wrote this script earlier in Version 1).

(2) Once a letter has been clicked and accepted into the reconstructed word, it needs to disappear from the jumbled word. This code can be added to the "update" script: if the new "myLetter" is "_" the clone should delete itself.

That is all! We are now equipped with the necessary capability to use graphical display for our game. We will name these display sprites "output" and "jumble".

In the next feature, we will add the necessary code in the backend logic to drive these sprites.

Feature Idea # 5: Integrate graphical display.

Modify the backend logic to use the new graphical display.

Design:

Currently, the backend logic simply updates the lists "outputL" and "jumbleL" to indicate the progress of the game. It also uses a simple ASK loop to receive use input. Now, we need to add code to make use of the graphical display and interact using mouse clicks.

(1) First, we will need to initialize both display sprites (by calling their "init" scripts) by passing the starting position (X,Y) and the length of each word. Only the Y value will be different so that the two words are displayed one below the other.

(2) Next, after creating "jumbleL" (scrambled version of the word) we will need to call the "update" script of the jumble display sprite.

(3) Both the "update" scripts require strings (and not lists), so, we will need to convert our lists into string formats before calling the update scripts.

For this purpose, we can use the "combine" operator. See example below:
Given "strlist" as a list of characters (or strings), the following reporter would concatenate and return all items in strlist into a single string.

(4) We will need to update both displays again from the "Letter clicked" script if a valid letter was clicked which would result in changes in both words.

(5) Earlier, we deleted letters from "jumbleL" after they were accepted in "outputL". But now instead of deleting we will replace them with "_".

(6) Since all costume names use uppercase letters, such as A-glow, B-glow, etc., we will convert the input word to all uppercase.

(7) We need a new trick to detect when the game is over. We do this by comparing the original word "inputWord" with the reconstructed word. As soon as they become equal, the game is over.

Save as Program Version 2

Congratulations! You have completed all the features of Version 2. Compare your program with my program at the link below.

File: scramble-2.xml

How to play the game:

1. Click the "Green flag".
2. Player1 should enter the secret word.
3. Player2 then tries to unscramble the word: Click the letters to recreate the word left to right.

Final version High Level Design

In this version, we will give finishing touches to our game by including the following features:

- A "Hint" feature which discloses one letter at a time.
- "Play" button which allows the game to be played again and again.
- Keeping score.
- Welcome and help screens.
- User input validation: single word, limited length
- Center the words

Feature Idea # 6: Hint

Every time user clicks "Hint" one letter is exposed.

Design:

This is a way to help Player2 when they are stuck. "Hint" would pick one of the remaining letters and show it. How do we provide this facility?

As always, there are multiple ways, but we will consider the following idea.

The "outputL" list can tell us which letters are as yet un-recognized; they are shown as "_". We could pick one of these and expose it. See the following steps:

(1) Pick a random blank letter ("_") from outputL, get its location I.
(2) Get the corresponding letter from inputWord at index I and call it C.
(3) Copy C in outputL at I, update output display.
(4) Locate C in jumbleL and replace it with "_", update jumble display.

Here is the algorithm:

```
Algorithm Hint
Steps:
If list outputL does not contain "_"
      Stop
End if
Len = length of outputL
Repeat until
      Pick a random number I between 1 and Len until
      the item at I is "_"
```

```
End repeat
C = letter in inputWord at index I
Replace item at I in outputL by C
Update output display
Traverse list jumbleL until you find "_". Note its location I.
Replace item at I in jumbleL by "_"
Update jumble display
```

Feature Idea # 7: Play button

Add a "Play" button which allows the game to be played again and again.

Design:

This is a straightforward change. So far, we have been running the game by clicking Green Flag. Instead, we will have the "Play" button send a "start" message to run the same script.

Feature Idea # 8: Welcome and Help

Include a welcome screen and provide instructions on how to play the game.

Design:

We will combine these two into a single screen which would appear when Green Flag is clicked. This welcome/help screen would be a separate sprite which would get out of the way upon clicking anywhere.

Feature Idea # 9: Score

Include a score feature which will give points when letters are recognized correctly and take away points otherwise or when hint is used.

Design:

We just need to create a "score" variable and decide the strategy for giving and taking away points. Here are the rules that I will use in my program:

- For every correct letter selected, give 10 points.
- For every incorrect letter selected, take away 5 points.
- For every use of "Hint", take away 5 points.

We will reset "score" to 0 every time Green Flag is clicked.

Feature Idea # 10: Error handling

Add code to validate user input.

Design:

Since we are using letter costumes to display words, we have certain limitations on what kind of input strings we can handle.

(1) We can only handle single words without SPACE characters.
(2) The words can only be as long as the screen can accommodate.

Include these checks in your startup script. The first check can be accomplished using the "split" operator: if the number of words is 1, we are okay.

Feature Idea # 11: Center the words

Display the words such that they appear centered on the screen.

Design:

We know the following: length of the words (same for both), width of each letter costume. Using this information, how can we ensure that the words are always displayed in the center of the screen?

We basically need to calculate the X position using the above information. Here is the formula that can do it for us:

X = -32 x round(wordlength / 2)

Here, I am assuming that the costume width is -32. We need to use "round" because sometimes the word length would be an odd number.

Save as Program Version "Final"

Congratulations! You have completed all the features of the game. Compare your program with my program at the link below.

File: scramble-final.xml

Berkeley website: Link

How to play the game:

4. Click the “Green flag”.
5. Read the instructions.
6. Click “Play” to start the game. Player1 should enter the secret word.
7. Player2 then tries to unscramble the word:
    a. Click the letters to recreate the word left to right.
    b. Click “Hint” when stuck.
8. Click “Play” to play the game again.
9. Check your score.

Author: Abhay B. Joshi (abjoshi@yahoo.com)
Last updated: 7 February 2021