Abraham Gebrekidan
EIP 01

Reading #29
Sep 02, 2024

Assignment #4—HangKarel

Due: Monday, Sep 16, 5:00P.M.
Note: This assignment must be completed individually

For Assignment #4, your mission is to write a JavaScript program that plays the classic word-guessing game of Hangman, but with a slight twist. Instead of drawing the traditional stick-figure of a human, the potential victim of this execution is our beloved Karel the Robot. Your mission as the user is to save Karel by guessing all the letters in the secret word before the entire image of Karel appears.

At the beginning of a game of HangKarel, the computer chooses a secret word from a list of words stored in a data file. It then displays the word with every letter replaced by a hyphen. For example, if the secret word is **FRUSTRATE**, the computer would display a row of nine hyphens, like this:

```
---------
```

The computer then asks the user to choose a letter by clicking the mouse on one of the 26 letters displayed on the graphics window. If the user guesses a letter that is in the word, the word is redisplayed with all instances of that letter shown in the correct positions, along with any letters correctly guessed on previous turns. For example, if the user begins by guessing the letter **E**, the computer will update the word to show that the secret word has an **E** in the final character position, as follows:

```
--------E
```

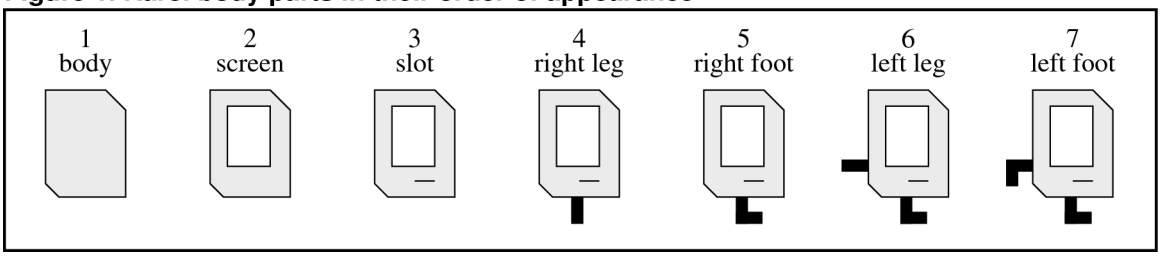
If the user were then lucky enough to guess **R**, the computer would fill in both copies of that letter and update the display like this:

```
-R---R--E
```

If the letter does not appear in the word, the user is charged with an incorrect guess. The user keeps picking letters until either (1) the user has correctly guessed all the letters in the word or (2) the user has made seven incorrect guesses.

When played by children, the morbid fascination of the game comes from the fact that incorrect guesses are recorded by drawing an evolving picture of the user being hanged at a scaffold. For each incorrect guess, a new part of a stick-figure body is added to the scaffold until the hanging is complete. In our HangKarel version, there are seven body parts added in the order shown in Figure 1 at the top of the next page, which means that the user must guess the secret word without making seven incorrect guesses.

Figure 1. Karel body parts in their order of appearance



HangKarel is played entirely on the graphics window using mouse events. The window size is set to 500 × 300, which is the largest window that SJS displays inside its graphical user interface. You can certainly make the window larger if you want to add more graphics and special features, but doing so means that the graphics window will often overlap the rest of SJS, which makes debugging harder.

The bottom of the graphics window shows 26 letters—one for each letter in the alphabet—each of which is implemented as a `GLabel`. These letters act as buttons. To guess a letter, the user clicks on one of these 26 buttons. If the guess is correct, HangKarel updates the display of the secret word, which is located just above the selectable letters. If the guess is incorrect, HangKarel adds the next body part to the top of the window. The program also changes the color of the letter to record the guess. Incorrect guesses are shown in red (specified by the constant `INCORRECT_COLOR`) and correct guesses are shown in green (specified by the constant `CORRECT_COLOR`).

Figure 2 shows the graphics window after the user has correctly guessed the `E` and `R` in `FRUSTRATE` but has also incorrectly guessed `I` and `N`. As in any well-structured program, the location and sizes of the elements appearing in the window are specified as constants, which are shown in the `HangKarel.js` starter file in Figure 3.

Figure 2. The HangKarel game in progress

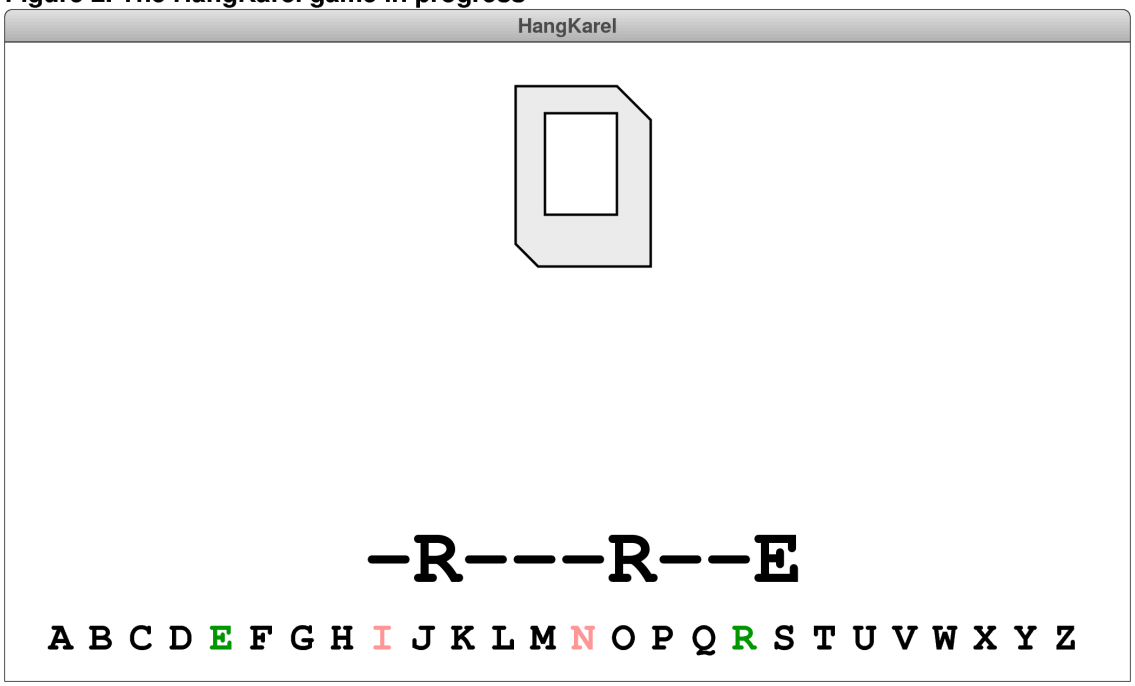


Figure 3. The HangKarel.js starter file

```
/*
 * File: HangKarel.js
 * ------------------
 * This program plays a version of the classic Hangman game in which your
 * mission is to save Karel from being hung.
 */

import "file";
import "graphics";
import "RandomLib.js";

/* Constants */

const GWINDOW_WIDTH = 500;          /* Width of the graphics window        */
const GWINDOW_HEIGHT = 300;         /* Height of the graphics window       */
const LETTER_BASELINE = 10;         /* Distance from bottom to the letters */
const LETTER_POINTSIZE = 18;        /* Font size used for letter buttons   */
const WORD_BASELINE = 45;           /* Inset from bottom to secret word    */
const WORD_POINTSIZE = 36;          /* Font size for the secret word       */
const MESSAGE_BASELINE = 110;       /* Inset from bottom to message area   */
const MESSAGE_POINTSIZE = 60;       /* Font size for messages              */
const MAX_INCORRECT_GUESSES = 7;    /* Number of incorrect guesses allowed */
const INCORRECT_COLOR = "#FF9999";  /* Color used for incorrect guesses    */
const CORRECT_COLOR = "#009900";    /* Color used to mark correct guesses  */

/* Constants that define the Karel image */

const KAREL_IMAGE_TOP = 20;         /* Inset from top to Karel image       */
const BODY_WIDTH = 60;              /* Width of Karel's body               */
const BODY_HEIGHT = 80;             /* Height of Karel's body              */
const BODY_COLOR = "#EEEEEE";       /* Fill color for Karel's body         */
const UPPER_NOTCH = 15;             /* Size of the upper right notch       */
const LOWER_NOTCH = 10;             /* Size of the lower left notch        */
const SCREEN_WIDTH = 32;            /* Width of the screen rectangle       */
const SCREEN_HEIGHT = 45;           /* Height of the screen rectangle      */
const SCREEN_INSET_X = 13;          /* Inset from left to the screen       */
const SCREEN_INSET_Y = 12;          /* Inset from top to the screen        */
const SLOT_WIDTH = 15;              /* Horizontal length of the disk slot  */
const SLOT_INSET_X = 30;            /* Inset from left to the disk slot    */
const SLOT_INSET_Y = 68;            /* Inset from top to the disk slot     */
const LEFT_LEG_INSET_Y = 55;        /* Inset from top to the left leg      */
const RIGHT_LEG_INSET_X = 24;       /* Inset from left to the right leg    */
const LEG_BREADTH = 8;              /* Breadth across each leg segment     */
const LEG_LENGTH = 20;              /* Length of each leg segment          */
const FOOT_BREADTH = 8;             /* Breadth across each foot segment    */
const FOOT_LENGTH = 20;             /* Length of each foot (overlaps leg)  */

/* Main program */

function HangKarel() {
   var gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
   var words = File.readLines("HangmanLexicon.txt");
   if (words === undefined) {
      console.log("The HangmanLexicon.txt file is missing.");
      return;
   }

   // You fill in the rest along with any helper and callback functions

}
```

As with the Breakout program, you should design, implement, and test your program in several parts, each of which represents an achievable milestone. The rest of this handout describes these milestones in more detail.

Milestone 1: Display the letters at the bottom of the window

Your first milestone is simply to display the 26 letters at the bottom of the window. Each one of these letters is stored in its own `GLabel` object. The baseline for these letters is given by the constant `LETTER_BASELINE`, and each letter should be displayed in a monospaced bold font whose point size is given by the constant `LETTER_POINTSIZE`. What you have to do is figure out how to arrange these labels so that they show the 26 uppercase letters and are spaced uniformly across the bottom of the window as shown in Figure 2.

Milestone 2: Detect mouse clicks on the letters

The fact that each letter is a separate `GLabel` makes it possible to use the `getElementAt` method to determine which letter the user has selected, in much the same way that you recognized collisions in the Breakout program. You need to add the code to detect mouse clicks and define a listener function that detects when the mouse is clicked on one of the letters. You first need to check whether the click is in the bottom portion of the window to ensure that none of the other `GObject` instances responds to the user action. As long as the click is in that region, any object returned by `getElementAt` must be one of the 26 `GLabel` objects containing a letter.

Once you have determined which `GLabel` was clicked, you can get the letter by calling the `getLabel` method, which returns the string the `GLabel` displays. At the moment, you don't have anything particularly useful to do with that information, but that fact shouldn't stop you from testing this milestone and making sure you have it working. You can, for example, call `console.log` to display the letter on the console. And since all letters are in some sense incorrect at this point, you might also change the color of the `GLabel` to the constant `INCORRECT_COLOR`. You can go back and modify the code for the mouse-event listener when you have built the rest of the game.

Milestone 3: Choose a random secret word and display it in its hidden form

Reading in the list of possible words and choosing a random one was planned to be part of the assignment. In SJS, this process is extremely simple as long as you can use the array and file operations described in Arrays session. Since you haven't seen those yet, we think that the best thing to do is simply to give you the necessary code. The lines necessary to read in the list of words—which computer scientists call a ***lexicon*** to emphasize that the data structure does not contain definitions of the sort you would expect in a dictionary—are included in the starter file:

```
var words = File.readLines("HangmanLexicon.txt");
if (words === undefined) {
   console.log("The HangmanLexicon.txt file is missing.");
   return;
}
```

Most of this code is concerned with what happens if you accidentally delete the lexicon file that is included with the starter folder. If the **HangmanLexicon.txt** file is there, the line

```
var words = File.readLines("HangmanLexicon.txt");
```

reads the entire file into an array of lines and then stores that array in the variable **words**. To select a random element from that array and assign it to **secret**, you simply write

```
var secret = words[randomInteger(0, words.length - 1)];
```

Picking the random word, however, is not the interesting part of this milestone. In addition to the secret version of the word you choose, your program has to keep track of the mystery word as the user sees it on the graphics window. Initially, the mystery word consists of a string of hyphens, one for each letter in the secret word.

To complete this milestone, you need to do the following things:

- Choose a random secret word from the lexicon.
- Create the mystery version of the word by assembling a string of hyphens.
- Create a **GLabel** that contains the mystery word and display it on the window. The word should be centered horizontally in the window at the baseline specified in the constants.

Milestone 4: Implement the code that updates correctly guessed letters

Your next task is to go back to the function that responds to mouse clicks and add whatever code you need to update the mystery word as the user guesses letters that appear in the word. To do so, it is useful to write a helper function that goes through the secret word and updates the corresponding position in the mystery word for every letter position in which the guess appears. If any such matches occur, your function that responds to the click action should change the color of the label to the shade of green represented by the constant **CORRECT_COLOR**. If no matches occur, the label should be set to the reddish color stored in **INCORRECT_COLOR**.

Milestone 5: Draw successive body parts of Karel for each incorrect letter

In addition to changing the color of the letter to red, each incorrect guess has to display the next body part in the Karel diagram at the top of the window. You will need to maintain a variable that keeps track of the number of incorrect guesses and then write the code necessary to add the **GObject** necessary to display the next piece of Karel's body, as shown in Figure 1. Effective decomposition is the key to success here.

Milestone 6: Determine when the game is over and display appropriate messages

The final step in the process consists of determining when the game is over. The user wins when all letters have been guessed correctly, at which point you need to display the message "You win!" centered horizontally **MESSAGE_BASELINE** pixels above the bottom of the window. The user loses when the number of incorrect guesses reaches **MAX_INCORRECT_GUESSES**. At that point, your program should reveal the rest of the mystery word and display "You lose!" in the message area.

Extensions

There are many things you could do with HangKarel to make it more sophisticated. Once you get the basic structure working, you could try some of the following ideas:

- Check to make sure that the user has not already guessed the letter and display some message to that effect in the message area of the window. You will need to remove that message when the user enters a new guess.
- Spice up the display a little. Each of the body parts in the assignment is a single `GObject`, but you could add more detail.
- Animate the graphical display. Instead of having the body parts and letters merely appear on the screen, you could have them move in from offscreen, as they often do, for example, in PowerPoint slides..
- Expand the program to play something like Wheel of Fortune, in which the single word is replaced by a common phrase and in which you have to buy vowels.
- Use your imagination!