Assignment #3—HangKarel

This assignment was brought to you by Eric Roberts and the letters E, S, and R.

Due: Friday, October 18, 5:00 p.m.

For Assignment #3, 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 is the beloved Karel the Robot, a central figure of our introductory CS106A courses. 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. It then displays that word with every letter replaced with 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 stems from the fact that incorrect guesses are recorded by drawing an ever-growing 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 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.
HangKarel is played entirely on the graphics window using mouse events. The window size is set to 500×300 (although you can certainly make the window larger if you want to add more graphics and special features).

The bottom of the graphics window shows 26 letters—one for each letter in the alphabet—each of which is rendered using a GLabel, which themselves function as tiny 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.
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 bold monospaced\(^1\) 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 \texttt{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 \texttt{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 \texttt{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 \texttt{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

When we’ve used Hangman as an exercise in past offerings of CS 106A, reading in the list of possible words and choosing a random one has been part of the assignment. In our dialect of JavaScript, this process is extremely simple as long as you can use the array operations described in Chapter 8. Since you haven’t seen those yet, we think that the best thing to do is simply to give you the necessary code. In addition to the \texttt{HangKarel.js} file you’ll be modifying, your assignment folder includes \texttt{HangKarelWords.js}, which contains the definition of a global constant so large it deserves its own file. You needn’t change it or even look inside if you don’t want to. But you do need to rely on its declaration of a global constant called HANGKAREL_WORDS, which is an array of the various strings HangKarel can choose from when establishing the secret word. To randomly select a word from that array, all you need to write is:

\begin{verbatim}
let numWords = HANGKAREL_WORDS.length;
let secret = HANGKAREL_WORDS[randomInteger(0, numWords - 1)];
\end{verbatim}

\(^1\) We ask for a monospaced font like \texttt{Courier}, so that all letters are equally wide.
Picking the random word, as it turns out, 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 via 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`. The GLabel call includes `getLabel()` and `setLabel(text)` methods that can be used to extract and update the text on display.

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

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, or even better, you can remove that message after a short time interval.

- 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 the class’s PowerPoint slides.

- Allow the user to press keys instead of using the mouse.

- Expand the program to play something like Wheel of Fortune, in which the single word is replaced by a common phrase.

- Use your imagination!

If you decide to extend HangKarel, be sure to maintain an extension-free version of your work so you can submit that aside the version with all the extensions.