Eric Roberts
CSCI 121

Handout #27
October 3, 2018

Project #2—Snowman

Due: Friday, October 11, 11:59 P.M.

For Project #2, your task is to write a Python program that plays the classic game of Hangman, with just a little updating to avoid seeming so macabre (and also to offer at least a bit of homage to Mary Shelley 200 years after the publication of *Frankenstein*). This version is called "Snowman" and draws its inspiration from the following cartoon from *Calvin and Hobbes: The Attack of the Deranged Mutant Killer Monster Snow Goons:*

Knowing that Calvin is about to create the first in an army of snow goons that will threaten the entire world, your mission is to guess the magic word that will disable the creatures before Calvin has finished creating the first one.

This assignment is designed to build your skills in the following areas:

1. *String manipulation.* The game requires you to update a string with every occurrence of a letter in a secret word.
2. *New graphical objects.* The game requires you to use the **`GPolygon`** and **`GCompound`** classes, so that you have some practice with those.
3. *Modular development.* The Snowman game is divided into four independent Python modules, two of which you have to write. Large projects are always subdivided into modules, and it is important to understand what information each module keeps private and what facilities it exports for use by other modules.

Playing the Snowman game

At the beginning of a game of Snowman, the computer chooses a secret word from a list of hard-to-guess English words stored in the **`PythonWords.py`** module, which is given to you. 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:

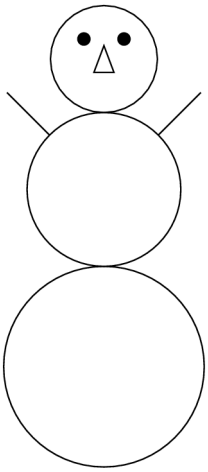
```
-R---R--E
```

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 eight incorrect guesses.

To remind you of what's at stake, each of your incorrect guesses are recorded by drawing a new piece of the growing snowman in the following order:

1. The snowball (i.e., a **GOval**) that represents the base of the snowman
2. The slightly smaller snowball (**GOval**) for the body
3. The still smaller snowball (**GOval**) for the head
4. The twig (**GLine**) for the left arm
5. The symmetrically defined twig (**GLine**) for the right arm
6. The charcoal briquette (filled **GOval**) for the left eye
7. An identical charcoal briquette (filled **GOval**) for the right eye
8. A carrot (**GPolygon**) for the nose

The finished snowman looks like this:



Snowman is played entirely on the graphics window using mouse events. 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, Snowman updates the display of the secret word, which is located just above the selectable letters. If the guess is incorrect, Snowman 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`).

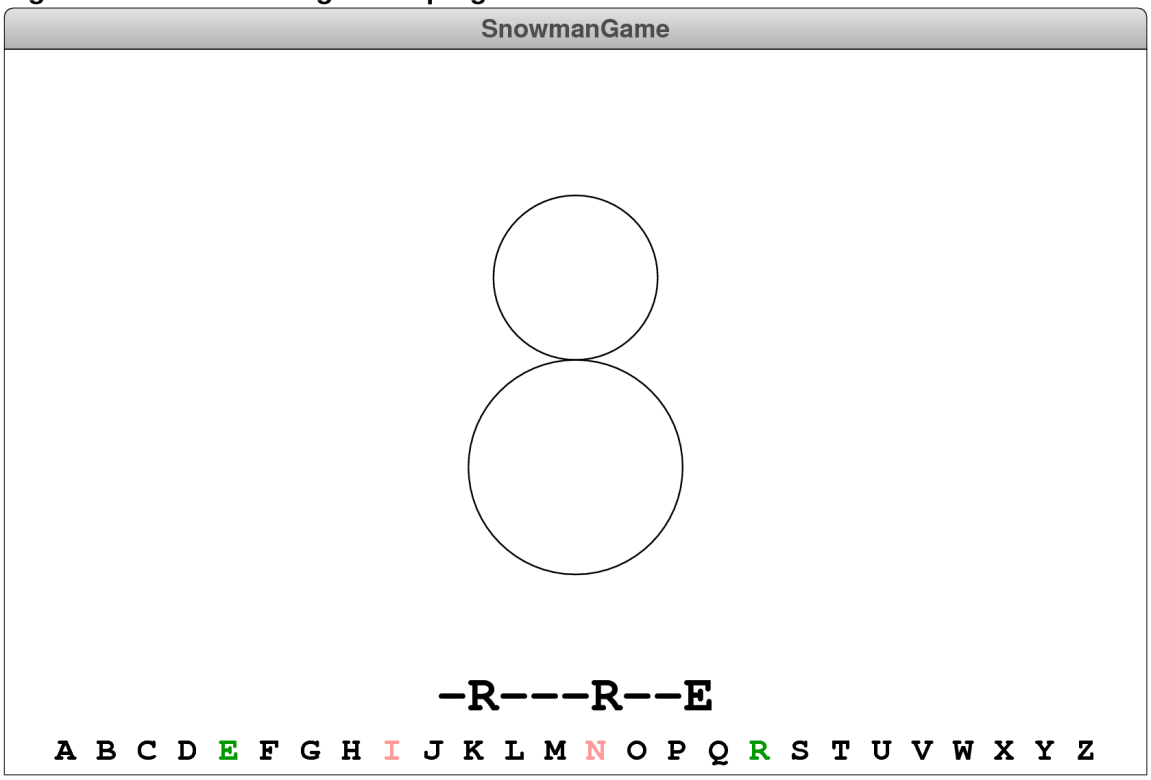
Figure 1 shows the graphics window after the user has correctly guessed the `E` and `R` in `FRUSTRATE` but has also incorrectly guessed `I` and `N`. As in any well-structured program, the location and sizes of the elements appearing in the window are specified as constants, which are shown in the `SnowmanConstants.py` file in Figure 2 at the top of the next page.

Module structure for the Snowman game

The Snowman game is implemented using four different Python modules:

- `SnowmanGame.py`. This module implements the game and is supplied to you with a stub implementation of the `SnowmanGame` function that creates the graphics window but does nothing else. Your job is to complete the implementation.
- `SnowmanGraphics.py`: This module is responsible for drawing the various pieces of the snowman figure on the graphics window. The version in the starter project

Figure 1. The Snowman game in progress



contains two functions—**createEmptySnowman** and **addSnowmanPart**—along with the docstring comments that describe what those functions are supposed to do. You just need to implement them.

- **SnowmanConstants.py**. This module defines all the constants used in the Snowman game. These constants are needed in both the **SnowmanGame** and **SnowmanGraphics** modules, so it makes sense to put the constants in a separate module and import them into the namespaces for each of the modules you have to write.
- **SnowmanWords.py**. This module exports a constant called **SNOWMAN_WORDS**, which is a list of all the words that the computer can choose. The list does not include most English words but rather a collection of words that are hard to guess.

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.

Figure 2. The SnowmanConstants.py starter file

```
# File: SnowmanConstants.py

"""
This module defines all the constants that are used in the Snowman game.
The easiest way to use this module in your code is to import all the
definitions by writing

    from SnowmanConstants import *

This statement brings every definition in this module into the local
namespace.
"""

GWINDOW_WIDTH = 800             # Width of the graphics window
GWINDOW_HEIGHT = 500            # Height of the graphics window
LETTER_BASE = 10                # Distance from bottom to the letters
WORD_BASE = 45                  # Distance from bottom to secret word
MESSAGE_BASE = 85               # Distance from bottom to message area
SNOWMAN_BASE = 140              # Distance from bottom to Snowman base
INCORRECT_COLOR = "#FF9999"     # Color used for incorrect guesses
CORRECT_COLOR = "#009900"       # Color used to mark correct guesses

# Body part constants

BASE_SIZE = 150                 # Diameter of the circle for the base
BODY_SIZE = 115                 # Diameter of the circle for the body
HEAD_SIZE =  80                 # Diameter of the circle for the head
ARM_LENGTH = 60                 # Length of the line used for the arm
EYE_OFFSET = 15                 # Vertical distance from head to eye centers
EYE_SEP = 30                    # Horizontal distance between eye centers
EYE_SIZE = 10                   # Diameter of the circle for the eye
NOSE_WIDTH = 15                 # Width of the triangle for the nose
NOSE_HEIGHT = 20                # Height of the triangle for the nose

# Fonts

WORD_FONT = "bold 36px 'Monaco','Monospaced'"
LETTER_FONT = "bold 24px 'Monaco','Monospaced'"
MESSAGE_FONT = "48px 'Helvetica Neue','Arial','Sans-Serif'"
```

Milestone 1: Choose a random secret word and display it in its hidden form

When I've used Hangman as an exercise in past offerings of an introductory course, reading in the list of possible words from a data file and choosing a random one has been part of the assignment. Since I won't get to files in CSCI 121 until the week after the midterm, the easiest thing to do is to put all the possible words in a single list. That list of words is called `SNOWMAN_WORDS`, and it appears in the `SnowmanWords.py` module. You don't have to do anything to this file; all you have to do is import it.

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

- Figure out how to import the `SnowmanWords` variable into your `SnowmanGame.py` implementation.
- Choose a random element from the list to be the secret word.
- Create the mystery version of the word by assembling a string of hyphens exactly as long as the secret word.
- 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 2: Display the letters at the bottom of the window

Your next milestone is 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_BASE`, and each letter should be displayed in a monospaced bold font given by the constant `LETTER_FONT`. 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 3: 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 `print` 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 `clickAction` listener when you have built the rest of the game.

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 the snowman 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 snowman 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 the snowman's body, as shown in Figure 1. Effective decomposition is the key to success here.

One of the requirements of the assignment is to make sure that the entire snowman figure is a `GCompound` object that has all of the various pieces in the right places. The function `createEmptySnowman` function in `SnowmanGraphics.py` should create the `GCompound`, add it to the window, and set its reference point to the location at the bottom of the body circle. Your code for `addSnowmanPart` must then add each part at the appropriate position relative to the reference point.

The implementations of `addSnowmanPart` and its helper functions consist almost entirely of code to ensure that everything is positioned correctly with respect to the reference point. The only minor complexity is in drawing the arms, where you will need to compute the starting and ending points of the line knowing that these lines run at 45° off the vertical. If you remember trigonometry, you can use the `math.sin` and `math.cos` functions here. If not, you can figure everything out using the Pythagorean Theorem. You might also notice that you can use the same helper function to draw each arm if you pass in an argument to show the direction. The same is true for the eyes.

The reason for making the snowman a single `GCompound` is that doing so allows you to move the snowman as a unit. If you want to extend the program so that the completed snowman starts chasing a suitable proxy of Calvin across the screen, the can easily do so. All you have to do is animate the motion of the `GCompound`.

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_BASE` pixels above the bottom of the window. The user loses when the number of incorrect guesses reaches 8 (this value is not defined as a constant because it is built into the operation of `SnowmanGraphics.py`). At that point, your program should reveal the rest of the mystery word and display "You lose!" in the message area.

Once the game is finished, clicking in the window should restart the game by clearing the graphics window, reinitializing all the variables, choosing a new secret word, and letting the user play again.

Extensions

There are many things you could do with Snowman 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. In particular, the arms in the assignment are single twigs. If you use the ideas from the brief introduction to Lindenmayer patterns that were part of the lecture on September 28 (see Handout #21), you could make them branching twigs, as they are in the Calvin and Hobbes cartoon.
- 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. Similarly, you could animate the finished snow goon so that it moves menacingly around the screen.
- Change the story line to make it more socially relevant. A popular bumper sticker at the time of the *Citizens United* court decision reads "I'll believe that corporations are people when Texas executes one." You could go back to a more traditional Hangman style of play using corporate icons instead of body parts.
- 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!