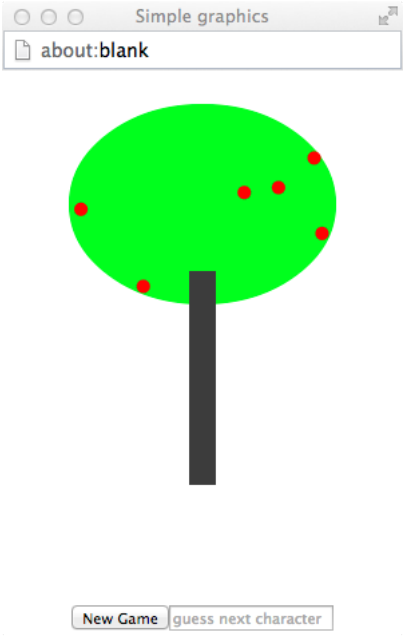
CS 51G Laboratory # 10
Apples

Objective: To practice using String methods.

In this lab we'll be implementing a version of the game Hangman. Hangman is a simple game in which one person, the "*supplier*", thinks of a word and another person, the "*guesser*", tries to guess it. The game starts with the *supplier* writing down a row of dashes, with each dash representing one letter in the word. The *guesser* then repeatedly guesses a letter. If the letter appears in the word, the *supplier* replaces each appropriate dash with that letter. If the letter doesn't appear in the word, the *supplier* makes a note of that. The *guesser* is allowed some number of incorrect guesses before they lose the game. In our case they'll be allowed 6 incorrect guesses.

The name Hangman comes from the fact that instead of simply noting that the *guesser* has lost because they've guessed 6 letters that don't appear in the word, traditionally the *supplier* has drawn a feature of a hanging person for each incorrect letter, with the *guesser* losing when the complete figure of a hanging person has been drawn.

In our case we'll be playing a (G-rated) version of the above game called Apples. In this version, the *supplier* (your program) draws a tree with 6 apples, and each incorrect guess by the *guesser* causes one apple to fall off the tree. When all the apples are gone, the *guesser* has lost.



Guesses in either uppercase or lowercase are acceptable. That is, if there is a "b" in the hidden word then a guess of either "b" or "B" will identify it. Furthermore, nothing happens if you guess a letter that you've already guessed before.

Overview

Your program will consist of two classes plus two objects:

dictionary will be an object that is responsible for reading a file of words to be guessed. It also has a method returning a random word from that file.

apples will be the object corresponding to the main program. It sets up the game interface.

appleGame will be the class that maintains the state of the game.

appleGameDisplay will be the class that maintains the image of the apple tree.

We will provide a complete implementations of the `appleGameDisplay` class, which is described. in the following section. You'll need to write the other three, following the specifications below.

The `dictionary` object The starter folder for apples contains a file appleWords.txt that contains the words that can be used in the game, with each on a different line of the file. The dictionary object will read in the contents of that file, placing all of the words in a list.

This object has a single method:

```
getRandomWord -> String
```

That randomly chooses a word from the list and returns it.

Be sure to load the file `appleWords.txt` into the browser before executing this object. Also be sure to wrap the file handling code in a try-catch block as we did in the demo program `Bookmark.grace` in class. If the file is not found, the catch clause should simply print a message saying that the file `appleWords.txt` was not found.

The `AppleGameDisplay` class This class deals with drawing the tree with the apples and provides methods that let you remove an apple and that let you refill the tree with apples. The type is:

```
type AppleGameDisplay =
    // show all the apples in the tree
    fillTree -> Done

    // return number of apples still visible in tree
    numApplesLeft -> Number

    // hide one of the apples in the tree
    removeApple -> Done
```

The class header is

```
class appleGameDisplay (numApples: Number) on (canvas: DrawingCanvas)
                          -> AppleGameDisplay
```

We have provided all of the code for `appleGameDisplay`, but you are welcome to read it.

The `appleGame` class The `appleGame` class will manage the state of the game. This involves three basic things:

- the word the person is trying to guess
- the current state of the dashed word (i.e. the *guesser's* current partial knowledge of the word), and
- what letters have been guessed so far.

  You should keep track of each of these as a string. (Don't use lists, as we want you to practice string operations.) Don't forget that `String`s are immutable, so any operation that seems to change a string actually generates a new string! Remember there are operations to create new strings out of pieces of old string (and you will need to use them).

  In class `appleGame`, you will: ,

- Generate and update the state of the tree using an `appleGameDisplay` object.
- Generate and update the textual display of the game.

The class for `appleGame` should have the following header:

```
class appleGame(numApples: Number)
          on (canvas:DrawingCanvas) -> AppleGame
```

where

```
type AppleGame = {
    // Set up a new apples games
    newGame -> Done

    // If a game is going on, take a guess and respond appropriately:
    // If correct, add all occurrences to the current word display
    // If wrong, and not guessed earlier, add to list of wrong guesses
    // and remove an apple from the tree
    guessLetter(letter:String) -> Done
}
```

Method `guessLetter` takes a character `letter` and, if it hasn't already been tried, determines whether or not it appears in the word, and updates both the text and the picture accordingly (remember the guesses should be case insensitive).

Method `newGame` resets both the text and tree display (i.e., makes all the apples visible), and chooses a new word from the dictionary. That is, you should be able to play as many games as you like without restarting the program.

The `apples` object This object manages the response to user actions. The object should set up the GUI so that there's a `Button` labeled "New Game" and a `TextField` for entering guesses. It should begin with a word for the user to guess (i.e., the user should not have to press the new game method in order to start playing).

The user interacts with the game by entering a letter into the field and pressing enter. If the string entered is more than one character or is not a letter then the program should not respond, and should again prompt the user to enter a letter.

Design

The things you need to do before coming to the lab are described in the Lab Design Form.

You only need to provide a design for the `appleGame` class and the `WordSupplier` and `apples` objects since we provide the code for the other class. At the beginning of the lab, we'll come around and briefly examine each of your designs to make sure you are on the right track. At the same time, we'll assign a grade to your design.

Starter code is available as usual in the CS51G folder on lab machines, but as a convenience you can also download it from the web:

- `http://www.cs.pomona.edu/classes/cs051G/labs/apples/Starter/AppleGame.grace`.
- `http://www.cs.pomona.edu/classes/cs051G/labs/apples/Starter/AppleGameDisplay.grace`.
- `http://www.cs.pomona.edu/classes/cs051G/labs/apples/Starter/Apples.grace`.
- `http://www.cs.pomona.edu/classes/cs051G/labs/apples/Starter/WordSupplier.grace`.
- `http://www.cs.pomona.edu/classes/cs051G/labs/apples/Starter/appleWords.txt`.

Implementation

As usual, we suggest a staged approach to the implementation of this program. This allows you to identify and deal with logical errors quickly. The dimensions of your window should be 300 pixels wide by 400 pixels tall.

- Start by getting the initial GUI setup so that the textfield and the button are in the right place.
- Now write the dictionary object to read in the contents of `appleWords.txt` into a list. Make sure everything is read in correctly and the `getRandomWord` method is working correctly.
- Now write the initialization code and the `newGame` method for the `appleGame` class, ignoring the tree display. Make sure that when you choose a new random word from the dictionary, you can display the right number of dashes. Add in the action to be taken in the `apples` object when the button is pressed and verify that the `New Game` button causes a new word to be chosen and the display to be updated properly.
- Next write the `guessLetter` method for the `appleGame` class. Again, ignore the tree display, and just make sure that your code correctly determines whether the letter appears in the word and that it updates the text display appropriately. Add in the action for the textfield and verify that the GUI works correctly.
- Finally, add in the code that uses the `AppleGameDisplay` class to update the image of the tree.

The challenging string manipulation question for this lab is how to update the dashed word when a new letter is entered. There are many ways to do this, one reasonable way is to first check if the letter appears in the word at all. If so, iterate through the characters in the word to be guessed, and if the character occurs in it, then update the dashed word. To do this, you can build up a new, temporary, dashed word as you go along, then when you get to the end replace the original dashed word with the temporary one. Think about this problem before beginning coding the `guessLetter` method! *Also don't forget that a letter may appear in a word multiple times.*

Submitting Your Work

The lab is due Monday at 11 p.m. as usual. When your work is complete you should deposit it in the appropriate dropbox. Give it a name of the form `lab11_lastnamefirstname`. Also make sure that your name is included in the comment at the top of all the files you turn in.

Before turning in your work, be sure to double check both its logical organization and your style of presentation. Make your code as clear as possible and include appropriate comments describing major sections of code and declarations.

Table 1: Grading Guidelines

Value	Feature
	Design preparation (4 points total)
1 points	`apples` object questions
1 points	`dictionary` object initialization
2 points	`appleGame` class questions
	Syntax style (5 points total)
2 points	Descriptive comments
1 points	Good names
1 points	Good use of constants
1 point	Appropriate formatting
	Semantic style (4 points total)
1 point	Conditionals and loops
1 point	General correctness/design/efficiency issues
1 point	Parameters, variables, and scoping
1 point	Miscellaneous
	Correctness (7 points total)
1 point	GUI works
1 point	game stops and restarts correctly
1 point	dictionary read in and words chosen properly
2 points	handles letters input correctly
1 point	text display correct
1 point	tree display correct
	Extra Credit (2 points maximum)
.5 point	Better text/tree graphics
.5 point	Better interface