Project 1: Word Guessing Game

Collaboration Policy: This assignment must be completed individually. You may discuss high-level ideas with other students, but you should not share code.

Description

The goal of this homework assignment is to create a word guessing game between two human players. Player 1 provides a word and Player 2 tries to guess what it is using feedback she received from previous guesses.

Player 1 first enters the secret word by entering a response to an "ask" block. The word is *immediately* *hidden* from view after Player 1 finishes entering it. The computer then starts by telling Player 2 how many letters are in the mystery word. After Player 2 makes a guess, the computer should tell the player whether her guess was right or wrong. Remember guesses should be case-insensitive. If Player 2 was wrong, the computer should also tell Player 2 how many letters in her guess were correctly positioned in the actual answer. The computer does not, however, have to say which letters were correct or which positions they were in. The conversation alternates back and forth until Player 2 guesses the secret word correctly.

Here is an example of one of the many ways the game might go:

Computer: What is the secret word?
Player 1: fireball

Computer: There are 8 letter(s) in the secret word. Try to guess it!
Player 2: supercalifragilisticexpialidocious

Computer: No, there are 8 letter(s) in the secret word.
Player 2: facebook

Computer: No, but 3 letter(s) are correct! Guess again.
[f, e, b are in the correct positions]
Player 2: firebell

Computer: No, but 7 letter(s) are correct! Guess again.
[f, i, r, e, b, l, l are in the correct positions]
Player 2: fireball

Computer: Yes, that is correct! You figured out the secret word in 4 guesses.
[...and the game ends]

Remember that this is one example of how the game might go. You need to generalize this to make it work for many different secret words and guesses.

Your game should have at least one sprite, but there are no graphical requirements other than that. Feel free to develop the graphics more if you'd like -- it will come in handy later!

Starting Materials

You should start from a blank SNAP! project.

Bonus

If you want credit for the bonus, you must add a note to the dropbox indicating that you completed it.

Instead of only telling the player that the number of letters that are correct, also tell her the exact letters that are in the correct positions.

Submission Requirements

The following must be placed in the D2L dropbox. Add a comment to the dropbox if you completed the extra credit.

Project file (70 pts)

This is the file with all your Snap! code. You will need to export this from the Snap editor. (Instructions for doing this were in the first lab.) Name the file "wordGuessing" plus your first and last name, such as: wordGuessing_MaryMosman.xml

Readme file (30 pts)

The readme file is a separate file that can be shared along side the code so to give context and extra information about the project. Write a file that has 2 sections:

- The first section, **Summary**, should describe what the program is and does in your own words. In this case, it should describe your game and how to play it. Assume the person reading it has not read these instructions.
- The second section, **Learning**, should describe what you found challenging (or not) about the assignment and how you worked through the challenges and solved the problem. Perhaps you tried something and it didn't work, then you tried a different approach. Perhaps you didn't know where to start and talked it through with a friend. Perhaps the problem was easy because you've done something similar before. The only "wrong" answer here is not talk about how you solved the problem.

Tips

- Save your code often and back it up using OneDrive, GitHub, or a free cloud storage service like DropBox or Google Drive.
- Code a little bit at a time and test as you go.
- Test! In a perfect world, users would read all of the user documentation provided and interact with a program exactly as the designer intended. But as we all know few people in the real world actually read documentation. We have to design good programs to prevent user mistakes or mitigate them. Most importantly, we have to make sure user error does not break our code. Remember to handle the situation when the user guesses a word that is too long or too short.