Project 1: Wordle™-lite

Version 1.0. Last Updated: 2023-06-20.
Due: 2023-06-26 11:59PM PST
We highly recommend reading through this spec in its entirety before you begin.

Contents	**Submission Guidelines**
	• Once you're done, on the Snap*!* window, select the paper icon in the upper-left corner. • Select "Save As" and ensure to save the file to your computer, not the cloud. It should save as a .xml file. • Make sure it's named: `BJC CS10 Project 1 Wordle™-lite.xml` • Submit this .xml file to Gradescope- **make sure to click "Add Group Member" in the top-right corner of the screen and add your partner!** • Your final score out of 10 on this project is your score on Gradescope (out of 1) multiplied by 10.

Do remember that while you may discuss general ideas with other students, sharing code with anyone but your partner would be academically dishonest.

Important: Ensure that you use the starter file we've linked below- do NOT create your own Snap*!* file for this project!

Introduction

In this project, **you are required to work with a partner** (and with assistance from course staff as needed) to finish three blocks in an existing program to play Wordle™-lite, a word-guessing game designed for two human players. In the game, the first player (Player 1) will enter a secret code, and the second player (Player 2) will try to guess that code. Player 2 starts with 100 points, and every wrong guess costs them 10 points, but they earn back the number of letters that match perfectly in the same place in both words (in Wordle™ this would be the green letters). This back-and-forth will continue until Player 2 successfully guesses the secret code or the score is not positive, and the game ends.

Here is an example of the game being played (make sure to resize **HISTORY** to see it better):

Program: Welcome to Wordle™-lite, a word guessing game.
Ready player one: enter a secret word...

Player 1: `love`

Program: Ready player two: time to guess the secret word;
you start with a score of 100
(wrong guesses cost you 10 but each matched letter earns back 1);
the game ends when you guess the word or the score gets to 0.
What is your guess? (hint: it's a 4-letter word)

Player 2: `like`

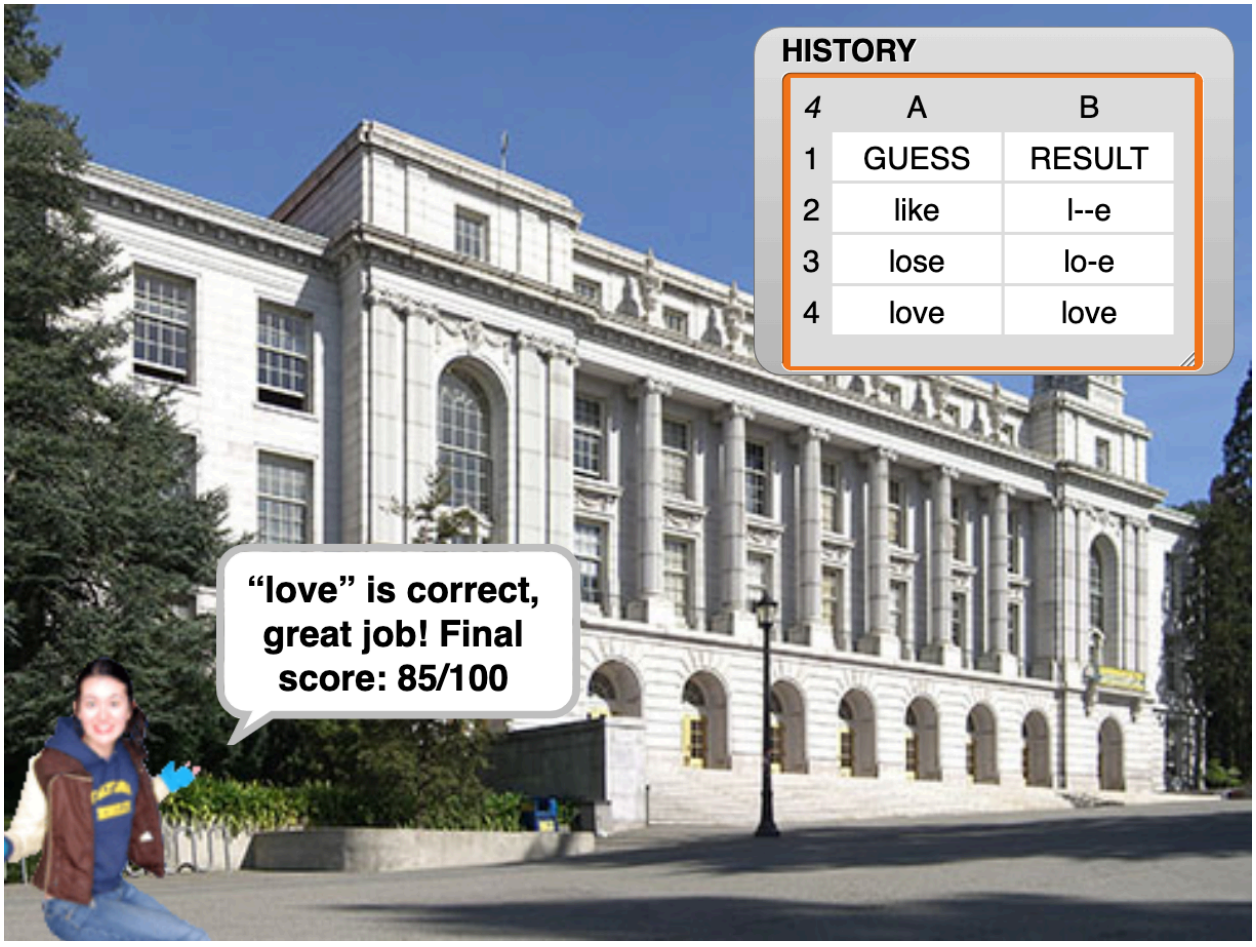
Program: Player two: your guess of "like" matched l--e; your score is now 92.
What is your guess? (hint: it's a 4 letter word)

Player 2: `lose`

Program: Player two: your guess of "lose" matched lo-e; your score is now 85.
What is your guess? (hint: it's a 4 letter word)

Player 2: `love`

Program: "love" is correct, great job! Final score: 85/100



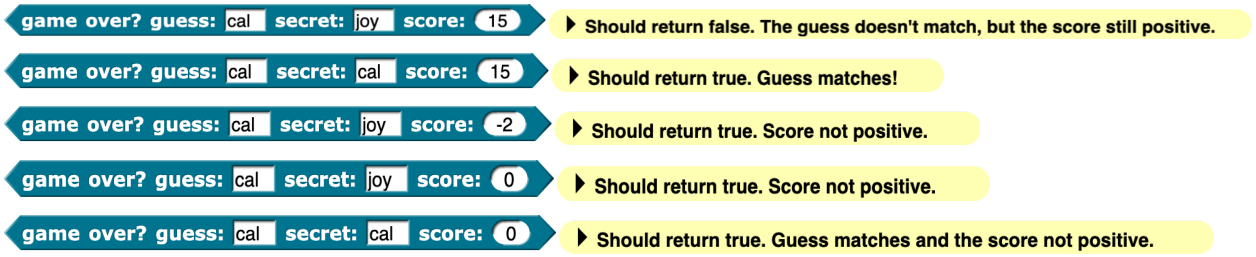
Some notes – nothing in our code prevents either **Player 1** or **Player 2** from typing a gibberish word. We would hope **Player 1** doesn't do that, since the game is almost impossible if the secret word is a set of random letters. However, one strategy that **Player 2** might employ (and this game allows it, making it easier than the actual Worldle™ game) is to type a word (say) of all **e**'s to see where the **e**'s are in the word. Also, the guesses **Player 2** enters aren't checked to see if they're the same length as the secret word. When the guessed word is longer, we only look at the first N slots of the guess (where N is the length of the secret word). When the guessed word is shorter, it just shows the letters that weren't guessed as missed "**-**" letters.

To begin, load **this starter project**.

If you're having trouble, please contact the course staff for assistance — Ed, Office Hours, project parties, and Labs are all here to help you feel good about the work you're doing! If you can't make the times, tell us, and we'll figure out how you can still get support. But, we won't know when or how to help unless you let us know.

Part 1:  Block

First, you'll build a **predicate block** (aka the *"game over"* block) that will know when the game is over. This should report  if the guess equals the secret or if the score is not positive, and  otherwise. Here are some examples:



Part 2:  Block

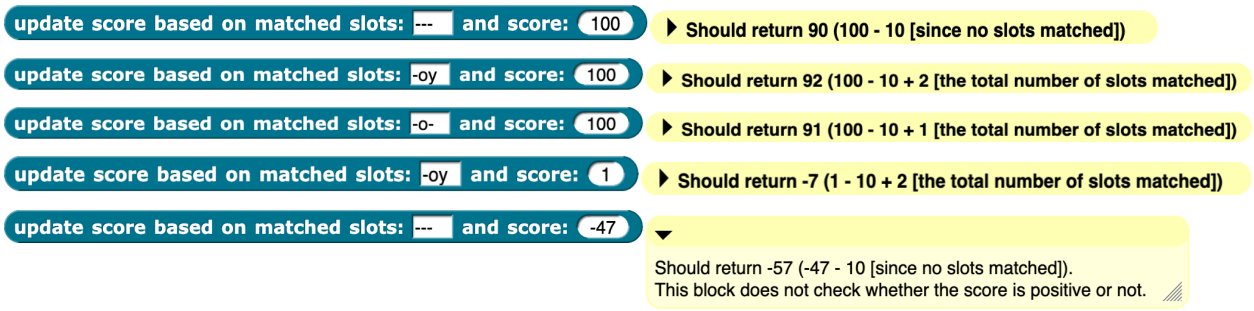
Next, you'll build a **reporter block** (aka the *"matching slots"* block) that can tell a player how close they are to guessing the secret code by indicating the correct "slots". Capitalization should not matter (i.e., the game should read *r* and *R* as the same), but thanks to the way Snap*!* implemented its "=" block, you don't have to worry about making a special case, just using "=" will treat lowercase *r* and uppercase *R* the same! 

Given two inputs, a guess and a secret code, the *matching slots* block should report a word of letters or dashes to indicate the matching slots between *guess* and *secret*, the same length as

secret, in which every matched letter is the letter itself, and any missed letter is a dash. Here are some examples to make this clear:

Part 3: update score based on matched slots: and score: Block

Finally, you'll build a **reporter block** (aka the *"update score"* block) that will take the result of the matched slots and the old score and report an updated score based on the old score minus 10 (cost for a guess) plus the number of slots that matched (were not "-"). Here are some examples to make this clear:



Grading

You have three blocks to write and five tests (shown above) for each block, according to the following table. So a perfect score would earn (5 × 0.4) + (5 × 0.8) + (5 × 0.8) = 10 points. Gradescope's autograder needs the number between 0 and 1, so we divide that score by 10 to send to the autograder. You should continue to work on your code until all test cases pass and the score reported by Autograder says: `{"score": 1}`

If at any point you'd like to see more details about how we calculate that out-of-10 score, you can run the Autograder Table block we provide, a, the *expected* value, the *actual* value, how many points it is worth, and how many points you've earned. The sum of all the earned points is tallied in the bottom-right cell.

Note: correct, working code should handle those test cases, *but not have the test cases hardcoded into your solution*, they should be able to handle *any* inputs according to the specifications.

Block	Points per test
	0.4
	0.8
	0.8