University of Zurich[UZH]

s.e.a.l. software evolution & architecture lab

Informatics I – HS18

Exercise 11

Submission Details

- **Submission Format:** ZIP file containing your .py solution files.
- **Submission Deadline:** 12:00, Tuesday 11th December
- The name of the ZIP file must have the following format:
  firstname_lastname_studentidentificationnumber_info1_exercise_11.zip,
  e.g. *hans_muster_12345678_info1_exercise_11.zip*.
- Your ZIP file should contain the following file: `game.py`. **Do not rename the python files.**
- **Important:** Please follow the naming conventions very strictly, otherwise we will not be able to grade your submission.
- Please submit the assignment using OLAT. You may upload multiple solutions, but note that **only the last submission** will be evaluated.
- Your submissions will be tested on systems running **Python 3.7.X**. Make sure that your solutions work on this version.

Introduction

In this exercise, you will extend the functionality of the 'Hacker Game' code you saw in the last lecture. You will be given the unfinished code and are required to implement some functions to make the game more complete. You will find all necessary instructions below.

1 Task: Hex Code Generator (2 Points)

Your first task is to implement a more sophisticated version of the `generate_hex_codes` function. First have a look at the current implementation (in the file `game.py`; it just returns a list of `0x0000`. To make the game look more realistic, change the `generate_hex_codes` function so that it returns random 4-digit hex codes. The hex codes must still be prefixed with `0x`, but the 4 remaining characters must be chosen randomly from the following alphanumeric characters: `0123456789ABCDEF`.

```
1 def generate_hex_codes(self):
2     return ["0x0000"] * (self.columns * self.rows)
```

Listing 1: Current implementation of `generate_hex_codes`

The exact implementation of your function is up to you, but you have to use the `random.choice` function[1] from Python's `random` package. Use this function to choose random characters of the sequence `0123456789ABCDEF` and construct the hex codes this way; you need to specify exactly this sequence, otherwise the automated tests of your exercise will fail. Like the current implementation, your function needs to return `rows * columns` hex codes.

Task: Modify the `generate_hex_codes` function in `game.py` as indicated above.

[1] https://docs.python.org/3/library/random.html#random.choice

2 Task: Word similarity distribution (3 Points)

The method `word_selection` chooses `num_words` words at random from the word pool for the game. Modify the game logic so that the chosen words tend to be more similar between each other.

First, define in `WordLogic` the method `compute_similarity(a, b, threshold)` which computes the similarity ratio between `a` and `b` and returns `True` if the ratio is (strictly) more than `threshold`. To compute the similarity, check out `SequenceMatcher.ratio`[2].

Finally choose the words as follows:

1. Pick one third of the words at random.
2. For the remaining two thirds:
   (a) pick at random any word already selected.
   (b) pick at random a word from the word pool and then check if it is more than 60% similar to the random pick in (a). If so, add it to the list of selected words. Otherwise, keep trying other words from the word pool until a sufficiently similar one is found.

Note: the return iterable from `word_selection` may not contain any duplicates
Task: Modify the `word_selection` function in `game.py` as indicated above.

[2] https://docs.python.org/3.7/library/difflib.html#sequencematcher-objects

3 Task: Number Passwords (5 Points)

The game uses words as password. Extend the game by creating a new logic class which uses numbers as password.

First of all, add a `GameLogic` abstract class, which defines the constructor the same way it is in `WordLogic` and two abstract methods: `word_selection` and `check`.

Then create a class `NumberLogic` which works as follows:

- The game shows sequences of numbers (e.g. for `length = 4`: 1234, 5251, 5950, 2001)
- The goal is to guess the exact correct sequence.
- When you enter a wrong number, the positions are NOT compared index-by-index, but instead, it just shows how many numbers in the guess appear in the password. For example, if the password is "1003", then the guess "1234" would have the feedback "2/4 correct", because the guess and the password both contain 1 and 3. The guess "1230" would have the feedback "3/4 correct". Or if the guess is "3001" it would have the feedback "4/4 correct" but the wrong order.

Note: the return iterable from `word_selection` may not contain any duplicates. Let both `NumberLogic` and `WordLogic` inherit from `GameLogic`.

Task: Create the new classes `GameLogic` and `NumberLogic`, which work as described.