Color Companion
How to Communicate Colors

Sahar Niknam
sniknam@uni-osnabrueck.de

Introduction

Observing the way people perceive and describe colors is an efficient way of studying the effects of demographic and personal characteristics on the languages structure and usage. And the reason is that while differences between the colors are objectively measurable, color perception and description is utterly subjective, based on the language structure, demographic, and personal characteristics. And that makes a firm ground for measuring and comparing perception and linguistic expression.

Problem Description

The subjectivity of color perception makes it almost impossible for people to have the same mental experience by looking at the exact same color. A further twist of the problem, or in a sense, the only difficulty is that apart from non-identical perception of one color (a specific wavelength) people have different judgements about the quantity and the quality of the difference between two colors. That is, while one person may agree that the difference between the colors *malachite* (00ff55) and *screaming green* (88ff55) is almost the same as the difference between the colors *razzmatazz* (ff0055) and *coral* (ff8855), the other may assess these differences quiet incomparable, both in quality and quantity (fig. 1). Alongside the different ways of perception, people use various terminology for describing colors. These terminologies differ due to the various semantic structure of languages and also individuals' experiences and preferences.

Mentioned reasons make it difficult for people to communicate colors through natural language and without unintuitive references like digital codes.

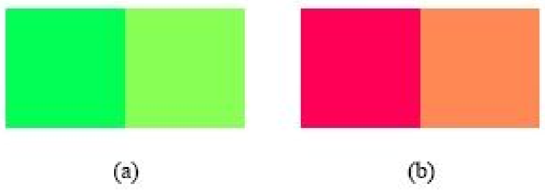


Fig 1. (a) Malachite (left), screaming green (right) (b) razzmatazz (left), coral (right). Based on the RGB color model the color difference in (a) is exactly the same as in the (b); both qualitative and quantitative.

It is an impractical goal to teach someone to make references to colors, accurately and in an objective way. The task is also impossible due to the inherent deficiency of the natural languages' vocabularies, that neither capture a full range of visible light spectrum nor a complete set of light features. However, using the color paradigm to push the limits of one's language skills to overcome the utter subjectivity of color perception in an effort for communicating, has its own undeniable learning benefits.

Color Companion is a software dedicated to such a goal. The software is a game developed on the concept of color description, playable by anyone with a fair command of English language (mostly for understanding the instructions) and, with regard to age, capable of understanding the concept of colors, distinguishing the difference between them, and expressing their ideas in written form.

The software's original code was written in the object-oriented programming language, Python version 3.7.4 [1]. And the MIT App Inventor (MAI) [2] was considered for the software's GUI. MAI is a web-based platform for creating apps for Android and iOS, designed to be operable for users having minimum levels of programming skills. However, the technical issues for processing natural language (the user) input using the limited coding capacities provided by the MAI's Block-Coding environment, served as a good reason for a change of plan. The extension of the software delivery deadline made it possible to consider various other options for the software's user interface. Among all the options, Unity, a C#-based game engine for developing 2D, 3D, and VR environments [3] was a feasible option, considering both the developer experiences and Unity's wide range of designing potentials which serves perfectly the aesthetic purposes.

Solution Design

Color Companion is designed to pressure the user to express their mental image through the (English) natural language. It may appear like a basic and trivial task for anyone engaging in a simple daily conversation. But there are three major differences in which, lie the educational values of the software: objectivity, fluency, and accuracy.

First of all, the communication partner is a machine, therefore the objective language is not merely an option. That is, the user has to express themselves clearly to avoid misunderstandings caused due to the machine‘s lack of common sense.

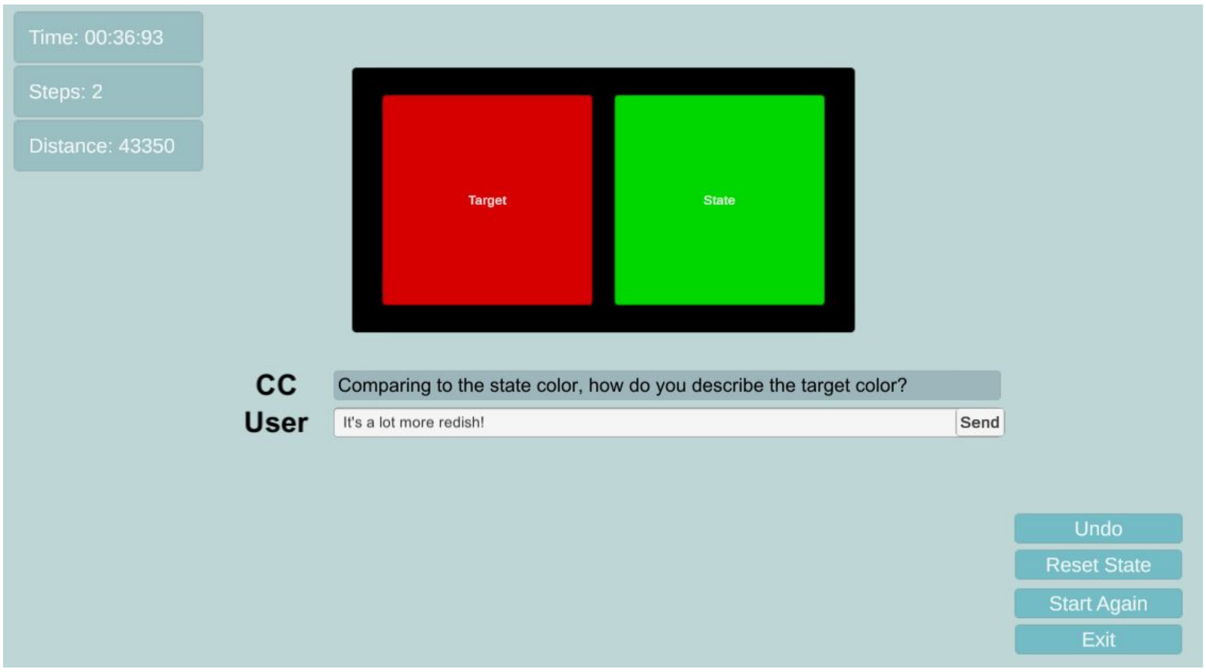
Secondly, the conversation topic is, not colors, but the relationship between two colors. This is of great importance for the purpose of the software because talking about a single color is only a 'coining problem,' which needs the user to coin or use words assigned to a specific wavelength (or digital code). But trying to explain the difference between two colors pressures the user to dig down their vocabulary to connect their perception, the best, to an almost objective term, comprehensible by the machine.

Thirdly, the game is designed in a time-limited format with an extra step-counter scheming. That is the user cannot afford tiptoeing around the problem and needs to be careful about choosing the right and most accurate words that connect two colors the fastest.

Implementation

Color Companion is a game designed by the Unity game engine. The game consists of 5 scenes: starting with displaying the instructions, picking the target color, picking the start color, modulating the state color, and ending with displaying the results. In the first scene the user is presented with the game's introduction and instructions. In the next scene, the user is asked to pick up a target color, from a palette of ## color tiles. The target color is the color that the user has to instruct the machine to reach. In the third scene, the user is asked to pick up one of the three primary colors of the RGB model as the start color. The user is instructed to pick a start color which to them seems closer to the target color.

In the fourth scene, which is the main game scene, the user is supposed to instruct the machine to change the state color, step by step, to get closer to the target color (fig. 2). The first state color is the start color. The user also has the option to reset the state color to one previous step or the start color. User's instructions to the machine are supposed to be in natural language and in the form of a comparison between the target color and the current state color, and in response to the machine's question: '*Compared to the state color, how do you describe the target color?*.' The user is required to include an adjective (the quality of the difference) and an adverb (the quantity of the difference) in their answers; although, the adverb is optional and in case of a missing adverb, the change by the machine would be applied at the minimum level. And the user has 5 minutes to lead the machine to reach the target color.



Comparing to the state color, how do you describe the target color?

It's a lot more redish!

Fig 2. The main game scene

Winning condition will be satisfied when the state color gets close enough to the target color. Colors are represented by the RGBA model and the distance between them is calculated with MSD (Mean Squared Deviation), that is the average of squared differences of the four channels: red, green, blue, and the alpha (transparency); however, for the sake of simplicity, the result is presented rounded to the closest integer. The maximum distance for winning is set based on the smallest possible adjustment unit, which is 20 out of a possible range of [0-255] for the color channels and [0-256] for the alpha channel. So whenever the state color reaches a to a unitless distance of 361, the user wins.

However, for applying the desired changes, the software does not parse the user command on a semantic level. Rather it has two dictionaries including sets of possible words and terms (## adjectives and ## adverbs) that will be scanned with each user entry to find any occurrences. The key values in the adjectives dictionary are the adjectives in the string format and the values are tuples consisting of four integer values, each representing the corresponding change to a channel in the RGBA model. For example, the adjective dictionary has an entry for the adjective *dark*, for which the corresponding value is *(0, -1, -1, 0)*. That means every time the user describes the target color darker than the state color, the software will reduce the green and blue channels' values of the state color and leave the red and alpha channels unchanged. The change tuples are defined experimentally by a small test group of 4 persons.

The quantity of the changes is decided upon by the adverbs. The software classifies its adverb set in three groups of *minor*, *moderate*, and *major*, and has one entry for each group. The dictionary's keys are the corresponding factors of 20, 40, and 60, quantizing the three classes of adverbs based on the amount of the desired change. The dictionary's values are lists of the adverbs, in the string format. Back to the previous example, if the user describes the target color 'a little darker' than the state color, the software recognizes the word *darker* as the adjective and retrieves the corresponding change tuple (0, -1, -1, 0) and it also finds the term *a little* as the adverb, in the class 'minor,' therefore the adverb factor will be set as 20. Then, the desired change will be calculated as follows:

$$\begin{aligned} change &= adv \times adj \\ &= 20 \times (0,\ -1,\ -1,\ 0) \\ &= (0,\ -20,\ -20,\ 0). \end{aligned}$$

(eq. 1)

And finally, the *change* will be added to the current state color's RGBA code to generate the new state color. In a case that the new state color code falls out of the valid range of RGBA code (e.g. (0, 0, 300, 0)), the user will receive a message stating that the action is not possible. And when the machine cannot find any of its stored adjectives in the user's input, it will ask for clarification. But the absence of an adverb will be ignored by considering the request for a minor change.

And the fifth scene will launch after 5 minutes, or when the state color reaches the maximum acceptable distance for winning. In this final scene, the results will be presented by the spent time (meaningful in case of winning), the distance (mostly meaningful in case of a losing) and the number of steps which is the number of times the user changed the state color before the time is up or before winning.

Summary

Considering the current version of Color Companion as a prototype, it fulfills what it promised. It is a simple game, anyone with an intermediate level of skill in English language who has some basic understanding of the colors can play. Besides the mentioned benefits of the software in leading the user to articulate their visual perception discussing a very subjective topic, it also gives them the opportunity to reinforce their color intuition.

But then there, of course, are major shortcomings. The initial idea of the project was a two-sided learning scenario; for the machine, as well as for the user. The idea was to provide the software with some moderate natural language processing skills, such that it can expand its adjectival and adverbial vocabularies by learning unprecedented phrases it finds in the user's entries. That is, in case of not recognizing any of stored adjectives, the machine conjectures the adjectival term in the input and proceeds the conversation by asking the user to define the term either by synonyms known to the machine, or by entering a corresponding change tuple. Such a design, also could turn the Color Companion to a unique tool for collecting demographic aspects of color perception. But the task turned out to be practically impossible considering the trade-off between the computational power of the software on one hand, and its GUI aesthetics on the other hand. Even though it is not impossible, per se, to have a good balance of aesthetics and usability, but the developer was not successful to find a suitable platform serving that purpose. While Python language has very powerful libraries for processing natural language, there are no user friendly platforms for GUI design (like Unity) integrated with those libraries.

Another important niche to work on is the software's change tuples. The tuples were defined experimentally and with a tiny set of only 4 persons as the referees. While it could be the subject of a complete and thorough study aiming at translating color perceptual features to RGBA code. Though we already have color models based on human perception (e.g. HSV) and there are mathematical models for conversion between the two systems, we are still lacking a translation system that chains the natural language to any of these color models. One improvement of the software, which can serve experimental purposes the best, could be a pre-test of the user to store a color perceptive profile of the individual and act as the software calibration.

There are also some minor improvements possible. For example, as explained, the new state color is the sum of old state color and the *change*, but that addition may result in values out of the valid range of RGBA code. In this case, the prototype approach is to simply throw an error of impossible change. While it would be optimal and also logically possible to handle the situation utilizing the color perception intricacy. Returning the example, if the user describes the target color darker which has and

References

1. Python Release Python 3.7.4, https://www.python.org/downloads/release/python-374.
2. MIT App Inventor, https://appinventor.mit.edu.
3. Unity (Game Engine), https://unity.com.

- **+add options for the target color (and/or make them randomly selected) +adjustable game difficulty by lowering the maximum acceptable distance, which also will affect the unit of changes +add instruction about how the answer should be in watermark under the input field. +for values exceeding the valid range is also possible to use alternatives rather than error message +*minor*, *moderate*, and *major*, instead of milli, moderate, and mega.**

Every group finishes implementation, creates a demo video (4 minutes) and writes a report (6 pages).