Color Sleuth and the AP Create PT

How the Color Sleuth Project Meets the AP Create Performance Task Requirements

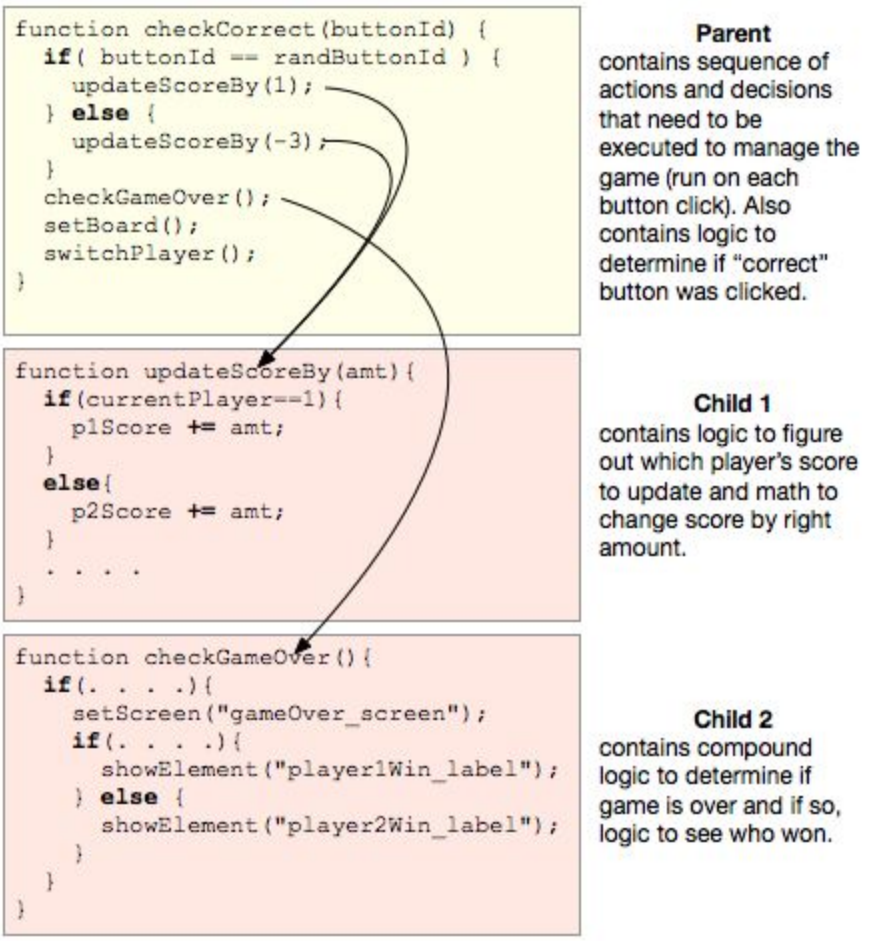
The Color Sleuth App, written as suggested in this lesson, is an example of a program that can meet the minimum bar for the AP Create Performance Task. Here's how.

Iterative Design Process (rows 2-3 of AP Create Task Scoring Guidelines) - for the AP you must discuss your overall "incremental and iterative development process" as well as two points along the way where you saw an opportunity, or some difficulty, that you worked out and it ended up in the final program.

Alexis and Michael's discussion throughout the tutorial is an excellent example of working collaboratively to iteratively write a program - they wrote in small parts, testing each part along the way, modifying it, or adding new functionality. The realization to use a parameterized function is a good opportunity to talk about. And any time they re-organized the code or changed their course of action is a response to some difficulty they were trying to overcome.

Algorithms (rows 4-6) - For the AP you need to show code of an algorithm that includes two or more algorithms where at least one of the included algorithms contains mathematical or logical concepts.

For a program structured like Color Sleuth, the main algorithm (or "parent") and included algorithms ("children") will likely be spread out across separate functions. An example of a choice that could be made along with arguments for the written responses is shown in the diagram. Note: the student would select all three of these functions as the "algorithm".



Abstraction (rows 7-8) - for the AP the code must contain a student-written abstraction that helps manage the complexity of the program. The functions in this program are strong evidence of using abstraction to manage complexity in the code.

A function with a parameter is often a good one to choose because the fact that the function has a parameter means that the problem has been abstracted so it can handle different types of input. `checkCorrect(buttonId)` and `updateScore(amt)` would be good choices that you should be able to justify easily in the written responses about how they help manage complexity.