2a. Provide a written response or audio narration in your video that:
identifies the programming language; identifies the purpose of your program; and explains what the video illustrates.
(Must not exceed 150 words)

The programing language I used is Scratch. The program's purpose is to test reaction times through playing "Reaction" or to allow users to increase mouse dexterity and reaction time through practice by playing "Targets". The video demonstrates both "Reaction" and "Targets". "Reaction" waits between 2 and 10 seconds before showing the user a green screen saying "Go!" which the user should click as quickly as possible to test their reaction time. "Targets" is a multi-difficulty game where the user should click targets as soon as the target appears. Once clicked, the reaction time and "precision" (how close their click was to the center of the target) for this target are recorded and later used to calculate data about your performance. After the game ends, all of the collected and calculated data is shown on the results screen for the user to see.

2b.
Describe the incremental and iterative development process of your program, focusing on two distinct points in that process. Describe the difficulties and/or opportunities you encountered and how they were resolved or incorporated. In your description clearly indicate whether the development described was collaborative or independent. At least one of these points must refer to independent program development. (Must not exceed 200 words)

While coding "Reaction", I noticed that Scratch's built-in "timer" displayed time to tenths of a second, which isn't fast enough to calculate reaction time accurately. After some research, I learned of a "days from 2000" timer that uses the days from 2000 to calculate time down to a millisecond. I chose to implement this approach instead to calculate reaction time because it was both more consistent (gives the number of days since 2000) and more accurate than the regular Scratch timer. Right after completing "Reaction", I tested the program and it seemed too uninteresting and boring with only one thing to do, therefore leading to the opportunity of developing the "Targets" game to satisfy that.

One difficulty encountered in "Targets" was that the player could easily get fast reaction times by holding down the mouse button and dragging the cursor over targets to activate them. I independently resolved this by using a call that activates when a sprite is clicked that I previously had not known about. This was far more optimal than my previous code that repeatedly checked for when both the mouse button was pressed and the cursor was over a target.

2c.

Capture and paste a program code segment that implements an algorithm (marked with an oval in section 3 below) and that is fundamental for your program to achieve its intended purpose. This code segment must be an algorithm you developed individually on your own, must include two or more algorithms, and must integrate mathematical and/or logical concepts. Describe how each algorithm within your selected algorithm functions independently, as well as in combination with others, to form a new algorithm that helps to achieve the intended purpose of the program. (Must not exceed 200 words)

After successfully completing either "Reaction" or "Targets", the data is processed and displayed by the "displayResults" algorithm. The "displayResults" algorithm allows the user to get useful data about their performance in the selected activity. If the user completes "Targets", "displayResults" will display a formatted list of the minimum and maximum precision and reaction time values and average precision and reaction time of the user. It does this by first going to each calculated value's respective function to determine the data value logically or mathematically and then adding the returned value to the displayed output list after formatting. Otherwise, if the user instead completes "Reaction", "displayResults" will instead format their reaction time of that test and display it in a text bubble.

Each of the calculation sub-algorithms for "Targets" does what their name suggests by incrementing through its respective list of data while performing logical or mathematical calculations depending on the value that will be determined. For example, "calcAveragePrecision" would increment through the precision data list to get all precision values, then average all of them to calculate average precision. After the sub-algorithm calculates its answer, it automatically adds the value to the displayed output list for the main algorithm.

2d.

Capture and paste a program code segment that contains an abstraction you developed individually on your own (marked with a rectangle in section 3 below). This abstraction must integrate mathematical and logical concepts. Explain how your abstraction helped manage the complexity of your program. (Must not exceed 200 words)

One of the abstractions used in my code, "setupTarget", is called every time a new target clone spawns in "Targets". Using the user's selected difficulty level and variables controlling the margins for the spawning zone, "setupTarget" places the target clone randomly within the game space excluding the marginal space and sizes the target clone according to the difficulty level. Once this is completed, the target is ready to be used in the "Targets" game. By implementing this abstraction, the readability of the code is increased and complexity is minimized as all thirteen blocks currently used to setup the target are condensed into a single function that can be called every time a new target is created. Not only this, but condensing the code also allows the

programmer to easily debug the target's code and implement new target features in the future. Without this vital abstraction, the main target clone code would be significantly longer, more complex, and more difficult for anyone to understand.

Sources:
http://pluspng.com/img-png/target-symbol-image-4534-1323.png